MARIAN

A hybrid, metric-driven, agent-based routing protocol for multi-hop ad-hoc networks

Nikos Migas

DSMA, Napier University,
n.migas@napier.ac.uk

Abstract

Recent advances in technology provided the ground for highly dynamic, mobile, infrastructure-less networks, namely, ad-hoc networks. Despite their enormous benefits, the full potential cannot be reached unless certain issues are resolved. These mainly involve routing, as the lack of an infrastructure imposes a heavy burden on mobile devices that must maintain location information and route data packets in a multi-hop fashion. Specifically, typical ad-hoc routing devices, such as Personal Digital Assistants (PDAs), are limited in respect to the available throughput, life-time, and performance, that these may provide, as routing elements. Thus, there is a need for metric-driven ad-hoc routing, that is, devices should be utilised for routing according to their fitness, as different device types significantly vary in terms of routing fitness. In addition, a concrete agent-based approach can provide a set of advantages over a non-agent-based one, which includes: better design practice; and automatic reconfigurability.

This research work aims to investigate the applicability of stationary and mobile agent technology in multi-hop ad-hoc routing. Specifically, this research proposes a novel hybrid, metric-driven, agent-based routing protocol for multi-hop ad-hoc networks that will enhance current routing schemes. The novelties that are expected to be achieved include: maximum network performance, increased scalability, dynamic adaptation, Quality of Service (QoS), energy conservation, reconfigurability, and security. The underlying idea is based on the fact that stationary and mobile agents can be ideal candidates for such dynamic environments due to their advanced characteristics, and thus offer state of the art support in terms of organising the otherwise disoriented network into an efficient and flexible hierarchical structure, classifying the routing fitness of participating devices, and therefore allow intelligent routing decisions to be taken on that basis.

Results derived from the experimentation phase proved that resource-constrained devices are significantly inferior to high-end devices in terms of routing data, buffering capabilities, and performing complex calculations, processes which are essential by any router. In addition, these processes were shown to have a significantly negative effect in relation to battery discharge rate, CPU utilisation, and memory usage. These facts were seriously taken into account in the modelling phase, where each device was defined so as to be required to determine several routing fitness metrics, which relate to various routing scenarios, and recalculate these each time a change occurs, e.g. battery drops, CPU utilisation increases. Accordingly, the protocol can use this information in order to dynamically adapt its routing decisions, and thus always retrieve the optimal routing paths that best suite the routing scenario that is required to accomplish.

Keywords: **Multi-hop ad-hoc routing, wireless ad-hoc networks, stationary and mobile agents, clustering, routing, QoS, automatic network reconfiguration, security, intelligent filtering**.

Table of Contents

1 Introduction

1.1 Prologue

Wireless Ad-hoc networks can be extremely important in situations, such as counter-terrorism and rescue operations. This is especially true, when fixed networks become suddenly unavailable, while their services are crucial for human lives. An example can be found in a hospital's emergency department, where the central computer network has gone down because of some failure, and thus doctors cannot retrieve patients' history records. Instead, ad-hoc networking can be used as an alternative, temporary solution, until the problem is repaired. This chapter introduces the research area of this thesis, and highlights important concepts. It then discusses the research aims and novelties. Finally, the structure of the thesis is presented.

1.2 Introduction to the research area

This thesis involves two major research areas: multi-hop wireless ad-hoc routing; and intelligent stationary and mobile agents. Background information on the definitions of concepts, which are involved with this thesis, is provided in Appendix B. This section only provides an introduction to these areas, and highlights the most important concepts.

Recent advances in technology have introduced a wide range of devices, with different performance characteristics, and the ability to remotely communicate without the need of a fixed infrastructure. Networks of this category are commonly known as ad-hoc networks, which can be generally defined as a collection of geographically distributed nodes that often communicate in a multi-hop fashion, each of which is responsible for location management and data routing (Wang, X., et. al., 2001, Liu, J., et. al., 2002). Mobility is normally an important feature of ad-hoc networks, and thus provides the ability for users to communicate, cooperate, and access the computer services in an *anytime-and-anywhere* manner (Frodigh, M., et. al., 2000). Ad-hoc networks have been proposed as a networking solution for situations where the network setup time is a major constraint, and/or where a network infrastructure is either not available, or not desirable (Ramarathinam, V. and Labrador, M. A., 2002). In real-life, applications of ad-hoc networks can be found in military, rescue, and antiterrorism operations, whereas some commercial ones include: conferencing; sensor networks; personal area networks; and embedded computing applications (Perkins, C. E.,

2001).

Despite the theoretical benefits of ad-hoc networks, there are certain constraints that limit the potential implementation of this technology. Routing is, for example, considered to be the most challenging one. This is due to the fact that participating devices are responsible for obtaining and maintaining routing tables, and actually route data packets, as there is no fixed infrastructure. Furthermore, as the majority of these devices are mobile, the network becomes highly dynamic, and routes that maybe considered good at a given time may become unavailable or undesirable, at a later time. In addition, mobile devices rely on battery power, which is shown to rapidly decrease while performing process intensive tasks, such as routing (Migas, N., et. al., 2005, Migas, N., et. al., 2004b).

The mobile agent paradigm is a relatively new technology that has its origins in intelligent agents, and has been proposed as an alternative approach to the client-server communications model. A mobile agent is a software entity that usually inherits several features of intelligent agents, and requires a mobile agent system for its execution. It differs from a stationary agent, as it can suspend its execution on a host computer, and then transfer its code, data state, and, possibly, its execution state (strong migration) to another agent-enabled host on the network, and resume execution on the new host. Overall, the aim of a mobile agent system (Silva, A. R., et. al., 2001) is to provide the appropriate functionality to stationary and mobile agents in order to execute, communicate, migrate, and use system resources in a secure way. Their application include: information retrieval (Cardi, G., et. al., 2000); e-commerce (Lee, T. O., et. al., 2001); network management (Marques, P., et. al., 2001); intrusion detection (Spafford, E. H. and Zamboni, D., 2000); collaborative applications (Wong, D., et. al., 1997); and, most importantly, mobile computing (Kotz, D., et. al., 1997). Although each application can be implemented with the existing technologies (Harrison, C. G., et. al., 1995), the use of mobile agents can contribute to build these distributed applications in a simpler and more effective manner (Puliafito, A., et. al., 2000).

It has been argued that traditional routing mechanisms in multi-hop ad-hoc networks are inappropriate because of: low bandwidth; dynamic topology; and, limited connection survivability (Hassanein, H. and Zhou, A., 2001, Yi, Y., et. al., 2002, Jiang, M.-H., et. al., 2002). Fortunately, agent-based approaches could be beneficial in environments, which are mainly characterised by low-bandwidth, high-latency, and unreliable network links (Vinaja, R., 2001). This is due to the fact that mobile agents offer two major advantages in comparison to traditional client-server approaches, such as task continuation and minimal connection use (Kotz, D., et. al., 1997). Various projects have thus applied this paradigm to ad-hoc routing (Minar, N., et. al., 1999, Chpudhury, R. R., et. al., 2000, Marwaha, S., et. al., 2002) with reasonable success (*see* Chapter 2).

1.3 Identification of the Problem Area

In any ad-hoc network, the most important function is considered to be routing. However, routing is a challenging issue, because of:

- Ad-hoc networks are infrastructure-less, that is, they do not rely on fixed hardware to perform routing tasks. Thus, it is up to mobile devices to derive and maintain routing tables, and to actually route data packets.
- Routers (*mobile devices*) in such environments are mobile, and thus can dynamically change their placement in an ad-hoc network, or even disconnect from it.
- Routing requires processing power that mobile hosts running on batteries may not be able to provide.

Furthermore, due to the nature of mobility, a route that can be considered as optimal at a given time, may break, or not be optimal enough some time later. Therefore, a great deal of research has been concentrated on how to design and implement efficient routing protocols.

In addition, different network traffic types impose diverse routing requirements. For example, real-time traffic requires high buffering capabilities and low latency, while asynchronous traffic has no such requirements. Therefore, a traditional shortest-path routing algorithm would fail to address this issue, as a shorter route may not be capable of accommodating a particular routing scenario, while another route, despite being longer, could actually accomplish the routing task efficiently. Figure 1.1 illustrates an ad-hoc network topology, where node *A* wants to transmit three different types of network traffic to node *B*.

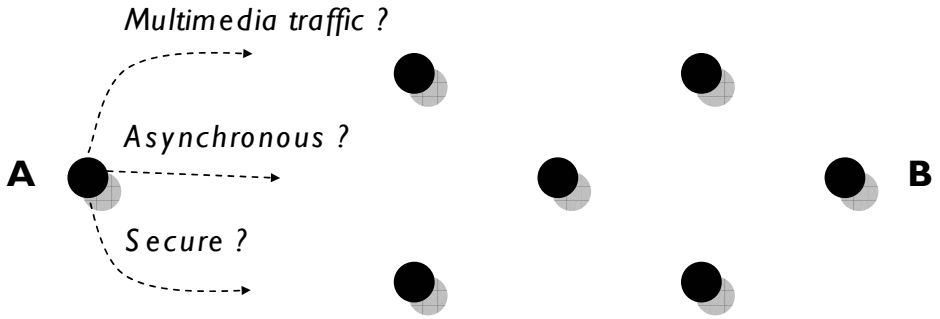

Figure 1.1: An ad-hoc network topology with different routing scenarios

Based on the shortest-path routing algorithm, all traffic types, as shown in Figure 1.1, would be routed through the middle route, which consists of the smaller amount of hops. This decision, however, would not take into account the performance characteristics of the

intermediate devices, nor their utilisation or battery status.

Another major concern in ad-hoc networks is security. This is mainly attributed to the ubiquitous nature of the wireless medium that makes it more susceptible to security attacks than in wired communications. Information can be easily intercepted by anyone that is equipped with a relatively simple wireless device. In addition, the routing process can also become maliciously altered by radio jamming and battery exhaustion (Stajano, F. and Anderson, R., 2000). Thus, it is vital that appropriate security countermeasures are taken into consideration during the design process of an ad-hoc routing protocol. However, security is rarely addressed by most to-date routing protocols.

1.4 Research challenges and context

Based on the identification of the problem area presented in Section 1.3, an interesting research question arises:

Can an agent-based system be used in wireless ad-hoc networks in such a way so as to classify the routing fitness of each participating device based on key metrics, which include: remaining battery life; utilisation status; maximum throughput; buffering capabilities; and, network error; and then weight it appropriately to suite various routing objectives, such as energy efficient; synchronous; asynchronous; critical; secure; and, burst network traffic?

A scenario is used throughout this section to explain, in more detail, the general research question presented above. A simplified case is considered, where *A*, *B*, and *C* are mobile nodes and belong to wireless domains WD_A, WD_B, and WD_C, respectively (Figure 1.2). Suppose that WD_A and WD_B intersect ($\cap$) and that WN_B and WN_C intersect as well. Therefore, *A* can see *B* as an adjacent node, and vice-versa, and *B* can see *C* as an adjacent node, and vice-versa. Unfortunately, *A* cannot see *C* and vice-versa, and not any other node that belongs to a network that does not intersect with its own. This is commonly known as the hidden node problem. In a multi-hop ad-hoc network this problem is common, and is usually solved by means of flooding the network with Route Request packets (RREQ) in order to retrieve a route to a desired destination. However, this method is highly inefficient in terms of network overhead, and thus an alternative solution is desirable:

Can mobile agents efficiently exploit a hierarchically structured multi-hop ad-hoc network in order to collect the network's topology information, along with the routing metrics involved, in such a way so as to minimise the total number of required migrations, and to provide resil-

ience in dynamic topology changes, and further reduce the amount of information transmitted by enabling redundant information filtering?

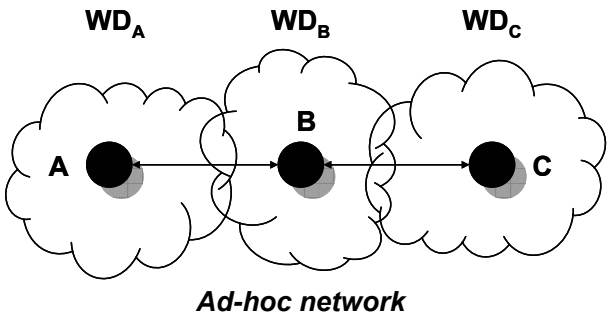

Figure 1.2: Discovering Network Neighbourhood

It is possible that a node could create a network topology gathering mobile agent and dispatch it to each of its neighbouring nodes. The newly instantiated mobile agent could then examine its neighbouring nodes, and if appropriate, create a number of clones, equal to the number of neighbouring nodes, and dispatch them. The process could then iterate until the complete network topology is collected, or until the specified threshold limit of allowed migrations is reached. Mobile agents could then migrate back to their original platform, pass the topology information to their parent agent, and kill themselves. Their parent agents could repeat the process, until all information is returned to the grandparent mobile agent, that is, the original mobile agent that initiated the network discovery process. Figure 1.3 presents an illustration of this hypothesis in a network topology information gathering example using mobile agents.

Based on the Figure 1.3, mobile node *A* could create a network discovery mobile agent, which could then examine node's *A* neighbouring nodes, and thus create and dispatch a cloned mobile agent to each of *A*'s neighbours (*B* and *C*). The process could then iterate until the complete network topology is gathered. However, there are various challenges involved, mainly because of the lack of an infrastructure that could assist the agents in deciding on the appropriate next hop in such as way so as to guarantee that nodes are never visited more than once, and, in addition, to enable the minimum possible total number of migrations. Furthermore, in large ad-hoc networks the relevant information can be large, and thus agents will require some sort of filtering capabilities that would enable them to discard redundant irrelevant routing information, and, consequently reduce their migration times.

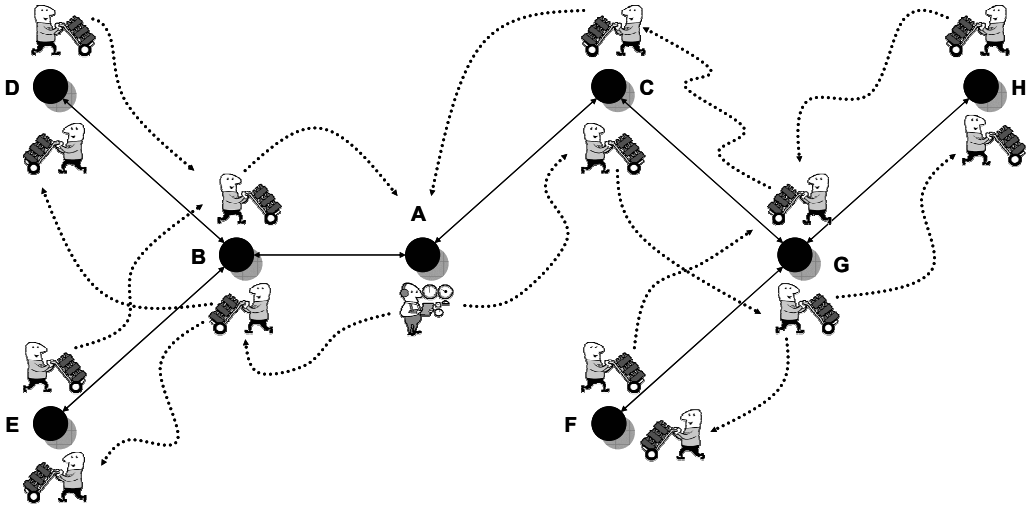

Figure 1.3: Network discovery using mobile agents

An ad-hoc network with six mobile nodes forming a circular network topology is presented in Figure 1.4. It is assumed that every mobile node is aware of the existence of all other nodes in the network, and also knows the routes to get to them. Thus, network discovery is assumed to have taken place, and the nodes haven't moved since. In this example, nodes *A*, *B*, *C*, *D*, *E*, and *F*, belong to wireless domains WN_A, WN_B, WN_C, WN_D, WN_E, and WN_F, respectively. Suppose that $WN_A \cap WN_B$, $WN_B \cap WN_C$, $WN_A \cap WN_D$, $WN_D \cap WN_E$, $WN_E \cap WN_F$, and $WN_E \cap WN_F$, thus forming a circular topology.

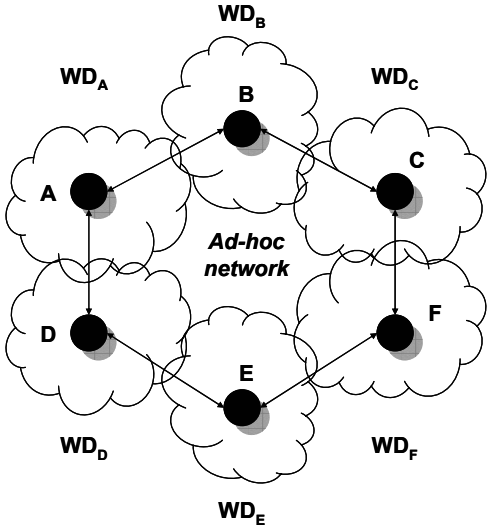

Figure 1.4: A circular ad-hoc network topology

In the case that mobile node B wants to communicate with mobile node E, it can pass network traffic through node A or node C, as WN_B, the mobile node B belongs to, intersects with WN_A and WN_C. A decision should be based on various parameters, such as the processing capabilities of mobile nodes A and C, as well as their utilisation status, remaining battery life, memory capacity, available throughput, and so on. In addition, this should be appraised in terms of the requirements imposed by the network traffic, which B indents of sending, and, in addition, to the application specific QoS requirements. Unfortunately, in current routing schemes, these considerations are typically not taken into account, and thus the decision is usually based on distance, measured in number of hops, that is, the *optimal* route is the one that involves the fewest intermediate nodes. This simplified approach can decrease scalability, availability, and performance.

Can intelligent agents conduct preliminary performance tests, and, in addition, monitor the device's available resources, and use this information in such a way so as to unambiguously determine the device's routing performance and dynamically adapt it to changing conditions?

In order to provide more evidence on the importance of this question, an example network topology is presented in Figure 1.5. According to the example, the source node S wishes to send some data traffic to the destination node D. The traffic can be routed by either one of the following available routes, $S \rightarrow B \rightarrow E \rightarrow D$; $S \rightarrow C \rightarrow F \rightarrow D$; and, $S \rightarrow D \rightarrow G \rightarrow H \rightarrow D$. A simplified approach, used in most traditional routing protocols, would be for node S to count the number of intermediate nodes (hops), and thus deduce the shortest route, which, in this case, would be the first and second route, as both are two hops away from the destination. The third route would be excluded from routing, as it has the highest number of intermediate nodes.

However this approach assumes that participating devices are of equal strength, that is, they can offer the same throughput, are equally reliable, have the same utilisation status and electrical power capacity, at any given time. In addition, this approach provides no support for multiple redundant paths, which could be efficiently used to maximise the network's performance, and therefore certain paths may become overburdened with routing requests, such as, in when the network topology presented in Figure 1.5 corresponds to the devices presented in Figure 1.6. As shown, the first most optimal route in terms of hops, is actually the worst route in terms of remaining battery capacity, CPU and memory utilisation, and available throughput (Kbits/s). Also, the previously considered *worst* route consists of devices which do not require battery power for their operation, have low utilisation status, and high available throughput.

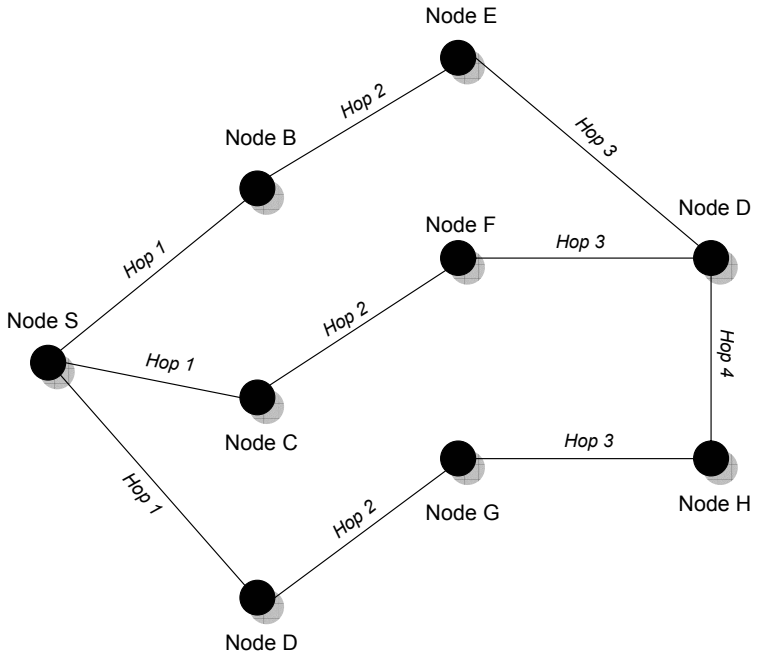

Figure 1.5: An ad-hoc random network topology

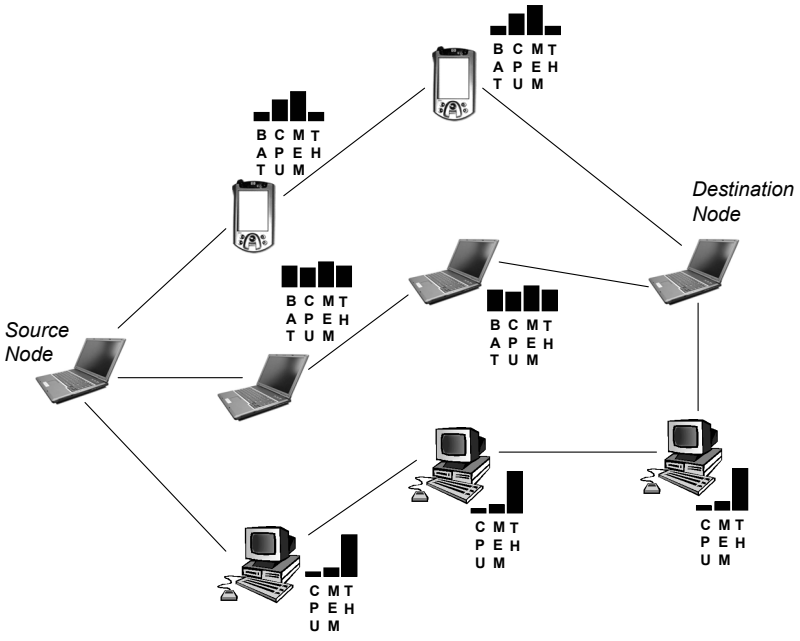

Figure 1.6: The underlying devices along with their performance capabilities and utilisation status

Therefore, an intelligent system would, preferably, judge upon the routing requirements of a

routing scenario, and assign the most appropriate route. For instance, heavy network traffic, in terms of requirements, would be routed through the third route, and, at the same time, lighter and medium network traffic, would be routed through the first and second route, respectively.

Is it possible to determine the capability, or, incapability of the routing fitness of each source route towards a single destination, by appraising it against the routing requirements imposed by each distinct routing objective, and further classifying it into an overall QoS routing metric?

Automatic network reconfiguration is an important element in ad-hoc networks. For instance, in a large wireless ad-hoc network, possibly consisting of a couple of hundred nodes, if the installation of routing protocol updates becomes necessary, then this typically involves updating each device manually. In contrast, automatic network reconfiguration could be achieved by allowing the software to be installed, automatically, without any manual control. In order for this to be partially achievable, the software has to have mobility capabilities, and thus migrate to every node in the network and automatically install itself.

Can mobile agent technology automate network reconfiguration, such as, updating the routing protocol currently running on participating ad-hoc mobile devices, without having to turn devices off, or gather them in a central place?

1.5 Research aims

The objective of this thesis is to develop new methods for routing in multi-hop ad-hoc networks that could be efficiently used in emergency and rescue operations, where participating devices of unequal strength could be utilised as routing elements, with fairness. In addition, this thesis aims to enhance current routing schemes by providing route redundancy, and metric-driven routing that will ultimately set a balance on utilisation factors between high-end and low-end nodes, as well as retrieving *optimal* routes in terms of various routing scenarios.

In particular, this thesis proposes to assess various agent-based models to determine the routing capability of each participating node and weight it against the required routing scenario. The determination of this route, though, is a complex one, and requires research into the best metrics to determine the best path, such as for processing capability, network performance, CPU utilisation, memory capacity, and battery reserves. This thesis shows that an

optimum model is to use a mixture of stationary and mobile agents to gather relevant information. These agents could perform important tests, which could be used to generate the best route through a network. It also analyses different models for the deployment of these agents, which balance the network nodes' important resources, and, most importantly, can be efficiently executed by resource-constrained devices, such as PDAs.

For this reason, a novel hybrid, metric-driven, agent-based routing scheme is proposed that provides the following novelties:

- **Metric-driven hierarchical structure**. Organises the otherwise disoriented network into a cluster-head structure, where fitter devices are elected as cluster-heads, and thus have clustering responsibilities within their own clusters, such as location management and routing, whereas, weaker devices are elected as cluster-head's members, and thus have minimum, or, no responsibilities. Key metrics that determine the nodal capability of being awarded a cluster-head role are: relative mobility; processing capabilities; network state; utilisation; and battery life. Therefore, it is ensured that re-clustering will occur at a minimum possible, as cluster-heads are not frequently moving, and, in addition, they are equipped with sufficient battery life to keep them operational for long enough. However, it is probable that changes will eventually occur, which might lead to critical situations, such as a cluster-head is running low on battery life, and are thus taken into account by allowing the system to dynamically adapt to important metric changes.
- **Route redundancy**. Supplies a source node with multiple routes, if present, towards a single destination for each initiated route discovery process. Along the same line, it supplies a cluster-head with multiple routes to each possible destination in the network for each initiated network topology gathering mobile agent. Multiple routes are kept in the node's routing cache, in case of non-cluster-head, whereas, in case of a cluster-head, they are maintained in the node's routing table, which is public for its own members. A transmitting node can thus use an alternative route in case of primary failure, hence avoid the initiation of a new route request. In addition, it can utilise more than one route for each required routing scenario, as it is likely that each route will have a distinct routing capability, and thus offer various levels of QoS.
- **Hybrid route and network discovery**. Provides an on-demand route discovery process that can be initiated by every node in the network, at a controlled manner, and, in addition, a proactive network topology gathering process that can be initiated only by cluster-heads based on either triggering events, or, in a periodic manner. Both processes benefit from the network's clustering formation, as the respective route request packets (RREQ) and mobile agents need only to traverse special nodes, and are thus not flooded to the

network, which results into decreased network overhead. This hybrid approach enhances the route availability at each cluster, which can be immediately employed, and therefore minimises the need for on-demand route discovery, which generally requires additional time to resolve.

- **Metric-driven routing**. Determines the capability, or, incapability of a node, and, consecutively, a route to accommodate a range of routing scenarios. Therefore, a route may be determined capable of accomplishing a certain scenarios, whereas incapable of accomplishing another. The more the number of routes supplied for a given destination, the more the likelihood of, at least one route, being determined as capable of routing a certain type of traffic. In the presence of two or more capable routes, the decision is based on two factors: the maximum level of QoS offered by each route; and, the level of QoS required. Therefore, a transmitting node requiring high level of QoS for a given routing scenario, decides on the route that offers the best services. The metrics are derived from performance and monitoring tests, which were performed throughout the experimentation phase, in an attempt to benchmark various devices as routing elements. These metrics include: complex calculation ability; buffering capabilities; maximum throughput; protocol error percentage; CPU utilisation; memory usage; and, remaining battery life. Once a node's overall metric is calculated, it is appropriately weighted to suite various routing objectives, and it is dynamically reconfigured in the presence of critical internal changes, such as rapid increases in the node's utilisation.
- **Protection from nodal over-utilisation and battery exhaustion**. A node's vital resources are protected, at all times, by unpleasant over-utilisation, and battery exhaustion. Specifically, a node with either high utilisation, or, low battery life, is exempted from the clustering formation, and routing duties, as it would otherwise result into an unreliable task accomplishment, or, even worst, it could lead into making the node unusable. This is achieved by incorporating key metrics, such as the node's utilisation status, and remaining battery life, into the metric-driven clustering formation and routing processes, and by further assigning a sufficient weighting so that rapid changes cause the overall metric to turn to routing and clustering incapability.
- **Benchmark the routing capability and resources consumption of various devices types**. In a wireless ad-hoc network it is likely that participating devices will have different hardware characteristics, and consequently different levels of performance when dealing with routing. A primary aim is to benchmark this capability and determine the performance of various devices, including: battery independent devices, such as workstations, and, battery dependent devices, such as laptops, and PDAs. Another equally important factor is the resources consumed throughout the routing process, this is, the CPU utilisa-

tion; the memory usage; and, the battery discharge rate.

- **Determine the underlying factors that may optimise routing tasks**. In addition to hardware-dependent, routing is also dependent on the underlying software, that is, the operating system (OS); and, the Java Virtual Machine (JVM), which is used to interpret the routing software. Therefore, routing optimisation is achieved by utilising the best software combination, which provides the highest throughput, and, most importantly, consumes the fewest resource.

1.6 Thesis structure

This thesis splits into the following six main chapters:

- Chapter 2 presents the literature review in the area of agent-based ad-hoc routing, and provides a synopsis on non-agent-based ad-hoc routing.
- Chapter 3 presents the Mobile Agents for Routing In Ad-hoc Networks (MARIAN) specification, which is a hybrid, metric-driven, agent-based routing protocol specifically defined for this research. It also presents the Benchmarking multi-Agent Software System (BASS), which is a multi-agent software system that assists MARIAN in metric determination.
- Chapter 4 presents the precise definition of experiments as well as the hardware and software used for their execution.
- Chapter 5 presents the experimentation results obtained throughout the research. It is composed by six sub-chapters, the preliminary experimentation phase, the proxy experimentation phase, the BASS experimentation phase, the experimentation of mobile agent migration phase, the metric simulation experimentation phase, and finally, a MARIAN-enabled network application scenario.
- Chapter 6 evaluates the research findings, and discusses the major contributions of this research by comparing these with similar research work.
- Chapter 7 concludes the thesis and presents the future work.

2 Literature Review

2.1 Problems & challenges of Ad-hoc networks: security, routing

Ad-hoc networks inherit the problems and challenges of wireless networks (*see* Appendix D.1), as they use wireless radio signals for communication establishment. In addition, two more fundamental challenging issues are: security; and routing.

The routing process is considered as the most important area of research in an ad-hoc network. However, routing in such a highly dynamic network is a challenging issue, as problems arise due to the nature of an ad-hoc network, such as the mobility and low-performance characteristics of mobile devices. Accordingly, as mobile devices are free to move in an arbitrary manner, performing routing is a challenging issue. Thus, a route which may be considered as the best path at the present, may break, or become non-optimal, some time later. The second reason is that, typically mobile devices, such as laptops and PDAs, running on batteries, and with low performance capabilities, cannot perform complex calculations that are necessary for the routing process. Thus, the majority of research in ad-hoc networks is concentrated on the design of energy-efficient routing protocols. Background information on ad-hoc routing is provided in Appendix D.

Security is also difficult to assure in ad-hoc networks due to the absence of an infrastructure, which typically does not allow the effective use of private- or public-key cryptography (Mohapatra, P. K., 2000, Buchanan, W. J., 2000). Security often plays a crucial role in ad-hoc networks, than, in infrastructure networks, that use wired communications, such as in LANs. This is mainly because of the ubiquitous nature of the wireless medium that makes it more susceptible to security attacks than in wired systems. Unfortunately, information sent over a wireless medium can be intercepted by anyone that is equipped with a relatively simple wireless device. Also, the sender of network traffic cannot be uniquely identified, as they can in fixed networks. Even further, eavesdropping or interference cannot be detected in a medium as ubiquitous as the wireless medium.

Although ad-hoc networking is more prone to passive eavesdropping attack, confidentiality is not the only, or even the main security requirement. Security properties for ad-hoc networks that need to be ensured include: availability, integrity, and confidentiality. Availability is the most important security property for most non-military applications of ad-hoc networks. Availability attacks include radio jamming and battery exhaustion (Stajano, F. and

Anderson, R., 2000). An attacker can deny services to mobile nodes of an ad-hoc network by jamming the radio frequencies they use, or interact with a mobile node in an otherwise legitimate way, but for no other purpose that to consume its battery capacity.

Integrity of a mobile node participating in an ad-hoc network ensures that the node has not been maliciously altered. It guarantees that a recipient receives the correct information from a genuine transmitter, and not from a node that has been modified to send out incorrect information. As already mentioned, cryptography is the most common and effective approach used to ensure integrity, but this cannot be applied to an ad-hoc network because of the lack of infrastructure. An alternative approach is to allow only tamper-proof (Anderson, R. and Kuhn, M., 1996) mobile nodes to participate in an ad-hoc network. However, this approach is difficult to implement, as the cost of tamper-proof mobile devices normally high.

Confidentiality ensures the secrecy of communications among participants in an ad-hoc network. It is tightly related with authenticity, which ensures that a mobile node communicates with the right participant. Authenticity is where the real issues are, and, once they are solved, protecting confidentiality is simply a matter of encrypting the session, using an appropriate encryption key (Stajano, F. and Anderson, R., 2000).

2.2 Comparison of routing protocols' main characteristics and complexity factors

The vast majority of routing protocols in the same category share many common characteristics. For example, some of the common characteristics of proactive protocols are: the frequency of updates broadcasted to the network; the routing structure; and the structure in which routing information is maintained. However, there are main differences in the overall performance, scalability, and ease of installation of these protocols. Table 2.1, summarises most of the basic features of each proactive routing protocol, which have been previously discussed.

Table 2.1 (Abolhasan, M., et. al., 2004): Proactive routing protocols, basic characteristics

Protocol	RS	No. Tables	Updates frequency	HM	CN	Characteristics
DSDV	F	2	Periodic & as required	Yes	No	Loop-free
WRP	F	4	Periodic	Yes	No	Loop-free
GSR	F	3, 1 list[a]	Periodic and local[b]	No	No	Localised updates
FSR	F	3, 1 list	Periodic and local[b]	No	No	Controlled frequency updates

STAR	H	1, 5 lists	Conditional[c]	No	No	LORA/ORA, Minimum CO
DREAM	F	1	Mobility-based	No	No	Controlled rate of updates by mobility/distance
MMWN	H	Database	Conditional	No	Yes, LM	LORA, Minimum CO
CGSR	H	2	Periodic	No	Yes, CH	CH exchange routing information
HSR	H	2[d]	Periodic, within each subnet	No	Yes, CH	Low CO, Hierarchy
OLSR	F	3	Periodic	Yes	No	MPR, reduced CO
TBRPF	F	1, 4 lists	Periodic and differential	Yes	Yes, PN	Broadcasting topology updates over a spanning tree

RS (routing structure); **HM** (*HELLO* messages); **CN** (Critical nodes); **H** (hierarchical); **F** (flat); **CO** (control overhead); **LORA** (least overhead routing approach); **ORA** (optimum routing approach); **LM** (location manager); **CH** (cluster-head); **PN** (parent node); **MPR** (multipoint replaying).

[a]GSR also has a list of all available neighbours.

[b]In GSR and FSR link-state is periodically exchanged with neighbouring nodes.

[c]In conditional update methods, the updates occur if a particular event occurs.

[d]Number of link-state tables may vary according to the number of logical levels.

The routing structure (RS) is referred to the ad-hoc network's structure. In the case where the network has no logical structure, the routing structure is supposed to be flat, while, if some sort of organisation is imposed into the network, the routing structure is hierarchical. Usually flat addressing is the most normal case for proactive routing protocols, due to its implementation simplicity. However, a flat structure suffers from frequent broadcasted control packets, along with their prolonged propagation into the network. A possible solution to this problem is the use of GPS (Kaplan, E. D., 1996) on each device on the network. For instance, DREAM obliges each ad-hoc device to be equipped with GPS, and thus the amount of information carried into the control packets is significantly minimised. Thus, nodes only exchange location information, such as geographical coordinates, instead of complete link-state, or distance vector information. However, GPS comes at a certain cost, and may, therefore, be unaffordable to some users. An alternative to GPS, which may achieve similar results, is the use of conditional updates rather than periodic ones, as in the case of STAR, which allows the dissemination of updates only when certain conditions occur. Routing protocols which use a logical hierarchical structure are also known to reduce the control overhead by localising the propagation of update messages. However, there is often a price to pay in terms of network overhead involved with structure maintenance. In addition, hierarchical protocols often require the use of critical nodes, which may become bottlenecks. Table 2.2 summarises the convergence time, memory overhead, control overhead, advantages and disadvantages, of each of the key proactive routing protocols, which were previously discussed.

Table 2.2 (Abolhasan, M., et. al., 2004): Proactive routing protocols, overhead

Protocol	CT	MO	CO	Advantage/Disadvantage
DSDV	O(D·I)	O(N)	O(N)	Loop free/high overhead
WRP	O(h)	$O(N^2)$	O(N)	Loop free/memory overhead
GSR	O(D·I)	$O(N^2)$	O(N)	Localised updates/high memory overhead
FSR	O(D·I)	$O(N^2)$	O(N)	Reduces CO/high memory overhead, reduced accuracy
STAR	O(D)	$O(N^2)$	O(N)	Low CO/high MO and processing overhead
DREAM	O(N·I)	O(N)	O(N)	Low CO and MO/requires GPS
MMWN	O(2D)	O(N)	O(X+E)	Low CO/mobility management and cluster maintenance
CGSR	O(D)	O(2N)	O(N)	Reduced CO/cluster formation and maintenance
HSR	O(D)	$O(N^2 \cdot L)$ + O(S) + O(N/S) + O(N/n)	O(n·L)/I + O(1)/J	Low CO/location management
OLSR	O(D·I)	$O(N^2)$	$O(N^2)$	Reduced CO and contention/2-hop neighbour knowledge required
TBRPF	O(D) or D + 2 for link failure	$O(N^2)$ + O(N) + O(N+V)	$O(N^2)$	Low CO/High MO

CT (convergence time); **MO** (memory overhead); **CO** (control overhead); **1** (a fixed number of update tables is transmitted); **V** (number of neighbouring nodes); **N** (number of nodes in the network); **n** (average number of logical nodes in the cluster); **I** (average update interval); **D** (diameter of the network); **S** (number of virtual IP subnets) **h** (height of the routing tree); **X** (total number of LMs, one location manager for each cluster); **J** (nodes to home agent registration interval); **L** (number of hierarchical level); **E** (endpoint nodes).

In relation to Table 2.2, the overhead imposed by most routing protocols in this category is significantly high. In particular, the memory overhead is significantly high, as each node needs to store and maintain routing information concerning every other node in the ad-hoc network, and that each node is required to transmit its complete routing tables. An exception to this may be DREAM, for the aforementioned reasons. Unfortunately, high overhead often results in scalability constrains. Thus, proactive routing protocols often do not scale well in large ad-hoc networks, with a limiting nodes set at around 100. Table 2.3 presents a summary of each of the key reactive routing protocol's main characteristics.

Table 2.3 (Abolhasan, M., et. al., 2004): Reactive routing protocols, basic characteristics

Protocol	RS	Multiple Routes	Beacons	Route metric	Routes maintained in	Route reconfiguration strategy
AODV	F	No	Yes	Freshest & SP	RT	ER then SN or LRR
DSR	F	Yes	No	SP, or next available in RC	RC	ER then SN
ROAM	F	Yes	No	SP	RT	ER &[a]

LMR	F	Yes	No	SP, or next available	RT	LR & RR
TORA	F	Yes	No	SP, or next available	RT	LR & RR
ABR	F	No	Yes	Route stability & SP & [b]	RT	LBQ
SSA	F	No	Yes	Strong signal strength & stability	RT	ER then SN
RDMAR	F	No	No	Shortest relative distance or SP	RT	ER then SN
LAR	F	Yes	No	SP	RC	ER then SN
ARA	F	Yes	No	SP	RT	Use alternative route or back track until a route is found
FORP	F	No	No	RET & stability	RT	A Flow_HANDOFF used to use alternative route
CBRP	H	No	No	First available route (first fit)	RT at CH	ER then SN & LLR

RS (routing structure); **F** (flat); **H** (hierarchical); **SP** (shortest path); **RT** (routing table); **ER** (erase route); **SN** (source notification); **LRR** (local route repair); **RC** (route cache); **RET** (route expiration time); **LBQ** (localised broadcast query).
[a]Start a diffusing search in case a successor is available; else send a query with infinite metric
[b]Route relaying load and cumulative forwarding delay

It can be observed that almost all protocols in this category are flat, with the exception of CBRP. As reactive protocols use flooding to discover on-demand routes, routing structures which are flat can prolong the propagation of messages, and can thus cause scalability concerns. In order for scalability problems to be eliminated, the route discovery and route maintenance processes should be controlled in some manner. For example, LAR restricts the propagation of RREQ packets in the greater vicinity of the destination node. However, LAR requires that each device to be equipped with GPS, which may not be easy to guarantee. Half of the protocols in this category allow nodes to store multiple routes to a single destination, which may generally be beneficial, as a node can immediately resume transmission in case of a primary route failure. In addition, most routing protocols of this category base their route metrics on the standard *shortest-path* algorithm, which simply chooses a *best* route to a destination, if the destination based on that route is fewer hops away than any other route maintained by the source node. Perhaps, more sophisticated route metrics can be found in ABR, where route decisions are based on route stability, and in SSA, where route stability metrics are further enhanced by signal strength measurements.

Table 2.4, summarises the time and communication complexities for route discovery and maintenance, and also outlines the advantages and disadvantages of each protocol. In relation to this table, it is clear that reactive routing protocols impose a significantly lower overhead

than proactive, especially in cases where GPS is required. This is because nodes are not required to periodically exchange large amounts of routing information, as route discovery is performed on-demand, and when necessary. This is especially advantageous for scalability, which could thus be extended compared to proactive protocols. In an approximate estimation, source routing protocols, such as DSR, could support up to a few hundred nodes, while point-to-point routing protocols, such as AODV, could scale even higher. Although these figures may provide just an indication, accurate estimations can only be deduced by large-scale simulation experiments, or ideally real-world experiments, where factors such as traffic levels, distance (number of hops), and mobility are being varied at multiple levels. In addition to the above, reactive routing protocols impose less storage requirements than proactive routing protocols, depending on the number of routes kept at each node. The main disadvantage of protocols in this category is that they normally introduce higher latency than proactive ones, mainly during route discovery.

Table 2.5 summarises the main features of each of the key hybrid protocols. Protocols in this category are mostly hierarchical, with the exception of ZRP. Hierarchical protocols, such as ZHLS, and SLURP, may perform significantly better than other hierarchical protocols previously described, as they require the use of GPS, and thus have a simplified location management process. Therefore, there is no specific requirement for critical nodes, such as cluster-heads, and this may result in an increased overall reliability of the protocol. Additionally, these protocols require less routing information, such as the node ID and zone ID, compared to other protocols, and could thus cope better in highly-dynamic networks. They also eliminate single-points-of-failure by allowing nodes to cooperate as a group. Storage requirements are hard to determine, as they are highly dependant on the size of each cluster or zone, which act proactively.

Table 2.4 (Abolhasan, M., et. al., 2004): Reactive routing protocols, overhead

Protocol	**TC[RD]**	**TC[RM]**	**CC[RD]**	**CC[RM]**	**Advantage**	**Disadvantage**
AODV	O(2D)	O(2D)	O(2N)	O(2N)	Adaptive to highly dynamic topologies	Scalability problems, large delays, *HELLO* messages
DSR	O(2D)	O(2D)	O(2N)	O(2N)	Multiple routes, promiscuous overhearing	Scalability problems due to source routing and flooding, large delays
ROAM	O(D)	O(A)	O(|E|)	$O(6G_A)$	Elimination of search-to-infinity problem	Large CO in highly dynamic environments
LMR	O(2D)	O(2D)	O(2N)	O(2A)	Multiple routes	Temporary routing loops
TORA	O(2D)	O(2D)	O(2N)	O(2A)	Multiple routes	Temporary routing loops
ABR	O(D+P)	O(B+P)	O(N+R)	O(A+R)	Route stability	Scalability problems
SSA	O(D+P)	O(B+P)	O(N+R)	O(A+R)	Route stability	Scalability problems,

						large delays during route failure and reconstruction
RDMAR	O(2S)	O(2S)	O(2M)	O(2M)	Localised route discovery	Flooding used if there is no prior communication between nodes
LAR	O(2S)	O(2S)	O(2M)	O(2M)	Localised route discovery	Based on source routing, flooding is used if no location information is available
ARA	O(D+P)	O(D+P)	O(N+R)	O(A+R)	Low overhead, small control packet size	Flooding based route discovery process
FORP	O(D+P)	O(D+P)	O(N+R)	O(N+R)	Route failure minimisation technique	Flooding based route discovery process
CBRP	O(2D)	O(2B)	O(2X)	O(2A)	Only cluster-heads exchange routing information	Cluster maintenance, temporary loops

TC (time complexity); **CC** (communication complexity); **RD** (route discovery); **RM** (route maintenance); **CO** (control overhead); **D** (network's diameter); **N** (number of nodes in the network); **A** (number of affected nodes); **B** (diameter of the affected area); **G** (maximum degree of the router); **S** (diameter of the nodes in the localised area); **M** (number of nodes in the localised region); **X** (number of clusters); **R** (number of nodes forming the route-reply path); **P** (diameter of the directed path); **|E|** (number of edges in the network).

Table 2.6 summarises the time complexity and communication complexities for route discovery and maintenance, and also outlines the advantages and disadvantages of each hybrid protocol. Routing protocols of this category generally cause less network overhead than proactive and reactive protocols. This further increases scalability, which may support more than 1000 nodes.

Table 2.5 (Abolhasan, M., et. al., 2004): Hybrid routing protocols, basic characteristics

Protocol	**RS**	**Multiple routes**	**Bc**	**Route metric method**	**Route maintained in**	**Route reconfiguration strategy**
ZRP	F	No	Yes	SP	Intrazone and interzone tables	Route repair at point of failure and SN[a]
ZHLS	H	Yes	No	SP or next available virtual link	Intrazone and interzone tables	Location request[b]
SLURP	H	Yes	No	MFR for interzone forwarding, DSR for intrazone routing	Location cache and a node_list	SN, then location discovery
DST	H	Yes	No	Forwarding using the tree neighbours and the bridges using shuttling	Routing tables	Holding time[c] or shuttling
DDR	H	Yes	Yes	Stable routing	Intrazone and interzone tables	SN, then source initiates a new path discovery

RS (routing structure); **H** (hierarchical); **F** (flat); **SP** (shortest path); **SN** (source notification); **Bc** (beacons).

[a]The source may or may not be notified.

[b]A location request will be sent if the zone ID of a node changes.

[c]Packets are held for a short period of time during which the nodes attempts to route the packet directly to the destination.

Table 2.6 (Abolhasan, M., et. al., 2004): Hybrid routing protocols, overhead

Protocol	TC[RD]	TC[RD]	CC[RD]	CC[RM]	Advantage	Disadvantage
ZRP	Intra: O(I)	O(I)	$O(Z_N)$	$O(Z_N)$	Reduce retransmissions	Overlapping zones
	Inter: O(2D)	O(2D)	O(N+V)	O(N+V)		
ZHLS	Intra: O(I)	O(I)	O(N/M)	O(N/M)[a]	Reduction of SPF, low CO	Static zone map required
	Inter: O(D)	O(D)	O(N+V)	O(N+V)		
SLURP	Intra: $O(2Z_D)$	$O(2Z_D)$	O(2N/M)	O(2N/M)	Location discovery using home regions	Static zone map required
	Inter: O(2D)[b]	O(2D)	O(2Y)	O(2Y)		
DST	Intra: $O(Z_D)$	$O(Z_D)$	$O(Z_N)$	$O(Z_N)$	Reduce transmissions	Root node
	Inter: O(D)	O(D)	O(N)	O(N)		
DDR	Intra: O(I)	O(I)	$O(Z_N)$	$O(Z_N)$	No zone map or zone coordinator	Preferred neighbours may become bottlenecks
	Inter: O(2D)	O(2D)	O(N+V)	O(N+V)		

TC (time complexity); **CC** (communication complexity); **RD** (route discovery); **RM** (route maintenance); **I** (periodic update interval); **N** (number of nodes in the network); **M** (number of zones or clusters in the network); Z_N (number of nodes in a zone, cluster or tree); **Y** (number of nodes to in the path to the home region); **V** (number of nodes on the route-reply path); **SPF** (single point of failure); **CO** (control overhead);

[a]In ZHLS, the intrazone is maintained proactively. Therefore, a fixed number of updates are sent at a fixed interval.

[b]SLURP's worst case scenario: the source node and the home region of the destination are on the opposite edges of the network.

According to each routing protocol's characteristics, and performance overheads, it is safe to conclude:

- Proactive routing protocols, with flat routing structures, impose heavy overheads and are not scalable.
- Proactive routing protocols, with hierarchical routing structures, may reduce overheads by a certain degree, and provide limited scalability.
- Proactive routing protocols often impose high memory overheads, as they require each node to maintain routing information for each node in the network.
- Reactive routing may be considered as a better routing approach in ad-hoc networks.
- Reactive routing protocols, with flat routing structures, impose high control overhead due to flooding, however, considerably less than proactive protocols.
- Reactive routing protocols with hierarchical routing structures impose less control overhead, as control packets are normally routed through critical nodes, which may also become performance bottlenecks.
- Reactive routing protocols can significantly reduce memory overheads, as nodes normally cache routes they actively using.
- Hybrid routing protocols normally impose less network overheads than proactive and reactive protocols, given that most protocols are hierarchical.

- GPS can dramatically improve the performance of routing protocols, however, it comes at a cost, which users or network administrators may not be willing to pay.
- All routing protocols reviewed so far, do not take into account performance characteristics of individual mobile devices, which can dramatically vary between powerful laptops and resources-limited handhelds.

2.3 Comparison of routing protocols via simulations and real-life experiments

In most cases, once an ad-hoc routing protocol has been designed and properly specified, simulation packages such as network simulator 2 (*ns2*) are used to evaluate the protocol's performance, in comparison to other existing protocols. *ns2* is a discrete event simulator that was originally developed by the University of California at Berkeley and the VINT project (Fall, K. and Varadhan, K., 2005). It provides substantial support for modelling and testing network protocols targeted for fixed, as well as wireless, networks. Recent versions of *ns2* provide built-in support for some well-known ad-hoc routing protocols such as DSDV, AODV, DSR, and TORA, and also provide tools for ad-hoc topology design and testing.

A number of simulation studies for ad-hoc routing protocols have been conducted with the help of *ns2* (Broch, J., et. al., 1998, Das, S. R., et. al., 2000, Aron, I. D., and Gupta, S. K. S., 2000, Dyer, T. D. and Boppana, R. V., 2001, Gray, R. S., et. al., 2004). The first study investigated four ad-hoc routing protocols (DSDV, TORA, DSR, and AODV) in a detailed packet-level simulation. The authors of this work extended the *ns2* to accurately model the MAC and physical-layer behaviour of the IEEE 802.11 wireless LAN standard, including a realistic wireless transmission channel model. The simulations carried out were based on ad-hoc network topologies consisting of 50 mobile nodes. The overall goal of this simulation study was to determine the ability of each protocol to react to network topology changes, while continuing to successfully deliver data packets to their destinations. The movement model was based on the *random waypoint* model (Johnson, D. B. and Maltz, D. A., 1996), and a range of node mobility rates and movement speeds were tested. Results showed that DSDV performed quite predictably, delivering virtually all data packets when the mobility rate and movement speed were low, while failing to converge as node mobility increased. TORA, though, managed to deliver over 90 (%) of the packets with medium data traffic flow, however, it was unable to cope when data traffic increased causing a significant amount of data packets to be dropped. On the other hand, DSR performed well at all mobility rates and movement speeds, although, the source routing option in DSR increased the number of routing overhead bytes required by the protocol. AODV performed almost as well

as DSR, without DSR's source routing requirements, however, at high levels of node mobility, it was more expensive than DSR in terms of network overhead.

Das, S. R., et. al., 2000, evaluated the performance of traditional link-state and distance-vector routing protocols including SPF (Cheng, C., et. al., 1989) and EXBF (Shankar, A. U., et. al., 1992b) against DSDV and on-demand routing protocols, such as TORA, DSR, and AODV. A discrete event, packet-level, routing simulator called MaRS (Maryland Routing Simulator) (Alaettinoglu, C., 1994) was used for comparative performance evaluation. The authors of this study augmented MaRS to provide node mobility, even though their study was limited to the network layer, and there was no modelling of link-layer or physical-layer details. Key metrics used for the performance evaluations of theses protocols included: the fraction of packets delivered, the end-to-end delay, and the routing load. Results showed that the proactive routing protocols, including SPF, EXBF, and DSDV, provided excellent performance in terms of end-to-end delay, however, at a high cost of routing load. In contrast, reactive routing protocols were shown to be significantly more efficient in terms of routing load, however, they suffered from suboptimal routes, as well as having worst successful packet delivery value. In addition, TORA was shown to perform worst than the other reactive protocols tested, even though it maintains multiple redundant paths to single destinations. In particular, the overhead of finding and maintaining this redundant information seemed to outweigh the benefits.

A study which aimed to investigate the effect of local error recovery against end-to-end error recovery in reactive protocols was conducted by Aron, I. D., and Gupta, S. K. S., 2000. The DSR protocol, which uses end-to-end error recovery, was compared to a similar reactive routing protocol, WAR (Aron, I. D., and Gupta, S. K. S., 1999), which uses local correction mechanisms to recover from route failures. The goal of this study was to determine which error recovery mechanism is suitable for a certain mobility rate in the mobile ad-hoc network and to quantify its performance in terms of average packet latency and cost of packet delivery as a function of parameters such as route length, size of the network, mobility rate, and packet arrival rate. Results obtained from this study revealed that the performance of DSR degrades extremely fast as the route length increases, and thus DSR is not scalable, while WAR maintains both low latency and resource consumption, regardless of the route length. The authors further suggested that unless some local error recovery technique is employed to deal with failures along the route to destination, the performance of reactive protocols is not scalable with the size of the network, in terms of route length.

Dyer, T. D. and Boppana, R. V., 2001, examined the performance of the TCP protocol over three routing protocols for mobile ad-hoc networks including AODV, DSR, and ADV (Boppana, R. and Konduru, S., 2001). The adaptive distance vector (ADV) routing protocol

combines an on-demand approach with proactive distance vector routing. This study used *ns2* simulator to evaluate the performance of the routing algorithms with the standard TCP Reno protocol, and Reno with fixed RTO, which was proposed by the authors in (Dyer, T. D. and Boppana, R. V., 2001), and has the ability to distinguish between route loss and network congestion, and may therefore be capable of improving the performance of the routing algorithms. The simulation experiments conducted for this study included a varied number of TCP connections, background constant bit rate (CBR), and number of CBR connections. Results acquired, yielded several interesting insights into the performances of the tree algorithms. Specifically, with standard Reno, ADV performs significantly better compared to AODV and DSR, as it provides lower connection times for TCP, higher throughputs, and lower routing overhead. However, when Reno with fixed RTO was used, the performance of AODV and DSR was significantly improved, while the performance of ADV remained at the same levels as with standard TCP Reno. This is a direct result from the fact that Reno with fixed RTO freezes the retransmission timer when dealing with packet losses due to broken routes, instead of doubling it, and thus data packet retransmissions occur more frequently. In this way, the route discovery process of on-demand routing protocols (AODV and DSR) is stimulated often enough so that they gain the ability to re-establish broken routes.

A more recent study conducted by Gray, R. S., et. al., 2004, was based on a real-life outdoor comparison of four different routing protocols including: APRL (Karp, B. and Kung, H. T., 1998), AODV, ODMRP (Lee, S. J., et. al., 2002), and STARA (Gupta, P. and Kumar, P. R., 1997), which were running on 802.11-enabled laptops moving randomly in an athletic field. Most previous comparison studies of wireless ad-hoc routing protocols involved simulator tools, or small-scale indoor trial runs, and, thus, this study may be considered innovative, as it provided insight into the behaviour of ad-hoc routing protocols in the real-life scales. The study use Any Path Routing without Loops (APRL) and System and Traffic-dependent Adaptive Routing Algorithm (STARA), which are both proactive routing protocols, and AODV and On-Demand Multicast Routing Protocol (ODMRP) which are both reactive. ARPL is simple, as it tries to discover a fixed number of routes, not necessarily shortest, while STARA is more complex, as it uses dynamic latency measurements to decide on best routes. On the other hand, AODV and ODMRP are closely related, with the main difference being ODMRP's multicast traffic support and its inclusion of data packets inside route discovery packets. All four algorithms were implemented in a similar way, as user-level applications, through the use of a tunnel device. The outdoor experiments showed that the reactive protocols performed better than the proactive. In addition, ODMRP outperformed AODV, a result that may be attributed to ODMRP's inclusion of the original

data packet in the flooded route-discovery packets. In contrast, the indoor experiments showed that AODV outperformed ODMRP. This may be attributed to the fact that in the indoor experiments every laptop can hear every packet due to the physical proximity of the nodes, a situation in which the 802.11b protocol can significantly reduce collisions through its standard CSMA/CA protocol.

2.4 Agent-based ad-hoc routing methods

Mobile computing and wireless networking are the most frequently proposed application areas for the mobile agent technology (Kotz, D., et. al., 1997). Mobile agents are considered to be particularly useful in: highly dynamic and unreliable environments; where available bandwidth is limited; and where network links impose high latency and are mostly unreliable. This may be attributed to two important characteristics of mobile agents, which are not present in traditional approaches:

- **Task continuation**. An agent can migrate to a host server to continue a processing task while the user is disconnected from the network (Vinaja, R., 2001).
- **Minimal connection use**. An agent can pre-process information at the server, or, at the mobile device, in order to reduce the communication bandwidth (Kotz, D., et. al., 1997).

Mobile agents have the ability to support asynchronous communications and flexible query processing (Hadjiefthymiades, S., et. al., 2002). Thus, the mobile user can assign a task to a mobile agent, and when the agent senses that there is communication availability, it can roam the network and fulfil the task delegated by its user. In this way, a mobile node requires less communication connectivity than it would need following traditional client-server techniques. Another equally important reason, is than mobile agents are well-known for their ability to reduce network traffic, under specific circumstances, for example, by performing appropriate filtering on data (Braun, P., 2003, Migas, N., et. al., 2004a). Furthermore, mobile agents can increase security (Samaras, G. and Panayiotou, C., 2002) by encapsulating user-profile data and private information, and block unauthorised access. Also, by utilising the built-in security mechanisms of the mobile agent's system, the confidentially and integrity of sensitive information can be strengthened by the use of cryptographic techniques. Background information on intelligent software agents is provided in Appendix C.

Recently, there have been numerous proposals of the mobile agent paradigm in ad-hoc routing (Anderegg, L. and Eidenbenz, S., 2003, Marwaha, S., et. al., 2002, Bandyopadhyay,

S. and Paul, K., 1999), topology discovery (RoyChoudhury, R., et. al., 2000), and clustering formation (Denko, M. K., 2003). In a highly-dynamic ad-hoc network, which suffers from frequent mobile host disconnections, the control overhead of unproductive route request packets may be significantly high. Moreover, in such a highly dynamic network, it is likely that even network traffic with low requirements, such as e-mails, would be inefficient, if not impossible, to be successfully routed from the source to the destination.

The authors in (Bandyopadhyay, S. and Paul, K., 1999) proposed a mobile agent scheme to address this issue. The underlying principle of the scheme is that mobile agents could act as messengers that would migrate from a source to a destination. A mobile agent could easily be dispatched from a source node with the communication data in its payload, autonomously navigate through the ad-hoc network, find the destination, and deliver the message. The authors evaluated the effectiveness of their scheme on a simulated environment, in a closed area of 1000 × 1000 units, with network sizes of 20, 40, and 60 mobile hosts, and two different mobility speeds (10units/sec and 30units/sec). The transmission range of each node was varied from 50 to 150 units. Simulation results showed that the average number of hops taken by an agent to deliver a message was not significantly high. Moreover, there was no strong association between the number of hops taken by the agent and the nodal population in the network. This is due to the fact, that a mobile agent can always find the appropriate *next hop*, using an efficient routing protocol infrastructure, and thus an increased network size would provide the agent with more migration options. Even with increased network sizes, the increase in traffic with relation to the increase in the number of nodes was shown to be low. This comes in contrast to standard flooding algorithms.

A comprehensive and novel routing protocol for ad-hoc networks, which utilises selfish agents that accept payments for forwarding data for other agents, was presented in (Anderegg, L. and Eidenbenz, S., 2003). Selfish agents announce their individual costs for forwarding data for other agents, and accept, only, if the payments made truly cover their expenses. The routing protocol, called ad-hoc-VCG, is reactive and achieves truthfulness, as it is designed in such a way that it is in the agent's best interest to reveal its true costs for forwarding data. In addition, it is financially cost-efficient, and thus guarantees that data packets are being routed along the most cost-efficient path. This is achieved by making payments to the intermediate nodes, consisting of a small premium, in addition to their *real* costs for forwarding the data packets. The *real* cost of an intermediate node which forwards data packets in favour of some source node, is defined as the total amount of wasted energy throughout this process.

The protocol is not budget-balanced, in the sense that the intermediate nodes receive premiums over their actual costs. However, the total overpayment is bounded by a factor of:

$$2^{\alpha+1} \cdot \frac{c_{max}}{c_{min}} \tag{2.1}$$

The variable α is the signal loss exponent and c_{max} (c_{min}) is the maximum (minimum) cost-of-energy declared by the nodes on the most cost-effective path. Therefore, the protocol guarantees that the total amount of payments made will be less than this factor, times the cost incurred by routing along the most cost-efficient path. The underlying idea for achieving truthfulness is to make cheating unattractive by making payments as high as a node could possibly expect to obtain by cheating. Ad-hoc-VCG is robust against a single cheating node, however, it may fail in the presence of coalitions of nodes which try to maximise their total payments. Ad-hoc-VCG's route discovery process is similar to DSR's (Johnson, D. B., et. al., 2004), and, it briefly works as follows:

- A source node *S* initiates a session for destination node *D*.
- Ad-hoc-VCG channels all information regarding shortest paths to the destination node *D*.
- The destination node *D* computes the shortest path, and all the VCG payments that need to be made.
- The destination node *D* sends this information back to the source.

The source node then sends its data packets along with electronic payments, to the destination through the shortest route. A disadvantage of ad-hoc-VCG is its requirement for the complete knowledge of the underlying topology, which inevitably creates a large overhead in the route discovery phase.

An additional study, in the context of ad-hoc routing with mobile agents support, was proposed by Marwaha, S., et. al., 2002, who propose a hybrid routing scheme, called Ant-AODV, which combines the on-demand nature of AODV with a proactive distributed topology discovery mechanism using ant-like mobile agents. The primary aim of this protocol is to reduce frequent update disseminations, usually required by proactive protocols, and further reduce route discovery latency and end-to-end delays, usually found in reactive protocols, and thus provides support for real-time data and multimedia communication. To verify their method they used *ns2* simulator to compare the Ant-AODV to conventional ant-based and AODV routing protocols. A network of 50 mobile nodes, moving according to the *random waypoint* model at a speed of 0-10m/s, and 20 constant bit rate (CBR) sources, was simulated. In addition, several combinations of ant population and history sizes were

used in the simulations. In terms of normalised routing overhead, the results matched the logical expected values, that is, the normalised routing overhead of the ant-based routing was, by far, the highest, while the overhead imposed by AODV was the lowest. Normalised routing overhead is referred to the number of routing packets transmitted per data packet received at the destination. The reason for the worst performance of the ant-based routing is that the actual data packets delivered were fewer and thus the ratio of control overhead to data packets delivered became too high. In contrast, the normalised overhead of Ant-AODV was shown to be slightly greater than AODV, and this may be attributed to the continuous movement of ants in the network. However, Ant-AODV achieved the highest connectivity and fewer end-to-end delays, at a cost of extra processing of the ant messages, and a slightly higher overhead in occupying network capacity.

RoyChoudhury, R., et. al., 2000, proposed a multi-agent based framework to address the aspect of topology discovery in wireless ad-hoc networks. The framework utilises mobile agents with purpose to collect topology-related information from each node and distribute their knowledge, in terms of updates, to all other nodes. The notion of *stigmergic communication* has been used throughout the implementation of a shared information cache in each node, while the concepts of *link stability* and *information aging* were used in order to assist a node with predicting the current network topology, based on the current network information stored at the node. Accordingly, each node in the network has a *recency* token, which is a counter that is initialised to zero. When an agent finishes its task on a node, it increments the *recency* token found on the node by one, just before self-migration, and then memorises the token's value and assigns it to the information retrieved by that node. This technique prevents an agent from updating a node n with routing information about node m, which is older than the information maintained at node n concerning node m. Simulation results showed that the average connectivity convergence improves with a decrease in mobility. For time-to-migrate (TTM) equal to 100ms and high mobility equal to 30m/s, the connectivity convergence goes below 80%. However, if the time-to-migrate is further reduced the connectivity convergence should be logically increased. In case that the predictive mechanism is used, even in the context of TtM equal to 100ms, the convergence values were shown to reach over 98%.

The use of mobile agents for clustering has been proposed by Denko, M. K., 2003, which uses a mobile agent-based clustering architecture for routing in mobile ad-hoc networks, aiming to reduce the routing overhead, and thus provide a more scalable solution, compared to non-agent-based approaches. According to the proposed architecture, mobile agents are responsible for cluster maintenance, and for updating routing information at each node. Mobile agents are also responsible to participate in cluster size adjustments, re-clustering, and

continuous cluster status monitoring. In addition, they proposed that intra- and inter-cluster routing can be carried out using different protocols, either reactive, or proactive. Each node maintains a clustering table, which includes information, such as the IDs of its neighbours, the node's role, mobility information, and so on, and also maintains a routing table which contains the routes to known destinations. Connectivity information is gathered by periodic *HELLO* messages as in standard clustering algorithms. In addition, two instances of the following mobile agents are implicitly associated with each cluster-head:

- **Routing mobile agent** (**RMA**). This mobile agent moves across the ad-hoc network and collects routing information, which it then stores and maintains in the routing table of its home node.
- **Clustering mobile agent** (**CMA**). This mobile agent migrates to adjacent clusters and collects clustering information. Once data gathering is completed, the mobile agent returns back to its home node, and submits the collected information to the node's clustering table.

Due to the long migration times of mobile agents, the clustering architecture requires a reasonable time to stabilise. However, once stable, each node has a complete knowledge of its neighbours, while the cluster-head may have additional information about other cluster-heads, gateways, and the routes to reach them. Therefore, the cluster-head has a central role compared to non cluster-head nodes. In particular, it is responsible for updating its members with clustering and routing information. They are currently evaluating the performance of their proposed architecture and its routing behaviour using various performance metrics, including: cluster-head changes, gateway changes, cluster size, and cluster membership changes.

2.5 Chapter Summary

This chapter has shown that proactive routing protocols cope well, with small-scale ad-hoc networks, normally consisting of up to 100 nodes, whereas, reactive routing protocols can cope well with increased network sizes, as they reduce the amount of routing update disseminated through the network, which is typically the case in proactive routing protocols, and thus scale better. However, reactive routing protocols have their limitations, as the network overhead imposed by frequent route-requests, especially in the case of high-mobility networks, can be highly, and thus they are ideal for medium-scale networks, normally consisting of up to a few hundred nodes. An alternative solution that attempts to minimise the scale of routing updates, as well as, the requirement for frequent route-requests, is found in

hybrid routing protocols. These, typically organise the network into single, or, multiple pro-active- and reactive-zones, where in the former nodes have knowledge of the topology, whereas in the latter, nodes learn routing information normally by means of route-requests.

Hierarchical routing protocols impose a hierarchical structure, normally by organising the network into clusters, which aim to restrict the propagation of control messages to key nodes, only, and thus reduce the network overhead. However, key nodes may become network bottlenecks, and single-points-of-failure, unless a sophisticated clustering formation mechanism is used. In addition, most current routing protocols do not provide route redundancy, and they typically rely on the hop-counting mechanism for *optimal* route identification, which significantly narrows the reliability, and performance of the system. In addition, hierarchical protocols, do not pay attention to key metrics, such as the processing capability, network state, and overall utilisation of key nodes, and thus it is possible for a resource-constrained device to become a cluster-head role, whereas, a high-end device to become a member, which is unacceptable, as a cluster-head typically has location management and routing responsibilities, whereas, a member has minimal responsibilities. Thus, the network's backbone is likely to consist of unsuitable devices, which significantly affect the network's overall performance, and, at the same time, introduce unfairness to resource-constrained devices.

3 Model

3.1 Introduction

This chapter presents the model and specification of a routing protocol, named Mobile Agents for Routing In Ad-hoc Networks (MARIAN), which was designed for the purpose of this research. MARIAN is specifically designed for use in multi-hop wireless ad-hoc networks, where participating nodes may range from resource-constrained devices, such as PDAs, to high-end devices, such as laptops and workstations. MARIAN provides the network with self-organisation and self-configuration abilities, and thus reduces the need for any existing network infrastructure or administration. It is based on the well-known, on-demand, CBRP (Jiang, M., et. al., 2001), and extends it in a number of ways. Initially, CBRP is purely on-demand, while MARIAN is a hybrid protocol, which uses stationary, on-demand, intelligent agents and proactive mobile agents for network topology discovery, routing, and network reconfiguration. In addition, CBRP uses single routes for any particular destination, while MARIAN allows each source node and each cluster-head to maintain multiple routes to a destination, and use these effectively. Furthermore, CBRP provides no mechanisms for deciding on optimal routes, while MARIAN is metric-driven oriented, and, thus, MARIAN can provide QoS by deciding on optimal routes in terms of the routing scenario which it is aiming to accomplish, and, at the same time, protect devices' vital resources, such as CPU, memory, and battery life.

3.2 Synopsis of MARIAN specification

The proposed routing protocol is simple and efficient, and is designed specifically for multi-hop ad-hoc networks, with resource-constrained devices, such as PDAs. An ad-hoc network operating under MARIAN can be completely self-organising and self-configuring, and thus requires no existing infrastructure, or external control. Network nodes within direct communication range may exchange data directly, while nodes that are far away from each other may pass network traffic over other intermediate, cooperating, network nodes.

Initially, the protocol organises participating network nodes into a number of intersecting, or, adjacent clusters. The clustering formation algorithm used by MARIAN is a variation of the well-known lowest-ID algorithm presented in (Gerla, M. and Tsai, J. T.-C., 1995). According to the standard lowest-ID algorithm, a node is elected as a cluster-head if

its ID is the lowest amongst its neighbouring nodes. The ID must be a unique number, such as the node's IP address. CBRP, for example, bases its clustering formation on this algorithm, however, it provides no mechanism for deciding upon the most optimal node to become a cluster-head. On the contrary, MARIAN uses a metric-oriented, lowest-ID algorithm, which is essentially similar to the lowest-ID algorithm, however, the unique ID is represented by a sophisticated metric, which integrates the mobility and performance parameters of the device. The mobility parameter is measured without the need of GPS, and similar to MOBIC, which is proposed in (Basu, P., et. al., 2001), while the performance is measured by conducting a number of preliminary tests, such as rigorous algorithm calculations, buffering capabilities, network state, and a number of continuous tests, such as CPU, memory, and battery-level monitoring. Thus, it is possible that two devices can calculate the same node-ID, as a result of very similar hardware characteristics and mobility patterns.

However, the lowest-ID algorithm (Gerla, M. and Tsai, J. T.-C., 1995) in which MARIAN's clustering formation is based, strictly forbids duplicate values. In order to guarantee that each device will calculate a fair and unique node-ID, a function which generates a unique ID from a non-unique value must be used. The most attractive approach is to multiplex the node's MAC address, which is guaranteed to be unique for each device, with the calculated node-ID, in such a way, so as to guarantee the node-ID uniqueness and most importantly leave the actual value virtually unmodified. An alternative approach would be to allow nodes with the same node-ID to re-execute the election process as many times, as necessary, as to eliminate the duplicate values. This is likely to be dissolved rapidly since a node-ID is linked to various frequently changing factors, such as the battery level. However, this approach may result in increased delays, network overhead, and implementation of coordination efforts.

Nodes which are directly linked to only one cluster-head are member nodes of that cluster, which is bound by the cluster-head, while nodes which belong to two, or more, clusters are gateways. A sub-class of a gateway is the distributed gateway pair, which consists of two nodes in sequence, where each node is attached to a different cluster-head, and thus the pair links two adjacent clusters, without them actually intersecting. A member node has minimal responsibilities in the routing process, while the cluster-head is required to maintain routing tables for inter-cluster routing, and actually route data within its own cluster. Thus, the clustering formation algorithm has to be precise, in terms of the selecting the most suitable nodes for taking up the cluster-head role, as normally, week cluster-heads can easily become performance bottlenecks, and single-points-of-failure nodes. Gateways and distributed gateway pairs are responsible for inter-cluster routing, and, thus, the selection of the appropriate

gateway, or, distributed gateway pair, is important, as nodes may significantly vary in their routing ability.

Once the clustering formation process is completed, each cluster-head can proactively dispatch a network topology mobile agent, so as to collect network topology information concerning clusters beyond its own. This process supports two levels of operation: a periodic, and a triggered-event dispatch. According to the periodic dispatch, a cluster-head may dispatch a network discovery agent every t (s) time, which may be dynamically-tailored to the needs of the cluster. According to triggered-event dispatch, the cluster-head bases its decision on various parameters, such as the frequency of route-requests (RREQs) heard from its member nodes during a certain time period. In addition to the proactive mobile agent approach, MARIAN supports a reactive stationary agent route discovery process, which is initiated each time a node requires a route, which does not exist in its route cache, nor is available in its local cluster-head. In this way, the proactive and reactive approaches operate, in parallel, that is, the proactive assists nodes in retrieving routes easily, by simply requesting them from their local cluster-heads, whereas, the reactive approach is used when the required route does not exist in neither the node's route cache, or, in the local cluster-head. Thus a source node earns valuable time by, initially, requesting the route from its cluster-head, which would, otherwise, be wasted in the network discovery process, and thus the scheme reduces overall latency.

Route requests are always forwarded along a repeated sequence of alternating cluster-head and gateway node pair(s), which is also the case for mobile agents. In this way, the propagation of route requests and mobile agents is limited as there is no need for them to visit every single node in the network, because the network is clustered. This can dramatically reduce network overhead, which is typically associated with standard flooding techniques (*see* Section 2.2). The on-demand, network discovery process of MARIAN is similar to that of CBRP, however, MARIAN is able to discover multiple routes for a given destination, along with the routing-metrics associated with each node. In particular, each node is associated with two metrics. The first one, which has previously been described, is the cluster-head metric, which is integrated into the node-ID, while the second one is the routing-metric, which is deduced by the output of standard performance tests executed on each device, in advance, including complex calculation tests, buffering capability tests, network throughput and packet error percentage, and so on. In addition, the routing-metric incorporates a number of varying parameters, such as the CPU utilisation, memory usage, and, if applicable, the battery level. As a result of this, the metric is non-fixed, and may vary considerably as these parameters change. For instance, a device with an initial strong routing-metric, which at some point in the future undertakes significant internal changes, such as when the CPU

utilisation significantly increases, or the battery drops below a certain level, can cause the metric to become considerably weaker.

When a source device receives multiple routes for a destination, along with the routing-metric of each intermediate device, it dynamically calculates the overall routing fitness of each route provided, with respect to the type of traffic which it intends to transmit. For example, for traffic types with high requirements, such as synchronous (real-time audio), the route with the lowest (strongest) overall routing-metric will be selected, while, for traffic types with low requirements, such as asynchronous (text), the highest (weakest) overall routing-metric route will be used instead. In this way, MARIAN allows network traffic to flow efficiently through the network, by appropriately utilising routes, that is, using the routes which can support the type of traffic being sent, and, thus, it provides an improved level of QoS support. In addition, MARIAN possibly avoids the over-utilisation of routes which consist of devices with considerably low battery, or high utilisation. In this manner, it should extend the life-time of low-battery devices that would be otherwise forced to route data, and, furthermore, prohibit the overburdening of already highly-utilised devices. The routing metrics are thoroughly discussed in Section 3.13.

MARIAN is a loop-free protocol as it bases its fundamental functionality on CBRP which is also loop-free. MARIAN allows mobile nodes to maintain their own route caches for routes which have been discovered by the on-demand network discovery process, while routes which have been discovered by the proactive mobile agents are maintained in the cluster-heads' routing table.

Route maintenance allows a source node to detect, while using a source route to a destination node, possible link breakages along that source route, which may be due to changes in the network topology as a result of nodal movements. When an intermediate node along a route senses that the next hop is not available, it creates a route error (RERR) packet and transmits it back to the source node. In particular, when a source node, or an intermediate node, forwards a data packet to the next hop, it is responsible for confirming that the data packet has been successfully received by the next hop. Such a confirmation can be provided by either the MAC protocol in use (e.g. link-layer acknowledgement frame defined by IEEE 802.11 (IEEE Standards, 802.11, 1999)), or by a *passive acknowledgement*. In case that an acknowledgement has not been received over a certain number of retransmissions, the link to the next hop is considered unavailable. Thus, in case the transmitting node is other than the source, it constructs a route error packet in order to inform the source that the next hop along the source route is unavailable. Unlike CBRP, MARIAN does not use route shortening or local repair mechanisms, since the resulting shortened or salvaged route may not correspond to the source's expectations, that is, it may not provide the QoS required by the

source. Thus, the source can immediately use an alternative route from its route cache, which is associated with the same or better QoS than the previous route, or, in case of not such a route existing, it may request a route from its cluster-head, or, finally, initiate a new route discovery.

The actual routing of data packets is performed using source routing, and is similar to DSR (Johnson D. B., et. al., 2004). The main advantage over hop-by-hop routing is that it eliminates the need of intermediate nodes maintaining up-to-date routing information.

3.2.1 MARIAN Assumptions

The underlying assumptions made by this routing protocol are the following:

- **Nodes are cooperating in the routing protocol, without cheating.** It is generally assumed, that each node in the ad-hoc network, which wishes to communicate with other nodes, will accept the role given to it by the protocol, and perform its duties, without cheating. For instance, a node which is given the role of a gateway will not purposely block its data forwarding mechanism in order to avoid resource-consumptions, such as battery discharge, increased utilisation, and so on.
- **Nodes can detect corrupted packets and discard them**. As a result of the unreliable nature of the wireless medium, packets may be often lost or corrupted, at least in comparison to a fixed network. It is assumed that a node receiving a corrupted packet can detect the error and discard the packet. The node can then request the retransmission of the packet.
- **Movement of network nodes is not extreme and continuous**. Nodes within the ad-hoc network, operating under MARIAN, are free to move in an arbitrary fashion, and may even move continuously, but within moderate speeds. In particular, if all nodes are continuously and rapidly moving, the clustering organisation will undertake frequent changes, and thus impose a significantly high network overhead. In addition, routing of data packets may become extremely difficult, and in excessive mobility scenarios, packets may be forced to be routed by means of flooding to every possible destination. Therefore, to guarantee proper operation of the routing protocol and gain the benefits offered by clustering, the speed should be moderate.
- **Node links are always bi-directional**. MARIAN assumes that every link in the ad-hoc network is bidirectional, and that each device supports an underlying MAC protocol, such as IEEE 802.11 (IEEE Standards, 802.11, 1999), which provides link-layer acknowledgements. This way, a node that receives a route-request can reverse the route,

which was taken by the route-request, and thus instantly obtain a route back to the requesting node.

- **Node IP addresses are assigned by an external mechanism.** The node IP addresses are assumed to be assigned by a mechanism external to the routing protocol, such as static assignment or dynamic assignment by the use of DHCP.

In addition, the network diameter (Δ) is the maximum distance, in hops, in which a data packet must transverse in order to reach the destination. In practice, however, the distance will usually be much less than the actual diameter of the network, and possibly significantly fewer hops due to the advantages provided by clustering formation.

3.3 Overall MARIAN model

This section presents an overview of the protocol's overall model, which is then decomposed and presented, in detail, in later sections. Figure 3.1 presents the overall model, based on a small-scale network topology, and depicts: the metric-driven clustering formation; the proactive network discovery mobile agent approach; the reactive route discovery stationary agent approach, and the data structures, which are maintained at each node.

As illustrated, the network is organised into four clusters, that is, cluster *A*, cluster *B*, cluster *C*, and cluster *D*, using the nodes' IDs, which are derived by the cluster-head metrics, as previously mentioned. Clusters *A* and *C* intersect, as nodes G_1 and G_2 are directly linked to the clusters' cluster-heads R_2 and R_3. Similarly, clusters *B* and *D* intersect, using the node G_3, and, clusters *C* and *D* intersect, using node G_4. Although clusters *A* and *B* do not intersect, as nodes DG_1 and DG_2, are not directly linked to both clusters' cluster-heads, the pair of nodes DG_1 and DG_2 can be used for inter-cluster routing, and thus it is called distributed gateway pair. The clustering formation process ensures that each connected node belongs to at least one cluster, and that each cluster is linked to its adjacent clusters, as long as, there is connectivity (*see* Section 3.5.1), thus, the clustering formation process is precise. In addition, as a result of the clustering formation algorithm, cluster-heads can be at either 2-hops, or 3-hops away, for example, R_1 is 2-hops away from R_4, whereas R_1 is 3-hops away from R_2.

Each node runs an agency, which provides the execution environment for stationary and mobile agents, whereas, a cluster-head, in addition to the agency, it runs a region registry, which provides a registration service for agencies, stationary and mobile agents, which exist in its cluster (*see* Appendix A). The agency and region addresses are assigned by the mobile agent system, which are, typically, two *strings*, and they consist of: the communications protocol used; the node's IP address; the port number, which is different for agencies and

regions; and the name of the agency/region. In addition, each node's agency produces an identifier, which is referred as the agency identifier, that is, a unique value in a distributed environment (*see* Appendix A), and is thus used by the reactive route discovery process, which is presented, in detail, in Section 3.6.1. Furthermore, each node has a node address, which is its IP address, normally, the IP of its wireless interface. Each non-cluster-head node registers to its local cluster-head's region registry, and receives a ticket, where, once expired, the node assumes that it is no longer in direct communication range with the cluster-head. As an example (*see* Figure 3.1), member node M_4 has a node-ID of 85, which represents its ability to become a cluster-head, whereas, it has a routing-metric of 78, which represents its routing ability. In addition, its node address is set to the IP address of its wireless interface, whereas the agency address incorporates the additional information that was previously mentioned.

Each node is required to broadcast its Neighbouring Node Table (NNT), which consists of the nodes, that is, the node address and agency/region addresses, a node can hear from, including, itself, along with the respective node-IDs, routing-metrics, and roles (*see* Section 3.4). Then, each node builds a 2-hop Neighbouring Node Table (2-hop NNT), which is deduced from incoming NNTs, and consists of the 2-hop topology, from the viewpoint of the node, which provides location information about cluster-heads that are 2-hops away. For example, DG_1 can deduce from the NNT broadcasted by DG_2, that DG_2 is directly linked to R_2, and, also, that DG_2 is a distributed gateway, which can provide inter-cluster routing services for cluster *A* and *B*. However, R_1 does not know the existence of cluster *A*, yet, as this information cannot be deduced by the NNT broadcasted by DG_1, and, thus, DG_1 is required to broadcast its 2-hop NNT, so that cluster-head R_1 can learn about cluster *A*. Specifically, each distributed gateway is required to broadcast its 2-hop NNT, in addition to, its NNT, and thus cluster-heads can obtain knowledge of each adjacent cluster-head, that is, cluster-heads which are 3-hops away. The information contained in 2-hop NNT is: the cluster-heads' node addresses; the cluster-heads' agency address; the distributed gateways' node addresses; the distributed gateways' agency addresses; and the respective node-IDs and routing-metrics (*see* Section 3.4)

As previously mentioned, MARIAN provides an on-demand route discovery process (*see* Section 3.6.1), which is based on stationary agents, and a proactive network topology gathering process (*see* Section 3.6.2) which is based on mobile agents. Both approaches coexist, and execute, in parallel, and thus mobile agents gather the network topology asynchronously, and provide the routing information to their cluster-heads, whereas a node that requires a route, assuming that the route does not exist in its cluster-head's routing table, it can independently initiate a route-request. As an example of the reactive approach, M_5 requests a route to M_3

from its cluster-head R_3, and the cluster-head responds back and provides the routing information, whereas, M_2 requires a route to M_4, but its cluster-head R_1 does not have this information, and thus, M_2 initiates a full-scale route-request. As shown in Figure 3.1, the route-request traverses only key nodes, such as cluster-heads, gateways, and distributed gateway pairs, and, in addition, it travels through all possible paths, which excludes the paths that involve multiple gateways. For example, the route-request travelled from R_2 to R_3 through G_2 only, as there is no need for the route-requests to traverse all possible paths, as this is left to the corresponding route-replies. Furthermore, the route-request was routed through G_2 and not G_1, as G_2 has a lower metric than G_1, and thus it provides a more reliable path. M_4 creates and transmits a route-reply for each incoming route-request, and specifies the reverse route that each route-request took, which consists of the cluster-head list only, as the gateway and distributed gateway information is not required, as each cluster-head knows its adjacent cluster-heads and the way to reach them. The route-replies gather the complete routing information, which includes all possible routes to M_4, as well as, the routing-metrics of each node along these routes, and, thus, M_4 can determine the most optimal routing path, in relation to the type of traffic it requires to transmit, as described in Section 3.13.

The mobile agents propagate in a similar manner to route-requests, that is, they visit each key node in the network by cloning themselves, until a dead-end is reached, that is, each key node has been previously visited, and thus they return back to their originating mobile agent systems, which they submit the routing information gathered to their parents, that is, each node's NNT, and then kill themselves. The mobile agents gather this information from the cluster-heads they visit, and before migration they filter relevant information, so as to reduce the size. However, the full-scale process is not illustrated in Figure 3.1, due to the lack of space, nevertheless, the process it is fully described in Section 3.6.2.

3.4 Message Formats and Data structures

MARIAN defines three data structures which are required for the clustering formation and route discovery processes: the Neighbouring Node Tables (NNTs); the 2-Hop Neighbouring Node Tables (2-Hop NNTs); and the Neighbouring Cluster Tables (NCTs).

Neighbouring Node Tables (NNTs)

Each node creates, maintains, and broadcasts its NNT, which contains information about the node's neighbours, including: the *NodeAddress*; the *AgencyAddress*; the *RegionAddress*, which is only applicable to cluster-heads; the *Node-ID*; the *Routing-Metric*; and the *Role (see* Appendix A*)*.

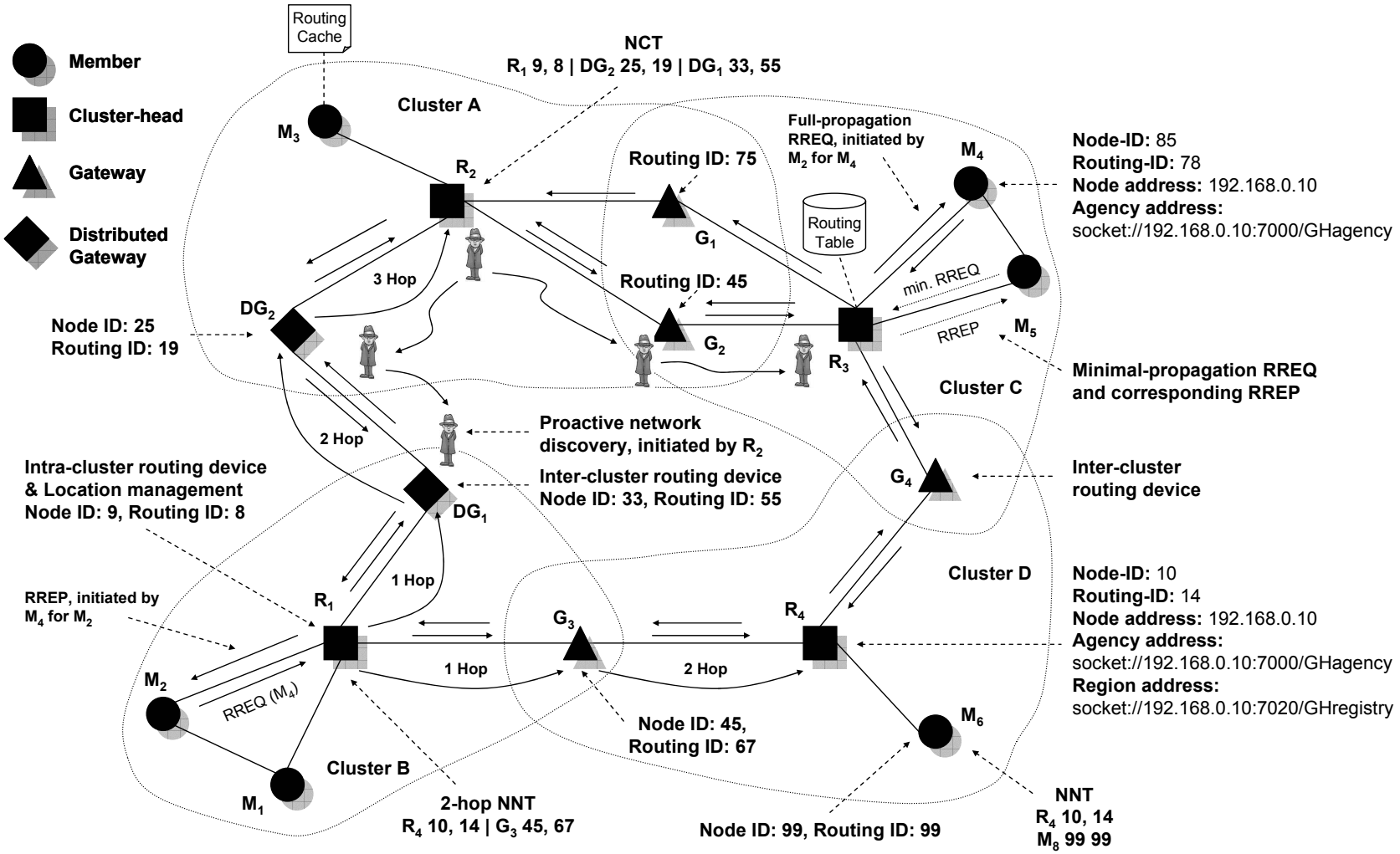

Figure 3.1: The MARIAN overall model

The NNT is broadcasted as an extension to a *HELLO* packet at every *HELLO_interval* time. The NNT is outlined in Table 3.1

Table 3.1: The NNT data structure

NodeAddress	**Agency/Region Address**	**Node-ID**	**Routing-Metric**	**Role**
IP_ address_1	A/R_address_1	cl_numb_1	r_numb_1	Undecided or Member or Cluster-head or Gateway or Distributed Gateway
IP_ address_2	A/R_address_2	cl_numb_2	r_numb_2	Undecided or Member or Cluster-head or Gateway or Distributed Gateway
…	…	…	…	…
IP_ address_n	A/R_address_n	cl_numb_n	r_numb_n	Undecided or Member or Cluster-head or Gateway or Distributed Gateway

The formatting of an NNT extension to a *HELLO* message is shown in Table 3.2. In case the broadcasting node is a non-cluster-head the corresponding *RegionAddress* field of the packet is set to *null*. Similarly, the neighbour *RegionAddress* field is set to *null*, if the neighbour node is a non-cluster-head.

Table 3.2: The NNT extension to a *HELLO* message

				TotalNeighbours			Role
Role	Role	Role	Role	Role	Role	Role	Role
own NodeAddress							
own AgencyAddress							
own RegionAddress							
own Node-ID							
own Routing-Metric							
neighbour NodeAddress [1]							
neighbour AgencyAddress [1]							
neighbour RegionAddress [1]							
neighbour Node-ID [1]							
neighbour Routing-Metric [1]							
…							
neighbour NodeAddress [TotalNeighbours]							
neighbour AgencyAddress [TotalNeighbours]							
neighbour RegionAddress [TotalNeighbours]							
neighbour Node-ID [TotalNeighbours]							
neighbour Routing-metric [TotalNeighbours]							

TotalNeighbours: The total number of listed neighbours
Role: The current role of each entry. (0 Undecided, 1 Member, 2 Cluster-head, 3 Gateway, 4 Distributed Gateway).

2 Hop Neighbouring Node Tables (2-Hop NNTs)

These tables are deduced from incoming NNTs, and contain information about neighbouring cluster-heads, and the corresponding gateway nodes, or, the distributed gateway pairs. As MARIAN utilises multiple routes, information contained in these tables includes all possible gateway nodes, or, distributed gateway pairs, leading to a single cluster-head. 2-hop NNTs are broadcasted by distributed gateways only as an extension to a *HELLO* message. The aim is to allow cluster-heads to identify adjacent cluster-heads that are 3-hops away, typically linked with one, or more, distributed gateway pair(s). The situation where, a cluster-head will be 3-hops away from another cluster-head, is not common, as the clustering formation algorithm, normally, organises the network into intersecting clusters, however, it is likely that in a large network topology there will be a number of non-intersecting clusters. Table 3.3 presents the formatting of the 2-hop NNT, which is maintained at distributed gateway nodes.

Table 3.3: The 2-hop NNT data structure

AdjacentClusterhead [*1*]	DistributedGateway [*1*]	Routing-Metric [*1*]
	DistributedGateway [*2*]	Routing-Metric [*2*]
	…	
	DistributedGateway [*k*]	Routing-Metric [*k*]
AdjacentClusterhead [*2*]	DistributedGateway [*1*]	Routing-Metric [*1*]
	DistributedGateway [*2*]	Routing-Metric [*2*]
	…	
	DistributedGateway [*k*]	Routing-Metric [*k*]
…	…	…
AdjacentClusterhead [*n*]	DistributedGateway [*1*]	Routing-Metric [*1*]
	DistributedGateway [*2*]	Routing-Metric [*2*]
	…	
	DistributedGateway [*k*]	Routing-Metric [*k*]

The information maintained at the 2-hop NNT includes: the *NodeAddress*; *AgencyAddress*; and *RegionAddress* of each adjacent cluster-head, as well as, the *NodeAddress*; *AgencyAddress*; and *Routing-Metric* of each distributed-gateway node leading to these cluster-heads. This data structure is maintained by cluster-heads and distributed gateways, however, only distributed gateways broadcast it as an extension to a *HELLO* message. The formatting of a 2-hop NNT extension is shown in Table 3.4.

Table 3.4: The 2-hop NNT extension to a *HELLO* message

CH_Length	DG_Length (k)	…	DG_Length (m)
AdjacentClusterhead NodeAddress [1]			

AdjacentClusterhead AgencyAddress [1]
AdjacentClusterhead RegionAddress [1]
DistributedGateway NodeAddress [1]
DistributedGateway AgencyAddress [1]
DistributedGateway Routing-Metric [1]
…
DistributedGateway NodeAddress [DG_Length (k)]
DistributedGateway AgencyAddress [DG_Length (k)]
DistributedGateway Routing-Metric [DG_Length (k)]
…
AdjacentClusterhead NodeAddress [CH_Length]
AdjacentClusterhead AgencyAddress [CH_Length]
AdjacentClusterhead RegionAddress [CH_Length]
DistributedGateway NodeAddress [1]
DistributedGateway AgencyAddress [1]
DistributedGateway Routing-Metric [1]
…
DistributedGateway NodeAddress [DG_Length (m)]
DistributedGateway AgencyAddress [DG_Length (m)]
DistributedGateway Routing-Metric [DG_Length (m)]

CH_Length: The total number of listed AdjacentClusterheads.
DG_Length (k): The number of DistributedGateways leading to the first listed AdjacentClusterhead [1].
DG_Length (m): The number of DistributedGateways leading to the last listed AdjacentClusterhead [CH_ Length].

Neighbouring Cluster Tables (NCTs)

These tables are maintained by cluster-head nodes only, and contain information on how a cluster-head can reach an *AdjacentClusterhead* which is 3-hops away. Cluster-heads deduce the NCTs from incoming *HELLO* packets, with the 2-hop NNT extensions, which are broadcasted by *DistributedGateways*. Information, contained in NCTs, includes:

- The *NodeAddress*, *AgencyAddress*, and *RegionAddress* of each *AdjacentClusterhead*.
- The *NodeAddress*, *AgencyAddress*, and *Routing-Metric* of each *DistributedGateway*.

NCTs in conjunction with 2-hop NNTs provide the necessary information to cluster-heads in order for them to know the complete list of their intersecting and adjacent clusters, that is, the complete information about every neighbour cluster-head and the intermediate gateway nodes. In contrast to 2-hop NNTs, NCTs are not broadcasted. Table 3.5 presents the NCT data structure, while the *DistributedGateway* is represented by *D.G.* for better presentation of the table. The first column of *DistributedGateways* refers to the intermediate node which is linked to the clusterhead, while the second column refers to the intermediate node which is

linked to the destination cluster-head. Thus, the pair of *DistributedGateways* allow two cluster-heads, which are 3-hops away, to communicate.

Table 3.5: The NCT data structure

AdjacentClusterhead (1)	D. G. [1]	Routing-Metric [1]	D. G. [1]	Routing-Metric [1]
	D. G. [2]	Routing-Metric [2]	D. G. [2]	Routing-Metric [2]
	…	…	…	…
	D. G. [k]	Routing-Metric [k]	D. G. [m]	Routing-Metric [m]
AdjacentClusterhead (2)	D. G. [1]	Routing-Metric [1]	D. G. [1]	Routing-Metric [1]
	D. G. [2]	Routing-Metric [2]	D. G. [2]	Routing-Metric [2]
	…	…	…	…
	D. G. [k]	Routing-Metric [k]	D. G. [m]	Routing-Metric [m]
…	…	…	…	…
AdjacentClusterhead (n)	D. G. [1]	Routing-Metric [1]	D. G. [1]	Routing-Metric [1]
	D. G. [2]	Routing-Metric [2]	D. G. [2]	Routing-Metric [2]
	…	…	…	…
	D. G. [k]	Routing-Metric [k]	D. G. [m]	Routing-Metric [m]

3.5 Neighbouring cluster discovery process

The clustering formation and clustering maintenance processes require minimal information to be constantly broadcasted as an extension to a *HELLO* message by each participating node, regardless of its role. However, the amount and nature of information broadcasted is related to the node's role. Specifically, nodes with *Undefined*, *Member*, *Gateway*, or *Clusterhead* status are required to broadcast their NNTs as an extension to a *HELLO* message, while nodes with *DistributedGateway* status are required to broadcast their 2-hop NNTs, as well as their NNTs, as an extension to a *HELLO* message. The neighbouring cluster discovery process uses this information to inform cluster-heads of their intersecting, as well as adjacent cluster-heads. In this way, a cluster-head learns information about its intersecting clusters which are 2-hops away by examining incoming NNTs, while it learns information about adjacent clusters which are 3-hops away by examining incoming 2-hop NNTs. In addition, a cluster-head also learns information about every possible gateway, and distributed gateway pair, leading to each cluster. Also, a cluster-head learns the routing-metrics, of each intermediate gateway and distributed gateway pair, and thus a cluster-head can determine, at all times, the *best* routing path, which may be used to reach its neighbouring clusters.

A cluster-head stores information about all of its 2-hop neighbouring cluster-heads in a 2-hop NNT, and all of its 3-hop ones in a NCT, which are maintained locally and are not broadcasted. As an example, consider Figure 3.1, where R_1 learns about R_4 by examining the NNT broadcasted by G_3, whereas R_2 learns about R_1 by the examining 2-hop NNT broad-

casted by DG_2. Specifically, in the first case, R_1 learns that R_4 is two hops away, and that the only intermediate node is G_3 with routing-metric of 67, whereas, in the second case, R_2 learns that R_1 is three hops, and that the intermediate distributed gateway pair is DG_2-DG_1, with routing-metrics of 19 and 65, respectively.

3.5.1 Cluster formation process

In respect to ad-hoc networks, clustering formation refers to the process which imposes a logical structure or hierarchy to an otherwise disoriented network. MARIAN bases its clustering process on a variation of the well-known lowest-ID algorithm (Gerla, M. and Tsai, J. T.-C., 1995). The mechanics of this algorithm along with its properties were thoroughly discussed in Appendix B.7, thus only the modifications performed are presented in this section. Initially, MARIAN adopts the LCC modification proposed in (Chiang, C.-C., et. al., 1997) and tailors it to its needs. The purpose is to extend the life-time of a cluster, and thus reduce network overhead often involved with re-clustering, by defining a set of rules, which are outlined bellow:

- A non-cluster-head never challenges the role of an existing cluster-head, even if its node-ID is lower.
- Only when two cluster-heads move next to each other, one of them loses the cluster-head role, a decision based on the lowest node-ID.
- An exception to the above rules is when a cluster-head produces a node-ID of infinitive (∞) metric; it then loses its role, moves to *Undecided* state, and thus re-clustering is performed.
- When a non-cluster-head node moves out of its cluster, and does not enter into any existing cluster, it forms a new cluster and becomes the cluster-head of this cluster, irrespectively to its node-ID.
- Member nodes which leave their cluster(s) will have to move to an *Undecided* state and re-execute the clustering algorithm.

In addition to the *Clusterhead*, *Member*, and *Gateway* states, which have been defined in (Gerla, M. and Tsai, J. T.-C., 1995), MARIAN defines an *Undecided* state for smoother operation of the clustering formation process, similarly to CBRP. Moreover, MARIAN defines the *DistributedGateway* role, which is not part of CBRP, which aims to simplify the clustering formation process, and, importantly reduce network overhead. This is achieved by requiring only *DistributedGateways* to broadcast their 2-hop NNTs, in contrast to CBRP

where each node is required to broadcast this information. Although MARIAN broadcasts additional information to CBRP, such as the nodal *AgencyAddresses* and *Routing-Metrics*, these reductions in network overhead may compensate for this.

Most importantly, MARIAN uses a metric-driven, lowest-ID algorithm, which employs the scheme presented in (Basu, P., et. al., 2001) and discussed in Appendix B.8. Briefly, this novel approach utilises a mobility metric, named MOBIC, which is calculated based on the ratio of power levels due to successive receptions at each node. MARIAN uses MOBIC for clustering formation in conjunction with performance metrics, such as buffering capabilities, calculation of complex algorithms, throughput, network error percentage, utilisation status, battery level, and so on, in order to calculate an overall node-ID, which represents the fitness of a node to become a cluster-head, and ranges between 0 (best) and 100 (worst).

Five transition state diagrams have been included in this section, which describe the basic functions performed by nodes according to their role, including: *Undecided*; *Clusterhead*; *Member*; *Gateway*; and *DistributedGateway*. Figure 3.2 illustrates the actions performed by an *Undecided* node. Initially, the node sets an *un_timer* and constantly broadcasts a *HELLO* message which includes its NNT. Each time a node receives a *HELLO* message from a node other than itself, it updates its own NNT. If the incoming *HELLO* message is received by a cluster-head, the node cancels the *un_timer* and updates its role to member of that cluster-head. If the *un_timer* expires, the node examines its NNT for entries. If at least one entry exists, the node compares its node-ID with the node-ID(s) of its neighbour(s). If the node has the lowest node-ID it changes its state to *Clusterhead* and broadcasts a triggered *HELLO* message. The node compares its own node-ID with only other *Undecided* nodes. If the node finds no entries in its NNT it sets a new *un_timer*.

Figure 3.3 illustrates the actions performed by a *Member* node towards clustering formation. Specifically, a *Member* node constantly broadcasts its own NNT as an extension to a *HELLO* message. In the case that a *Member* node loses the registration ticket of its cluster-head, it changes its role to *Undecided*. If the *Member* node receives a *HELLO* message from another cluster-head, it changes its role to *Gateway*. If it receives a *HELLO* message from a *Gateway*, it simply updates its own NNT by adding the *Gateway*'s information. In the case where a *Member* node receives a *HELLO* from another *Member*, it examines the other node's NNT and identifies whether the node is a *Member* of a cluster-head other that its own. If this is true it becomes a *DistributedGateway*, otherwise it updates its own NNT inserting the information of the new node. The same goes for a member node which received a *HELLO* message from a *DistributedGateway*.

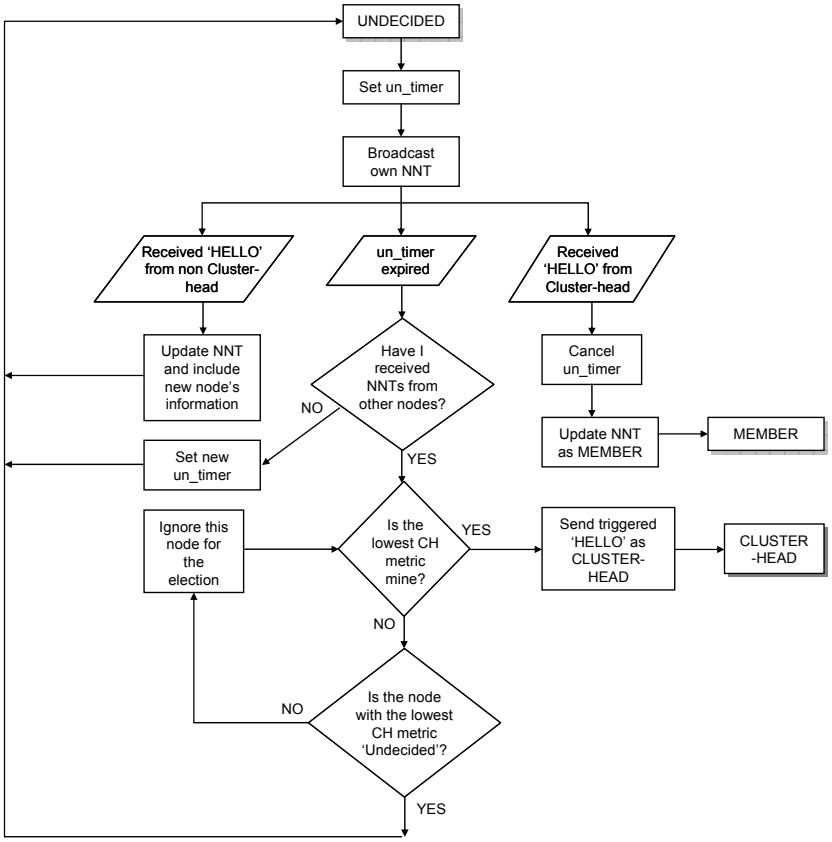

Figure 3.2: Undecided state transition diagram

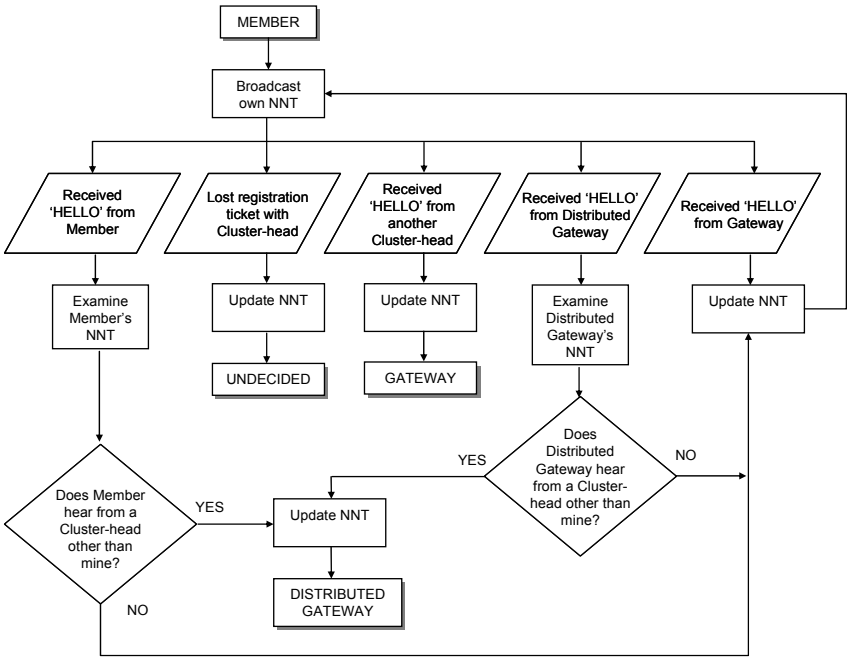

Figure 3.3: Member state transition diagram

Figure 3.4 illustrates the actions performed by a node with a *Clusterhead* role. Once a node has been elected as a *Clusterhead* it starts a region registration service and registers its own agency (*see* Appendix A). A *Clusterhead*, just like a *Member*, constantly broadcasts its own NNT as an extension to a HELLO message. If a *Clusterhead* receives a HELLO from a non-*Clusterhead* node, it updates its NNT with the new node's information and registers the new node's agency to its region registration service. The reason for doing so is that nodes within the same cluster that are two hops away from each other can query the region registry, which runs on the *Clusterhead*, in order to retrieve routing information for each other. Another reason is that network discovery mobile agents can gather information about the member nodes within the cluster, without having to visit every single one, by just querying the cluster-head's region registration. If a *Clusterhead* stops receiving HELLO messages from a registered node, it deregisters its agency from its region registration service. In the case that a *Clusterhead* receives a HELLO message from another *Clusterhead* it sets a *cl_timer*. When the *cl_timer* expires, the *Clusterhead* examines its NNT to verify whether the other cluster-head is still in direct communication range, or that it has left its cluster. If the two cluster-heads are still linked, they compare their node-IDs. The lowest node-ID wins, while the highest loses and thus shuts down its region registration service, changes its role to member and sends a triggered HELLO message as *Member*.

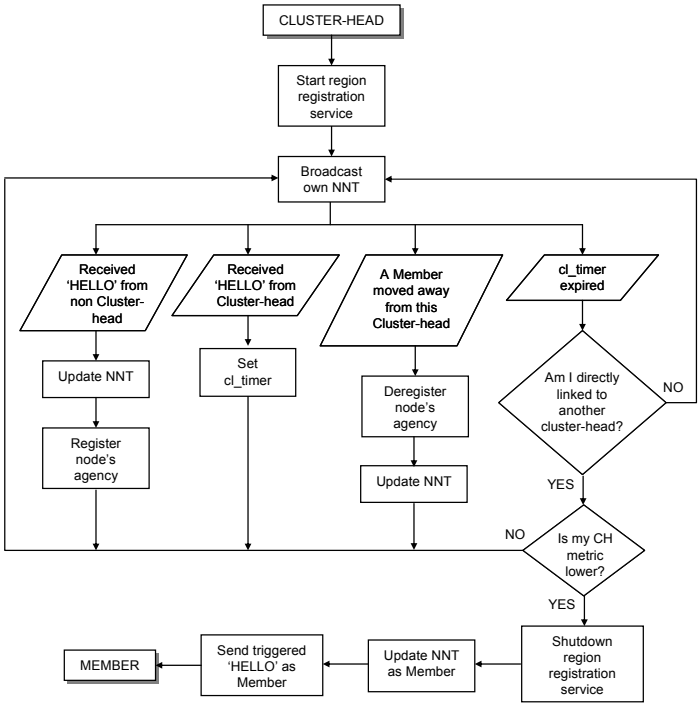

Figure 3.4: Cluster-head state transition diagram

Figure 3.5 illustrates the actions performed by a *Gateway* node. Initially, it examines a *boolean* variable, named *D.G_G*, which reveals if this *Gateway* has *DistributedGateway* responsibilities. If this is true, the *Gateway* broadcasts its own NNT and 2-hop NNT as an extension to a HELLO message, otherwise it simply broadcasts its own NNT. If it receives a HELLO message from a new *Clusterhead* it increases the total number of tickets by one and updates its own NNT with the new cluster-head's information. If the *Gateway* node receives a HELLO from a non-*Clusterhead*, it updates its own NNT with the new node's information. If it loses a registration ticket which was associated with one of its *Clusterheads*, it decreases the number of total tickets by one, and examines the remaining number of tickets. If the remaining number is greater or equal to two, the node remains a *Gateway*. If the remaining number is one, the node becomes a *Member*, in the case of the *D.G_G* variable being false, otherwise, it becomes a *DistributedGateway*. However, if the remaining tickets are less than one, the node becomes *Undecided*.

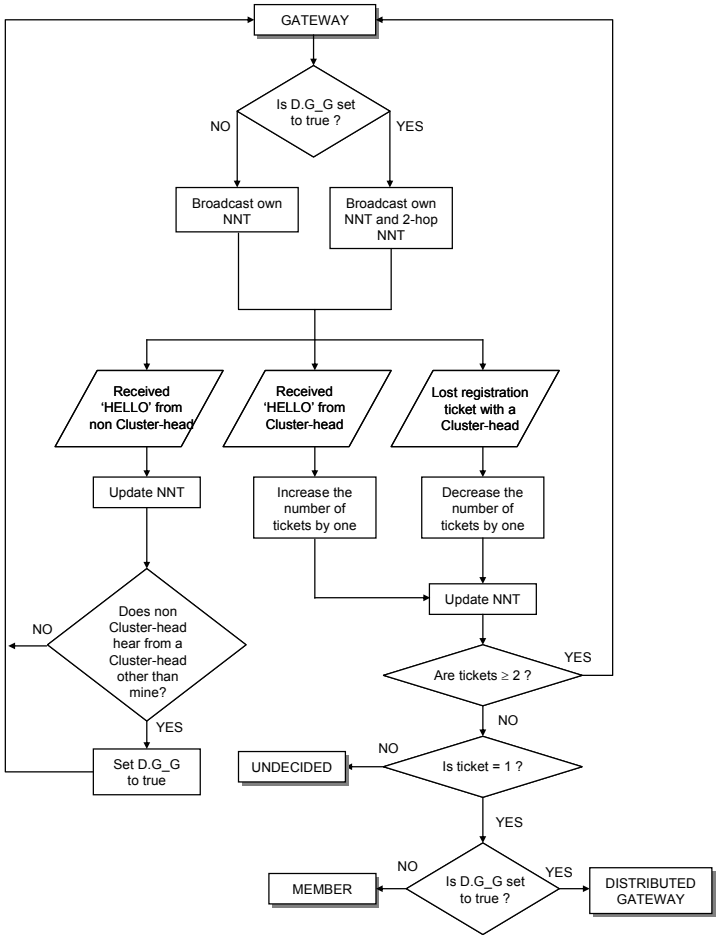

Figure 3.5: Gateway state transition diagram

Figure 3.6 illustrates the actions performed by a *DistributedGateway* node. By default, it constantly broadcasts its own NNT and 2-hop NNT as an extension to a HELLO message. Initially, it sets a *dg_timer* which is usually set to a small amount of time. If the timer expires the *DistributedGateway* checks its own NNT in order to identify whether the *DistributedGateway*, which it is linked to, is still within direct communication range. If this is true, the *DistributedGateway* does not change its state, otherwise, it becomes a *Member* node. If a *DistributedGateway* loses its registration ticket to a cluster-head's region, it changes its state to *Undecided*. If it receives a HELLO from a new *Clusterhead*, it changes its role to *Gateway*, however, it still maintains its distributed *Gateway* capabilities, and thus keeps-on broadcasting its NNT and 2-hop NNT as an extension to a *HELLO* message. If it receives a *HELLO* from a non-*Clusterhead*, it updates its NNT with the new node's information.

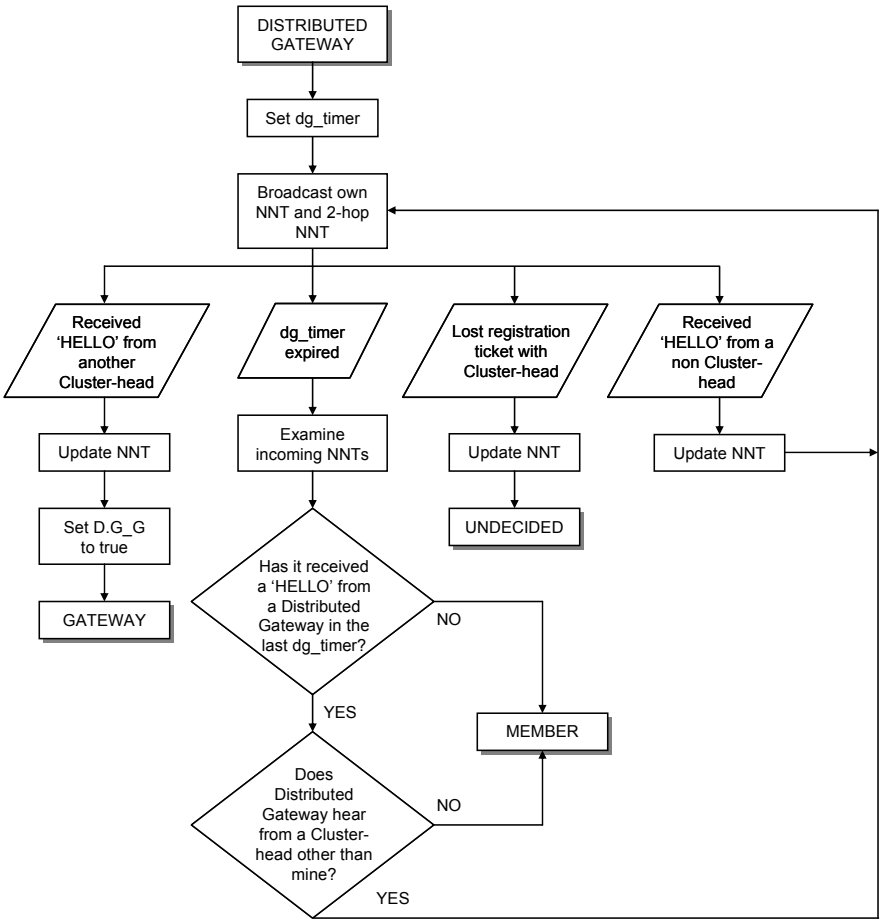

Figure 3.6: Distributed gateway state transition diagram

3.6 Route discovery in MARIAN

MARIAN performs route discovery in a reactive, as well as a proactive manner. The reactive nature of MARIAN is similar to CBRP, however, MARIAN allows multiple routes to be discovered for a single route discovery. In addition, the routing-metrics of each device along each discovered route are returned to the source node. The source can then calculate an overall routing-metric for each route, tailored to its routing scenario. This process may allow intelligent routing decisions to be taken by the source, as opposed to standard hop-counting routing mechanism. For instance, assuming that node *S*, discovers two routes to node *D*, with the first consisting of high routing-metrics, while the second with low. MARIAN enables the source to transmit its high requirements network traffic, such as real-time audio, through the route with the lowest overall routing-metric, while transmit the low requirements network traffic, such as asynchronous chat, through the route with the highest overall routing-metric. In this way, single routes are not over-utilised which can frequently result in over-consuming devices' important assets, and, furthermore, it may provide network scalability, and extend the network's life-time.

MARIAN's proactive nature is based on the mobile agent concept. Specifically, compact, intelligent, goal-oriented agents are migrated from one key node (nodes other than *Member* and *Undecided*) to another, cloning themselves, cooperating with each other, and finally building the network's topology. Cluster-heads are the only nodes which are allowed to initiate these agents, and are responsible for their propagation and population. In particular, agent propagation can be controlled by allowing cluster-heads to dynamically set a propagation limit, which reflects the network discovery-depth the cluster-head wishes to retrieve. The maximum propagation limit is the complete ad-hoc network, while the minimum is the second row of adjacent cluster-heads, for which a cluster-head has no default knowledge. As previously mentioned, MARIAN defines two levels of agent initiation, including a periodic and a triggered-event. According to the periodic dispatch, a cluster-head may dispatch a network discovery agent every t (s) time, which may be dynamically tailored to the needs of the cluster. According to the triggered event dispatch, the cluster-head bases this decision on various parameters, such as the frequency of route-requests heard from its member nodes during a certain time period. The time limits between successive dispatches must be sufficiently large in order to guarantee that the population of these agents is not exceeding a certain threshold, however consistent with the needs of each cluster. Similar to reactive route discovery, these agents gather the routing-metric of each routing node, in addition to the complete network topology. This approach aims to significantly reduce latency often involved with purely reactive methods, and thus maximise support for multimedia

transmissions. This is achieved by providing a complete routing table at each cluster-head, and thus allowing non-cluster-head nodes to retrieve routes to desired destinations, in a much faster way than on-demand route discovery. However, if a desired route is not available in the cluster-head's routing table, or the route does not provide the QoS required by a source node, it may initiate an independent, on-demand route discovery based on the reactive approach.

The clustering structure is employed by the both reactive and proactive approaches, such that the flooding traffic is significantly minimised for the reactive approach, and the mobile agent migrations are significantly reduced for the proactive. Moreover, the combination of these approaches, along with their features can reduce latency, maximise network performance, by employing multiple routes, and provide improved QoS by utilising the routing-metrics.

3.6.1 Reactive route discovery

Each time a non-cluster-head node *S* wishes to send network traffic to destination *D*, it searches its local NNT and identifies if *D* is in direct communication range. If this is true, *S* transmits directly to *D*, otherwise it searches its route cache. If a route is found, such that the QoS required by node *S* is guaranteed, it transmits its data through this route, in a way similar to DSR. Otherwise, *S* constructs a minimal propagation route-request packet (*see* Table 3.6) targeted for its cluster-head, asking for a route to *D*. Node *S* enters its *AgencyAddress*, and, if the route's *NodeAddresses* are required, it enters the target's *NodeAddress*, otherwise, it leaves it blank, and fills in the target's *AgencyAddress* instead, which means that the route's *AgencyAddresses* are required.

Table 3.6: Minimal propagation route request packet format

0 0 1	Identification
Originator's AgencyAddress	
Target's NodeAddress	
Target's AgencyAddress	

Identification: A unique string identifying this minimal propagation RREQ. Perhaps, incorporating the source's agency identifier with the number of this request attempt.

001: Type of MARIAN packet, minimal propagation RREQ.

Target: The agency and node address of the destination node for which the route is requested.

Originator: The agency address of the source node.

Algorithm 3.1 outlines the process followed by a source node, before it starts its data transmission.

Algorithm 3.1: Preparation for data packet transmission by a MEMBER node

```
IF S is a MEMBER
    Examine the NNT
    IF D exists in NNT
        Transmit to D directly
        Terminate
    ELSE IF S's route cache has a route to D which satisfies S's QoS requirements
        Transmit over that route using source routing
        Terminate
    ELSE
        Construct a minimal propagation RREQ
        Transmit RREQ to cluster-head
    ENDIF
    IF the corresponding RREP(s) includes a route such as node S's QoS requirements can be satisfied
        Transmit over that route using source routing
        Terminate
    ENDIF
ENDIF
```

If the cluster-head has a single route to *D*, it replies back with a single minimal propagation route reply (RREP) packet targeted for *S*, and informs it about the complete, ordered list of nodes, and their corresponding routing-metrics. In the case that the cluster-head has two, or, more routes to *D*, it creates and transmits multiple minimal propagation route-reply packets to the requesting node, whereas, if the cluster-head does not have a route to *D*, it creates and transmits a route-not-available (RNA) packet. The minimal propagation route-reply packet format is presented in Table 3.7, while the route-not-available packet format is presented in Table 3.8.

Table 3.7: Route-not-available packet format

1 1 0	RNA	Identification
Originator's AgencyAddress		
Target's NodeAddress		
Target's AgencyAddress		

110: Type of MARIAN packet, route-not-available (RNA).
Identification: The unique identification copied from the minimal propagation RREQ.
Originator: The cluster-head's agency address.
RNA: Always set to 1, and means that the requested route does not exist in the cluster-heads routing table.

Table 3.8: Minimal propagation route reply packet format

1 1 0	TotalSourceAddresses	Identification
Originator's AgencyAddress		
SourceRoute NodeAddress [1]		
SourceRoute AgencyAddress [1]		
Routing-Metric [1]		
…		
SourceRoute NodeAddress [TotalSourceAddresses]		
SourceRoute AgencyAddress [TotalSourceAddresses]		
Routing-Metric [TotalSourceAddresses]		

011: Type of MARIAN packet, minimal propagation RREP.
TotalSourceAddresses: The total number of node or/and agency addresses composing the source route.
Identification: The unique identification copied from the minimal propagation RREQ.
Source Route: The node and agency addresses of an intermediate device along the source route.
Originator: The cluster-head's agency address.

In case that the procedure depicted in Algorithm 3.1 does not supply *S* with the desired route to destination *D*, then *S* may construct and transmit a full-scale route-request, as outlined in Table 3.9.

Table 3.9: The route request packet format

0 1 0	TotalClusterheadGatewayPairs	TotalClusterheadsTraversed	Pointer	Identification
Originator's AgencyAddress				
Target's NodeAddress				
Target's AgencyAddress				
Gateway's NodeAddress [1]				
Neighbouring Clusterhead's NodeAddress [1]				
…				
Gateway's NodeAddress [TotalClusterheadGatewayPairs]				
Neighbouring Clusterhead's NodeAddress [TotalClusterheadGatewayPairs]				
Clusterhead NodeAddress [1]				
Clusterhead NodeAddress [2]				
…				
Clusterhead NodeAddress [TotalClusterheadsTraversed]				

010: Type of MARIAN packet, RREQ.
TotalClusterheadGatewayPairs: This number is the total number of neighbouring cluster-head-gateway pairs this RREQ packet has traversed.
TotalClusterheadsTraversed: This number is the total number of cluster-heads this RREQ packet has traversed.

Pointer:	The position of the neighbouring cluster-head pointer. It is used by the cluster-head in order to identify which is the next neighbour to send this RREQ packet.
Identification:	A number that uniquely identifies this RREQ. It can either be the originator's node address or agency address, along with the number which identifies this cloned packet.
Target:	The node and agency addresses of the destination node.
Originator:	The agency address of the source node. The node address is not required, as the destination can obtain it from the IP source address field in the IP header.
Gateway:	The node address of the intermediate node that will forward this RREQ to the next cluster-head in the list.
Neighbouring Clusterhead:	The address of the cluster-head which this RREQ packet will be or has been forwarded by the corresponding gateway.
Cluster-head:	The node address of the last visited cluster-head.

As a general rule, if node *S* is a non-cluster-head, it sends the packet to each of its host cluster-head(s), that is, the cluster-heads the node has a direct communication link, otherwise, in the case that node *S* is a cluster-head, to its adjacent cluster-heads. As MARIAN imposes a clustering structure, RREQ packets traverse only through key nodes, such as cluster-heads, gateways, and distributed gateways, and, thus, ordinary nodes, are not disturbed.

The information entered in the RREQ depends on the node's role that originates the RREQ, however, each node is required to initially perform the following:

- Enters the unique identification *string* which uniquely identifies this RREQ packet.
- Fills in its own *AgencyAddress*, and the target's *NodeAddress* and *AgencyAddress*.

Then, if node *S* is a *Member* or *DistributedGateway*, it performs the following steps:

- Fills in the packet's *Neighbouring Clusterhead NodeAddress* and *Gateway NodeAddress* fields, with the *NodeAddress* of its cluster-head, in both cases.
- Sets the *TotalClusterheadGatewayPairs* variable to one, and also sets the *TotalClusterheadsTraversed* and *Pointer* variables to zero (*see* Table 3.9).
- Transmits the RREQ packet to its host cluster-head.

If node *S* is a *Gateway* it performs the following steps:

- Fills in the packet's *Neighbouring Clusterhead NodeAddress* and *Gateway NodeAddress* pair(s), with the *NodeAddress* of its cluster-head(s), in both cases.

- Sets *TotalClusterheadsTraversed* and *Pointer* to zero, and sets *TotalClusterheadGateway-Pairs* to the total number of its host cluster-head(s).
- Transmits the RREQ packet to each of its host cluster-heads.

If node *S* is a *Clusterhead* it performs the following steps:

- Fills in the packet's *Neighbouring Clusterhead NodeAddress* field(s), with the *NodeAddress* of its intersecting, or/and, adjacent cluster-head(s),
- Fills in the packet's *Gateway NodeAddress* field(s), with the *NodeAddress* of the corresponding gateways node(s) leading to it's intersecting, or/and, adjacent cluster-head(s). In case of multiple gateways leading to a single cluster-head, the lowest routing-metric gateway is chosen.
- Fills in the packet's *Clusterhead NodeAddress* with its own *NodeAddress*.
- Transmits the RREQ packet to the chosen *Gateway NodeAddress* field(s).

Algorithm 3.2 presents the actions performed by a cluster-head *N* when it receives a RREQ packet. Initially, the cluster-head checks the RREQ's identifier and compares it to the RREQs' identifiers maintained in the cluster-head, and, if a match is found, it discards the packet, otherwise, it proceeds. It then determines if the destination node *D* is in direct communication range, and, if so, it unicasts the packet to *D*, otherwise it proceeds. First, it compares the intersecting and adjacent cluster-head *NodeAddresses*, maintained in its 2-hop NNT and NCT, with the *Neighbouring Clusterhead NodeAddress* field(s) and the *Clusterhead NodeAddress* field(s) stored in the RREQ. The matching node addresses are stored in a temporary list of excluded cluster-heads, which is used as reference for this particular RREQ. Next a cloned packet is created for each of its non-excluded intersecting and adjacent cluster-heads, and is transmitted to the corresponding *Gateway NodeAddress* field(s). It is worth mentioning that each cloned packet has a distinct identification, and is thus treated independently.

Algorithm 3.2: Cluster-head route request packet handling

IF N has already seen RREQ
    Discard RREQ
ELSE IF D's NodeAgency or AddressAddress exist in N's NNT
    Record the RREQ's identifier
    Append the character '_' to RREQ's identifier
    Append N's agency identifier to RREQ's identifier
    Increment RREQ's TotalClusterheadsTraversed by 1
    Record N's NodeAddress to RREQ's Clusterhead NodeAddress [TotalClusterheadsTraversed]

```
    Unicast RREQ to D
ELSE
    Record the RREQ's identifier
    Append the character '_' to RREQ's identifier
    Append N's agency identifier to RREQ's identifier
    Increment RREQ's TotalClusterheadsTraversed by 1
    Record N's NodeAddress to RREQ's Clusterhead NodeAddress [TotalClusterheadsTraversed]
    Copy the value of TotalClusterheadGatewayPairs to Pointer
    Define and set an integer variable "replica" equal to 0
    Examine N's 2-hop NNT and NCT
    FOR each Cluster-head entry (i) in N's 2-hop NNT and NCT DO
        Examine the RREQ's list of Neighbouring Clusterhead NodeAddresses
        FOR each RREQ's neighbouring Clusterhead NodeAddresses (j) DO
            Compare N's Cluster-head entry [i] to RREQ's Neighbouring Clusterhead NodeAddress [j]
        ENDFOR
        IF a match is found
            Add the Cluster-head entry [i] in the temporary list of excluded cluster-heads (one separate list is dy-
            namically created for each incoming RREQ packet and deleted once the cluster-head dealt with the
            RREQ)
        ELSE
            Examine RREQ's list of Clusterhead NodeAddresses
            FOR each RREQ's Clusterhead NodeAddress (k) DO
                Compare N's Cluster-head entry [i] to RREQ's Clusterhead NodeAddress [k]
            ENDFOR
            IF a match is found
                Add the Cluster-head entry [i] in the temporary list of excluded cluster-heads
            ENDIF
        ENDIF
    ENDFOR
    FOR each Cluster-head entry (m) in N's NNT and 2-hop NNT DO
        IF NOT Cluster-head entry (m) is in the temporary list of excluded cluster-heads
            Clone RREQ packet
            Increment replica by 1
            Append a "_replica" String to the identifier of the cloned RREQ packet, where…
            … "replica" represents the actual value of the variable
            Record cloned RREQ's packet identifier
            FOR each Cluster-head entry (n) in N's NNT and 2-hop NNT DO
                IF NOT Cluster-head entry (n) is in the temporary list of excluded cluster-heads
                    Increment TotalClusterheadGatewayPairs by 1 in the cloned RREQ packet
                    Record Cluster-head's NodeAddress [m] in …
                    …RREQ's Neighbouring Clusterhead NodeAddress [TotalClusterheadGatewayPairs]
                    IF multiple Gateways are found to be leading to the Cluster-head entry [m]
                        Choose the Gateway with the lowest Routing-Metric
                    ELSE
                        Choose the single Gateway
                    ENDIF
                    Record the chosen Gateway's node address in …
                    …RREQ's Gateway NodeAddress [TotalClusterheadGatewayPairs]
                ENDIF
            ENDFOR
            Unicast cloned RREQ to Gateway NodeAddress [Pointer + replica]
```

```
        ENDIF
    ENDFOR
ENDIF
```

Algorithm 3.3 presents the detailed actions performed by a gateway node (*G*) when it receives a RREQ packet. It should be noted that *Gateway* in the following algorithm refers to both *Gateway* and *DistributedGateway* nodes. The gateway's function is simple, in the sense that it is required to forward the RREQ to the destined cluster-head, without performing any modifications. However, if the gateway node is not in direct communication range with the cluster-head, it searches its tables for an intermediate gateway between itself and the cluster-head and performs the following actions: it substitutes its own *NodeAddress* in the RREQ with the *NodeAddress* of the gateway found in its tables; and transmits the RREQ to that gateway. In the case that multiple gateways are found, it then compares their *Routing-Metrics* and transmits the RREQ to the node with the lowest *Routing-Metric*.

Algorithm 3.3: Gateway route request packet handling

```
IF Gateway G finds an entry for D in its NNT table
    Unicast RREQ to D
ELSE IF Gateway node G is specified as a Gateway node to only one Cluster-head C in RREQ packet
        Examine G's NNT
        IF G is directly linked to Cluster-head C
            Unicast RREQ to C
        ELSE
            Examine G's 2-hop NNT
            IF only one Gateway G2 is found such that G2 is directly linked to Cluster-head C
                Modify RREQ by substituting the Gateway NodeAddress entry …
                …which leads to Cluster-head C with G2's NodeAddress
                Unicast RREQ to G2
            ELSE
                IF multiple Gateways are found (Gateways (n)), such that, …
                …each of them is directly linked to Cluster-head C
                    Choose the Gateway with the lowest metric, e.g. Gateway [k]
                    Modify RREQ by substituting the Gateway NodeAddress which leads to …
                    …Cluster-head C with Gateway's [k] NodeAddress
                    Unicast RREQ to Gateway's [k]
                ENDIF
            ENDIF
        ENDIF
ELSE IF Gateway node G is specified as a Gateway node to multiple Cluster-heads …
…(Cluster-heads (n)) in RREQ packet
    Examine the "_i" String in the end of RREQ's identifier
    Examine G's NNT and 2-hop NNT
    IF G is directly linked to Neighbouring Clusterhead NodeAddress [Pointer + i]
        Unicast RREQ to Neighbouring Clusterhead NodeAddress [Pointer + i]
    ELSE
```

```
        IF a Gateway G2 is found, such that, G2 is directly linked to …
        …Neighbouring Clusterhead Node Address [Pointer + i]…
        …and G2 has the lowest Routing-Metric among every other similar option
            Modify RREQ by substituting the Gateway NodeAddress [Pointer + i] to G2's NodeAddress
            Unicast RREQ to G2
        ELSE
            Discard RREQ
        ENDIF
    ENDIF
ELSE
    Discard RREQ packet
ENDIF
```

In relation to Algorithm 3.2 and 3.3, it can be seen that the route request packets' propagation is limited to nodes such as cluster-heads, gateways, and distributed gateways, while member nodes do not participate in this process. A RREQ packet is forwarded from one hop to another, normally along a repeated sequence of alternating cluster-head and gateway node pairs. Using the approach of packet cloning as shown in the Algorithm 3.2, cluster-heads send the RREQ packets along every possible cluster-head path in the ad-hoc network, and thus redundancy is added. In addition, cluster-heads always select the gateway with the lowest routing-metric to forward a RREQ, assuming that multiple gateways exist, and thus RREQs may reach their destination faster and more reliably.

The route in which a RREQ is propagated with purpose to reach the destination will always be in the general form of: *Source*, *Cluster-head 1*, *Gateway 1*, *Cluster-head 2*, *Distributed Gateway 2*, *Distributed Gateway 3*, *Cluster-head 3*, *Gateway 4*, *Cluster-head 4*, … , *Destination*. Thus, the recorded *Clusterhead NodeAddress* list will be: *Cluster-head 1*, *Cluster-head 2*, *Cluster-head 3*, *Cluster-head 4*, … , *Cluster-head n*.

When a node receives one, or more, RREQs, where the packets' *Target NodeAddress* or *Target AgencyAddress* matches its own address(es), it replies back to the source by initiating one, or multiple, RREP packets (*see* Table 3.10). A RREP always follows the cluster-heads path, which the corresponding RREQ has taken. However, in case of multiple gateways, or distributed gateways leading to the next cluster-head, the RREP is cloned and transmitted through each possible combination. In addition, the RREPs gather the routing-metrics of each node they visit. In detail, once the destination node *D*, receives a RREQ packet, it creates a RREP packet with the same identifier and copies the inverted list of *Neighbouring Clusterhead NodeAddress* fields from the RREQ to the RREP's *Clusterhead's NodeAddress* list. It then fills in the first entry of the *Calculated Route* list with its own *NodeAddress* and *AgencyAddress*, and its *Routing-Metric*, and increments *TotalSourceRouteAddressPairs* by one. Node *D* can store the route which the RREQ packet followed, and use it at a later stage, if it

requires to transmit data to the source node *S*. However node *S* may be reluctant to use that route, as it contains no routing-metric information, and thus *S* cannot calculate the QoS.

Table 3.10: The route reply packet format

1 0 0	TotalClusterheadsTraversed	TotalSourceRouteAddressPairs	Identification
Target AgencyAddress			
Originator AgencyAddress			
Clusterhead NodeAddress [1]			
Clusterhead NodeAddress [2]			
…			
Clusterhead NodeAddress [TotalClusterheadsTraversed]			
Calculated Route NodeAddress [1]			
Calculated Route AgencyAddress [1]			
Routing-Metric [1]			
Calculated Route NodeAddress [2]			
Calculated Route AgencyAddress [2]			
Routing-Metric [2]			
…			
Calculated Route NodeAddress [TotalSourceRouteAddressPairs]			
Calculated Route AgencyAddress [TotalSourceRouteAddressPairs]			
Routing-Metric [TotalSourceRouteAddressPairs]			

100: Type of MARIAN packet, RREP

TotalClusterheadsTraversed: The total number of cluster-head node addresses this packet will traverse.

TotalSourceRouteAddressPairs: The total number of addresses in the calculated route, which assumes that a node/agency address counts as one.

Identification: The same identification as in the corresponding RREQ packet.

Target: The agency address of the destination node, which issued the corresponding RREQ.

Originator: The agency address of the source node, which created this RREP.

Clusterhead NodeAddress: The node address of each cluster-head in the sequence for which the RREP packet will have to traverse in order to reach the target. This list of cluster-head addresses is copied from the corresponding RREQ packet, and inverted.

Calculated Route: A sequence of node and agency addresses that provide a route from a source to a destination. This may be used for both static and mobile agent implementation.

Routing-Metric: The preliminary metric array of each device in the calculated routes list (*see* Section 3.13).

A RREP packet always follows the route defined by the cluster-head's *NodeAddress* list in order to reach the source *S*. Each cluster-head, in turn, forwards the packet to the next cluster-head in the list, until the last entry has been reached. The corresponding gateway nodes are not copied from the RREQ packet to the RREP packet, as each cluster-head has knowledge on how to reach its neighbouring cluster-head. Another reason for doing so is due to mobile

gateway nodes which move away, and others, which may appear. Thus, it is more sensible to provide a RREP with knowledge of cluster-heads path rather than the *Neighbouring Cluster-head NodeAddress-Gateway NodeAddress* pairs that the RREQ traversed. Moreover, each time a cluster-head identifies multiple gateways, or multiple distributed gateway pairs leading to the next cluster-head in the list, it clones the original RREP packet and sends each replica to each gateway node.

One novelty of MARIAN is based on the collection of routing-metrics during the RREP's propagation to the source node *S* and in providing multiple redundant routes to a single destination. Each node that the RREP visits, submits its own routing-metric which is related to the node's fitness of routing data. Thus, node *S* can choose the best route according to the type of traffic it wishes to transmit, assuming that multiple RREP were actually produced. For instance, multimedia traffic, which typically has high buffering requirements and requires low latency, can be sent through a route with a low overall routing-metric, while asynchronous traffic, which typically has low buffering requirements, and few latency problems, can be sent through a different route, with, possibly, a high overall metric. Based on this method, the overall performance of the network can be improved, as multiple routes to a destination are utilised, and thus avoids single routes being over-utilisation. Furthermore, redundancy is added so that, if a *NodeAdress* along a source route becomes unreachable at some stage, the source can immediately use an alternative route. Finally, the most significant factor is that lower specification devices will rarely be used as routing elements, especially for heavy network traffic, due to the metric-driven clustering process, and due to the routing-metric calculation, as long as a route with more powerful devices exists. It is thus well suited to emergency situations, where reliability is an important issue.

Similar to CBRP, MARIAN utilises a mechanism by which cluster-heads calculate an optimised hop-by-hop route while forwarding a RREP packet. Accordingly, when a cluster-head receives a RREP packet, it examines if the previously visited node is in direct communication range with the next in the RREP's list. If this is true, the cluster-head forwards the packet without recording its own information in the *Calculated Route* and *Routing-Metric* fields, otherwise, it records it.

Nodes are limited on how many RREQs can be issued at a time. Also, a node which issued a RREQ and has not received a RREP during a certain period of time, enters an exponential backoff algorithm before resending another RREQ. Routes learned throughout the route discovery process are stored in memory cache, thus, when a node requires a route to a destination, it initially checks its memory cache before issuing a RREQ.

Algorithm 3.4 presents the detailed actions performed by a Cluster-head *N* when it receives a RREP packet. In the same way as in previous algorithms, the *Gateway* is referred to gateway nodes as well as distributed gateway nodes.

Algorithm 3.4: Cluster-head route reply packet handling

```
Cluster-head N decrements TotalClusterheadsTraversed by 1
IF NOT TotalClusterheadsTraversed equal to 0
    Cluster-head N examines its 2-hop NNT and NCT
        IF Clusterhead NodeAddress [TotalClusterheadsTraversed] can be reached by only one Gateway G
            Examine N's NNT
            IF Calculated Route NodeAddress [TotalSourceRouteAddressPairs] is directly linked to G
                Unicast RREP to G
            ELSE
                Increment TotalSourceRouteAddressPairs by 1
                Record N's Cluster-head NodeAddress in …
                …RREP's Calculated Route NodeAddress [TotalSourceRouteAddressPairs]
                Record N's Cluster-head AgencyAddress in …
                …RREP's Calculated Route AgencyAddress [TotalSourceRouteAddressPairs]
                Record N's Cluster-head Routing-Metric in…
                … RREP's Routing-Metric [TotalSourceRouteAddressPairs]
                Unicast RREP to G
            ENDIF
        ELSE IF multiple routes are available
            FOR each possible Gateway G to Clusterhead NodeAddress [TotalClusterheadsTraversed] DO
                Clone original RREP packet
                Examine N's NNT
                IF Calculated Route NodeAddress [TotalSourceRouteAddressPairs] is directly linked to G
                    Unicast cloned RREP to G
                ELSE
                    Increment TotalSourceRouteAddressPairs by 1
                    Record N's Cluster-head NodeAddress in the…
                    …RREP's Calculated Route NodeAddress [TotalSourceRouteAddressPairs]
                    Record N's Cluster-head AgencyAddress in the…
                    … RREP's Calculated Route AgencyAddress [TotalSourceRouteAddressPairs]
                    Record N's Cluster-head Routing-Metric in the…
                    …RREP's Routing-Metric [TotalSourceRouteAddressPairs]
                    Unicast cloned RREP to G
                ENDIF
            ENDFOR
        ELSE
            Discard RREP packet
        ENDIF
ELSE
    Increment TotalSourceRouteAddressPairs by 1
    Record N's Cluster-head NodeAddress in the…
    …RREP's Calculated Route NodeAddress [TotalSourceRouteAddressPairs]
    Record N's Cluster-head AgencyAddress in the…
    … RREP's Calculated Route AgencyAddress [TotalSourceRouteAddressPairs]
```

Record N's Cluster-head Routing-Metric in RREP's Routing-Metric [TotalSourceRouteAddressPairs]
Examine N's NNT
IF a NodeAddress entry in N's NNT matches the IP address of the IP packet…
… or an AgencyAddress entry in N's NNT matches the target's AgencyAddress of RREP packet
Unicast RREP to target
ELSE
Discard RREP packet
ENDIF
ENDIF

When a gateway node receives a route-reply, it performs the actions outlined in Algorithm 3.5, where the *Gateway* refers to both *Gateway* and *DistributedGateway* nodes.

Algorithm 3.5: Gateway route request packet handling

Gateway G increments TotalSourceRouteAddressPairs by 1
Records G's NodeAddress in Calculated Route NodeAddress [TotalSourceRouteAddressPairs]
Records G's AgencyAddress in Calculated Route AgencyAddress [TotalSourceRouteAddressPairs]
Records G's Routing-Metric in Routing-Metric [TotalSourceRouteAddressPairs]
Examines G's NNT
IF Clusterhead NodeAddress [TotalClusterheadsTraversed] is directly linked to G
Unicast RREP to Clusterhead NodeAddress [TotalClusterheadsTraversed]
ELSE
Examine G's 2-hop NNT
IF Clusterhead NodeAddress [TotalClusterheadsTraversed] is directly linked to only one Gateway G2
Unicast RREP to G2
ELSE IF multiple Gateways (Gateways (n)) are available
FOR each Gateway (n) leading to Clusterhead NodeAddress [TotalClusterheadsTraversed] DO
Clone original RREP packet
Unicast cloned RREP to Gateway [n]
ENDFOR
ELSE
Discard RREP packet
ENDIF
ENDIF

3.6.2 Proactive route discovery

Each cluster-head has the right to create a network discovery mobile agent with purpose to collect the topology of the whole ad-hoc network. No other key or ordinary node has the same right. Cluster-heads exercise this right under a controlled manner, for example, on triggered events. Such events may include the following:

- A registered agency requested this action.
- A foreign, but authenticated mobile agent, requested this action.
- A local timer has expired.

- The cluster-head has heard enough RREQs from its members, specifically more than a certain threshold value.

Triggered events can be configured in advance, or, in real-time. For instance, the administrator of an ad-hoc network can pre-configure a number of triggered events before the network gets into operation, or, dynamically dispatch a number of autonomous, trusted mobile agents for reconfiguration purposes, while the network is in operation, and thus allowing the network to cope with dynamic changing environmental factors, such as when overall mobility changes. For example, in situations where the mobile devices of an ad-hoc network are stationary for long periods of time, it may be best to configure the cluster-head's dispatch timer to expire infrequently, or even set it to infinity. In this way, the mobile agents collect the whole networks topology, in the beginning, and settle down, as they will no longer be needed, as topological updates will be uncommon.

Mobile agents can exist in one of two states: *Exploring* or *ReturningHome*. When a mobile agent is in *Exploring* state, its goal is to collect routing information from its docking cluster-head, while, when in *ReturningHome* state, its goal is to return to its home platform, and submit the topology information collected throughout the *Exploring* state. Initially, an *explorer* agent examines the 2-hop NNT and NCT of its docking cluster-head in order to identify the intersecting and adjacent cluster-head *AgencyAddresses*. It then clones itself, and sends one copy to each intersecting and adjacent cluster-head, as long as the destination cluster-head has not been visited by itself, or from one of its previous clones. Even though the agent's destination is set to the cluster-head's *AgencyAddress*, cluster-heads are never in direct communication range, and thus the agent is initially dispatched to the intermediate gateway with the lowest routing-metric, which leads to the agent's final destination. The parent agent passes the following arguments to each cloned agent upon creation, and before dispatch:

- **The destination cluster-head *AgencyAddress***. This *AgencyAddress* is set to the intersecting or adjacent cluster-head's *AgencyAddress*, that is, the agent's final destination.
- **Parent's agent information *Object***. This allows the agent to contact its parent when it returns to its home-platform.

The mobile agent stores and maintains the following information, which carries it with itself during its self-migration, and throughout its existence:

- ***Destination Clusterhead AgencyAddress***. This is the destination which was assigned by its parent agent.

- ***Neighbouring Clusterhead AgencyAddresses* list**. This is a list of cluster-heads that has been visited by other cloned agents and it thus must be avoided by this agent.
- ***Clusterhead AgencyAddresses* list**. This is a list of cluster-heads that has been visited by its ancestor agents and it thus must be avoided by this agent.
- ***RoutingInformation***. This is the network's topology information collected so far, and is in the form of NNT, 2-hop NNT, and NCT.

The list of excluded cluster-heads is declared *transient*, and, thus, it is not carried along with the agent's self-migration, and, thus, this list is emptied each time the agent migrates to a new cluster-head. Algorithm 3.6 presents the actions performed by a network discovery mobile agent (MA) when it is initiated by a cluster-head. Initially, the creator cluster-head sets the mobile agent's goal to *Exploring*.

Algorithm 3.6: Network discovery mobile agent actions when residing on a cluster-head

```
MA examines its goal
IF goal is equal to Exploring
    MA stores C's AgencyAddress in the list of Clusterhead AgencyAddresses (maintained by the MA)
    MA examines C's 2-hop NNT and NCT
    Define and set integer variable "replica" equal to 0
    Examine N's 2-hop NNT and NCT
    FOR each Clusterhead entry (i) in N's 2-hop NNT and NCT DO
        MA examine the list of Neighbouring Clusterhead AgencyAddresses (maintained by the MA)
        FOR each MA's Neighbouring Clusterhead AgencyAddress (j) DO
            MA compares N's Clusterhead entry [i] to its Neighbouring Clusterhead AgencyAddress [j]
        ENDFOR
        IF a match is found
            Add the Clusterhead entry [i] in the transient list of excluded cluster-heads (the mobile agent…
            … creates and remembers the contents of this list while awake; however, the list's contents…
            … are not being carried along with the agent's self migration)
        ELSE
        MA examines the list of Clusterhead AgencyAddresses (maintained by the MA)
            FOR each MA's Clusterhead AgencyAddress (k) DO
                MA compares N's Clusterhead entry [i] to its Clusterhead AgencyAddress [k]
            ENDFOR
            IF a match is found
                Add the Cluster-head entry [i] in the transient list of excluded cluster-heads
            ENDIF
        ENDIF
    ENDFOR
    FOR each Cluster-head entry (m) in N's NNT and 2-hop NNT DO
        IF NOT Cluster-head entry (m) is in the temporary list of excluded cluster-heads
            MA clones itself (creates an explorer MA)
            MA passes its agent info (parent) Object to its cloned copy upon creation
            MA passes the Cluster-head entry [i] to its cloned copy upon creation, which is stored in…
            … the clone's Destination Clusterhead AgencyAddress
```

```
            FOR each Cluster-head entry (n) in N's NNT and 2-hop NNT DO
                IF NOT Clusterhead entry (n) is in the temporary list of excluded cluster-heads
                    MA passes the Clusterhead's AgencyAddress [m] to its cloned copy upon creation, which is…
                    …stored in the clone's Neighbouring Clusterhead AgencyAddresses list
                    IF MA finds multiple Gateways leading to the Cluster-head entry [m]
                        MA dispatches its cloned copy to the Gateway with the lowest Routing-Metric
                    ELSE
                        MA dispatches its cloned copy to this single Gateway
                    ENDIF
                ENDIF
            ENDFOR
        ENDIF
    ENDFOR
    MA collects C's NNT, 2-hop NNT, and NCT and stores them to memory
    IF at least one cloned copy of the MA has been created
        MA sets a clone_timer
        WAIT for clone_timer
            MA constantly checks for updates in C's NNT, 2-hop NNT, and NCT tables
            IF new information become available
                MA updates information stored in its memory
            ENDIF
            IF cloned agent contacted this MA
                MA collects NNT, 2-hop NNT, and NCT from cloned copy
                MA stores information in its memory
                MA filters data and removes redundant information (e.g. an indirect link from MA's docking…
                …cluster-head (C) to intersecting cluster-head (C1) through gateway (G1) may be known…
                … by the MA and cloned copy)
            IF MA collected network topology from all of its cloned copies
                Stop waiting
            ELSE IF clone_timer expired
                Stop waiting
        ENDWAIT
    ENDIF
    MA sets its goal to ReturningHome (home is the agent's originating platform)
    MA invokes itself
ELSE IF goal is equal to ReturningHome
    MA checks the docking platform's agency identifier and compares it to its home platform's agency identifier
    IF NOT a match is found
        MA examines C's NNT, 2-hop NNT, and NCT
        IF MA's home platform's AgencyAddress is found in C's Neighbouring Cluster-head AgencyAddresses list
            IF MA finds multiple Gateways leading to its home platform
                MA sets its Destination Clusterhead AgencyAddress to its home platform's AgencyAddress
                MA migrates to the Gateway with the lowest Routing-Metric
            ELSE
                MA sets its Destination Clusterhead AgencyAddress to its home platform's AgencyAddress
                MA migrates to this single Gateway
            ENDIF
        ELSE
            MA sets a waiting_timer
            IF waiting_timer has expired and a Gateway G is not found, such that, G leads to its home platform
                MA kills itself
```

```
            ELSE
                MA sets its Destination Clusterhead AgencyAddress to its home platform's AgencyAddress
                MA migrates to the Gateway G
            ENDIF
        ENDIF
    ELSE
        MA updates the RoutingInformation of its home platform
        MA tries to contact its parent MA
        IF NOT parent is available or exists
            MA kills itself
        ELSE
            MA passes the new RoutingInformation to parent
            MA kills itself
        ENDIF
    ENDIF
ENDIF
```

Once a network discovery mobile agent arrives at a gateway node, either in *Exploring* or *ReturningHome* state, it performs the same exact actions, as outlined in Algorithm 3.7, where the *Gateway* (G) refers to both *Gateway* and *DistributedGateway* nodes.

Algorithm 3.7: Network discovery mobile agent actions when residing on a gateway

```
MA examines its goal
IF MA's goal is equal to Exploring or ReturningHome
    MA examines G's NNT and 2-hops NNT
    IF the MA's Destination Clusterhead AgencyAddress is in G's NNT (direct communication range)
        MA migrates to Destination Clusterhead AgencyAddress
    ELSE IF Destination Clusterhead AgencyAddress is in G's 2-hop NNT (indirect link)
        IF multiple Gateways (n) leading to Destination Clusterhead AgencyAddress
            Migrate to Gateway AgencyAddress which has the lowest Routing-Metric
        ELSE
            MA migrates to this single Gateway
        ENDIF
    ELSE
        MA sets a waiting_timer
        IF waiting_timer has expired and a Gateway G is not found, such that, G leads to its home platform
            MA kills itself
        ELSE
            MA sets its Destination Clusterhead AgencyAddress to its home platform's AgencyAddress
            MA migrates to the Gateway G
        ENDIF
    ENDIF
ENDIF
```

The network discovery mobile agents propagate through the network in a similar manner to the RREQ packets, however, during both *Exploring* and *ReturningHome* phases, each agent

travels a maximum distance of three hops for each phase. This is because, while *Exploring*, a mobile agent is allowed to migrate from a cluster-head to only one intersecting or adjacent cluster-head, and, while *ReturningHome*, from that cluster-head back to the originator cluster-head. The grandparent of all agents, which is created by the cluster-head, that issued this proactive route discovery, remains stationary in the originator cluster-head, and dispatches its clones for this purpose. Once the agent clones return back, the grandparent agent obtains the NNTs, 2-hops NNTs, and NCTs of each key node in the network. Thus, the agent will be able to create a complete roadmap of the ad-hoc network, and associate an overall routing-metric to each derived route. The complete routing information of the network can then be stored in the cluster-head's routing table. The detailed description of the process involved in deriving the network's roadmap is out of the scope of this thesis, as it can use techniques, such as sorting and matching algorithms. However, the association of retrieved routes with an overall routing-metric is thoroughly analysed in Section 3.13.

3.7 MARIAN source routing - static approach

Data packet routing is performed in a source-routing manner, rather than a hop-by-hop. This means that a data packet carries the complete ordered list of node addresses that it traverses over in order to reach the destination. The main advantage of this is that network nodes do not have to maintain up-to-date hop-by-hop routing information. In addition, MARIAN provides the functionality for agent-based source routing, as a source node has knowledge of each node's *AgencyAddress* along a route (*see* Section 3.8).

A node originating a packet, either for the purpose of route discovery, route error, or source routing, has to initially construct a MARIAN header and add it to the packet along the following sequence of steps (assuming the inexistence of other headers that need to be placed before the MARIAN header):

- The node inserts a MARIAN header after the IP header, but before any other headers that may be present.
- The node sets the *NextHeader* field of the MARIAN header to the *Protocol* number field of the packet's IP header.
- The node sets the *Protocol* field of the packet's IP header to the *Protocol* number assigned for the MARIAN header.

The MARIAN header format is shown in Table 3.11:

Table 3.11: The source routing packet format

NextHeader	PayloadLength
Option	

NextHeader: Identifies the header type which is immediately following the MARIAN header.
PayloadLength: The total length of the Option field, excluding the header's fixed portion.
Option: Options include: Minimal Route-Request, Minimal Route-Reply, Route-Not-Available, Route-Reply, Route-Error, and Source-Route. Only one of these options may be included in a single MARIAN header.

When a source node originates an IP data packet for a destination node *D*, which is not in direct communication range, it modifies the packet and includes the MARIAN header as previously described, and it constructs a source-routing packet as shown in Table 3.12.

Table 3.12: The source routing packet format

0 0 0	TotalSourceRouteAddresses	Pointer
SourceNodeAddress [1]		
SourceNodeAddress [2]		
…		
SourceNodeAddress [TotalSourceRouteAddresses]		

000: Type of MARIAN packet, source routing.
TotalSourceRouteAddresses: The total number of node addresses in the source route ([1 … TotalSourceRouteAddresses])
Pointer: A pointer used to specify the currently visited node address.
SourceNodeAddress: The node address of an intermediate node which the data packet has to traverse in order to reach its destination.

The source routing option shown in Table 3.11 is then appended to the MARIAN header. Once the data packet is completed, the source node transmits it to the next hop along the source route. When a node receives a source routing data packet it performs the following steps:

- **It examines the IP header's destination address**. If the node finds that the IP address matches its own address, it consecutively infers that the packet is destined for itself, and performs no further actions concerning source-route forwarding, otherwise it continues.

- **It examines the SourceNodeAddress [Pointer]**. If this does not match the node's IP address, it discards the data packet. Otherwise, it increments the *Pointer* by one, and transmits the data packet to SourceNodeAddress [Pointer].

In case that the transmission along the next hop in the source route fails, which may be a result of various factors such as nodal movements, the intermediate node constructs a route error packet and transmits it back to the source node. The packet's format is shown in Table 3.13.

Table 3.13: The source routing packet format

1 0 1	TotalSourceRouteAddresses
SourceNodeAddress [1]	
SourceNodeAddress [2]	
…	
SourceNodeAddress [TotalSourceRouteAddresses]	
BrokenLinkFromNodeAddress	
BrokenLinkToNodeAddress	

101:	Type of MARIAN packet, route-error.
TotalSourceRouteAddresses:	The total number of node addresses in the source route ([1… TotalSourceRouteAddresses])
SourceNodeAddress:	The node address of an intermediate node which the error packet has to traverse in order to reach the destination node (the source node which initiated the source-routing data packet, for which an error occurred).
BrokenLinkFromNodeAddress:	The RERR packet's originator.
BrokenLinkToNodeAddress:	The unreachable next hop node address, as specified in the original source route packet.

When a source node is informed of a broken route, by means of a RERR packet, it resumes its data transmission over an alternative route, either found in its routing cache, or in the cluster-head's routing table. The alternative new route, of course, should match the source node's QoS requirements. However, if such a route is not found by either these two methods, the source node may initiate a new route discovery process.

3.8 MARIAN source routing - Mobile agent approach

This approach utilises the full potential of the mobile agent paradigm, and may be adaptable to various implementations, and thus this section provides a general guide on how this could be achieved. Initially, a source node which has some data to transmit to a destination node,

in a multi-hop manner, creates a routing mobile agent. The originator node initialises the agent by performing the following steps:

- Setting the home agent platform of the mobile agent to its own *AgencyAddress*.
- Setting the destination agent platform of the mobile agent to the *AgencyAddress* of the destination.
- Setting the itinerary of the mobile agent to the list of source *AgencyAddresses*.
- Passing the transmission data to the agent's payload.

Upon creation, the mobile agent will autonomously, and sequentially, migrate to each *AgencyAddress* found in its itinerary, and, finally, to the agent's ultimate destination, where the data will be delivered. In practice, this is not an efficient method, as the agent migration times are significantly greater than the source-routing data packet propagation (*see* Section 5.4). However, the delivery time may not be the sender's crucial requirement, as other factors may influence this decision, including: guaranteed delivery; robustness; and security. A routing mobile agent could possibly provide the following advantages:

- **Successful propagation over unreliable links**: A routing agent who according to its itinerary requires to migrate to a next-hop, although the communication link turns to be unavailable, it can wait for a certain amount of time until the link is re-established, while performing some other task.
- **Dynamic alteration of its itinerary**. A routing agent can dynamically alter its itinerary, in the case that a source route which is equipped with, turns out to be inaccurate. For instance, if the agent's next hop is unreachable, the agent can bypass that hop and replace its itinerary with an alternative route. This could be achieved by either taking into advantage the information maintained on the current node, or, by issuing a route discovery process for the hop following the unreachable next hop.
- **Confidentiality of information**. Mobile agents can be initially used to distribute the public keys of their users, who are willing to participate in confidential communications. Then, a routing agent can encrypt its user's message with the other user's public key and append the encrypted message to its payload. The agent can then deliver the message at the destination, where it can get decrypted with the corresponding private key.

3.9 Agent-based metric-driven routing

The majority of ad-hoc routing protocols suffer from sub-optimal route identification, typically based on hop-counting mechanisms, which generally underestimate the importance of the routing devices' performance characteristics and their current utilisation status. In particular, a route is represented by the number of intermediate nodes that need to be traversed to reach the destination. For example, a route with three intermediate nodes is considered to be stronger than a route with more than three. However, this approach oversimplifies such a complex decision by ignoring the fact that participating devices may have considerably unequal performance characteristics and current utilisation status. Accordingly, it is possible for a *best* route to be composed of devices with high utilisation status or low battery level, which results in an overall unreliable route. If this information was made available to the routing protocol on which it based its routing decisions, it would have a significant impact on the network's overall performance and reliability. In addition, different routing scenarios impose different requirements, and thus a path may be ideal for a certain routing objective but inappropriate for another.

MARIAN employs a metric-driven routing approach, which bases its routing decisions on various key metrics, such as the devices' processing power, memory capacity, battery reserves, network reliability, routing throughput, utilisation status, and so on. This section outlines the Benchmarking multi-Agent Software System (BASS), which can be executed by resource-constrained devices, and aims to benchmark the fitness of various ad-hoc device types as routing elements. It achieves this by performing a number of tests, that is, tests that are executed once and tests that constantly monitor the device's resources, where test results are used to produce an overall routing metric, which is tailored to the needs of various routing scenarios. For this purpose, BASS incorporates various test agents, which can cooperatively calculate the routing ability of a device, including: 1D bubble sort; CPU merge; memory test; client-server throughput; proxy throughput; TCP error; IP error; UDP error; CPU utilisation; memory usage; and battery level.

The 1D bubble sort agent performs an intensive sorting algorithm, and can thus be used to benchmark the devices processing power, which may be equivalent to the intensive processing tasks that are typically required by routing, whereas the CPU merge agent performs a less intensive sorting algorithm, and thus provides an alternative test for devices that may not be able to execute the 1D bubble sort, such as mobile phones, and so on. The memory test agent creates a number of files of varying sizes, and a varying number of files of constant sizes, and can thus benchmark the buffering capabilities of resource-constrained devices, as they use the RAM as their persistent storage. Ad-hoc routing devices typically buffer incom-

ing data, in the case of routing congestion, and thus the memory test can provide a useful benchmark. The client-server test is used to benchmark the throughput, when devices are in direct communication range, whereas the proxy test is used to benchmark the throughput that a routing device can offer, and is thus an ideal test for ad-hoc routing devices. This is due to the fact that by knowing, in advance, the devices' routing speeds, it can assist the routing protocol to decide on *optimal* routes for network traffic that requires fast delivery. The TCP, IP, and UDP error tests constantly monitor the network protocol errors, and are thus particularly useful, as frequent protocol errors can compromise the reliability and efficiency of an ad-hoc routing device. The CPU utilisation and memory usage tests constantly monitor the device's overall utilisation, which typically increases while the device is routing data, and are thus particularly useful in benchmarking the current status of a routing device, and further protect it from over-utilisation. Finally, the battery level test constantly monitors a device's battery reserves, and is used as a key routing metric, as remaining battery life is, perhaps, a device's most valuable resource, and as shown in Section 5.1, the battery discharge rate is significantly reduced while a device has its wireless feature on, in comparison to when it is in an idle state. These agents are described in more detail in the next section.

3.10 Benchmarking multi-Agent Software System (BASS)

The main objective of BASS is to gather system performance, and utilisation status information, and use this to derive an overall routing metric of the device. Performance tests are scheduled to execute in a periodic, or in an on-demand fashion. This is important due to the unstable nature of ad-hoc environments, that is, that optimal routes often die due to mobility issues, and thus optimal routes need to be re-discovered. The tests are implemented in such a way as to not waste the mobile device's processing power. An extensive study in energy efficient routing protocols is provided in (Buchanan, W. J., et. al., 2004a).The performance of ad-hoc routing often depends on processing strengths, memory and buffering capabilities, battery capacity, and the networking capabilities of the devices (Buchanan, W. J., et. al., 2004a). Thus, performance tests were designed to exercise the strength of devices based on these factors. In general, they can be grouped into the following categories:

- **Kernel level**. These tests aim to identify the hardware characteristics of the device.
- **Network level**. These tests aim to determine the current status of an ad-hoc network, from the device viewpoint.
- **Application level**. These tests aim to monitor the system's utilisation, at a given time.

- **Group level**. These tests aim to group devices according to common characteristics that they may share. For example, all devices with the same operating system (OS) may require to run a specific test, in a specific way.

The most critical factor for mobile devices is typically the battery reserves. If possible, unless in emergency applications, devices with low battery reserves should not be used for routing, as they are likely to become unavailable within a short time. More precisely, PDAs, typically, turn off their wireless activities when the battery reserves drop below a certain level. The control of the power consumption of mobile devices in ad-hoc networks, largely improves the battery reserves, and can extend the lifetime of the network. Another equally important factor is memory buffering, which is particularly important in a PDA device, as memory is a valuable resource as the data is held in RAM, and not on a hard drive. This, therefore, limits the amount of data that can be buffered to the amount of available memory.

BASS (Buchanan, W. J., et. al., 2004a) is especially designed to support MARIAN's routing metric determination process (Migas, N. and Buchanan, W. J., 2005), however, it can be adaptable to any metric-driven routing protocol. This is due to the fact that BASS is decoupled from MARIAN, in a way that BASS produces a number of preliminary metrics based on test results, while MARIAN uses these to produce a capability/incapability policy targeted for each device to accomplish various routing tasks, as well as the route(s) in which a device is positioned to, in a particular instance (*see* Section 3.13). Figure 3.7 presents MARIAN's overall architecture (Migas N. et al, 2003b) and the layer in which BASS lies. The architecture can be logically divided into the following:

- **Foundation**. This is the physical layer and consists of all mobile nodes in an ad-hoc network. BASS resides on each fixed, or mobile, device in the foundation layer.
- **Intermediate**. This layer sits on top of the foundation layer, and is divided into two categories: the stationary agent model, and the mobile agent model.
- **Core layer**. This is top layer and is a combination of the stationary and mobile agent models.

3.11 BASS overall architecture

BASS is designed on a multi-agent principle, where tests are represented by goal-oriented stationary agents, which report their findings to their supervisor agents. BASS is implemented in Java language, and conforms to the J2ME specification (*see* Appendix A), which enables it to execute on resource-constrained devices, such as PDAs which are J2ME-

enabled. However, due to J2ME's limitation to extract low-level information, such as the battery reserves of a PDA, the test has been developed in C language, and it is called by Java code, through the use of JNI (Sun, Microsystems, 2003c). Figure 3.8 presents the high-level design of BASS.

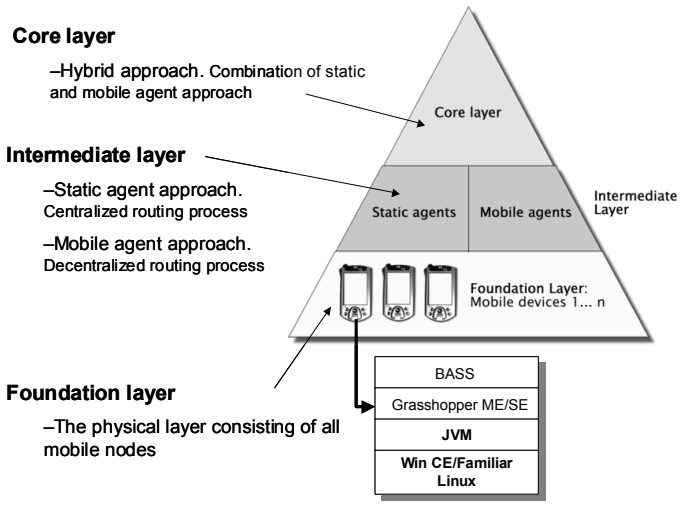

Figure 3.7: BASS position in MARIAN's overall architecture

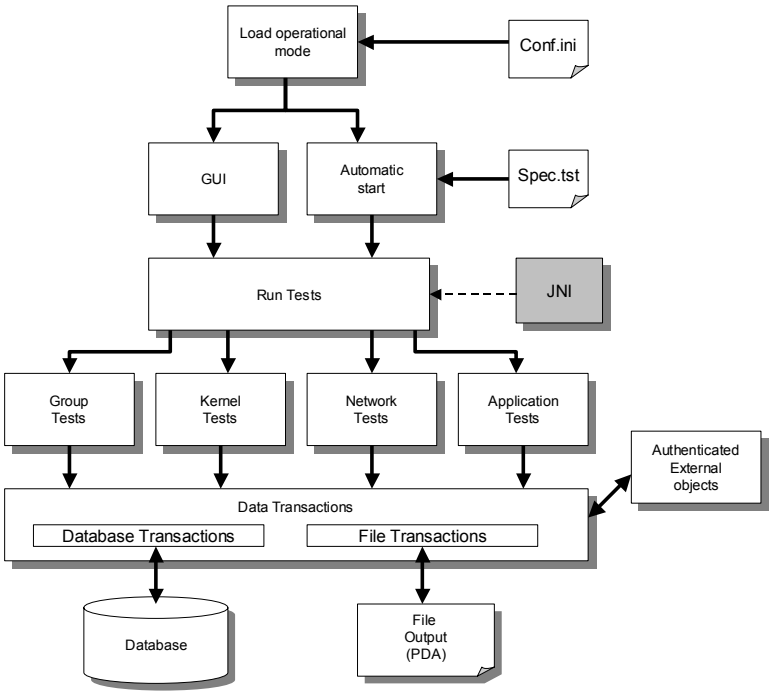

Figure 3.8: BASS architecture

3.12 BASS multi-agent model

As previously mentioned, each performance test has been designed as a stationary agent, where each test agent is responsible for conducting the corresponding test exactly the same way as described in a test specification file. In addition to test agents, supervisor agents are responsible for the control of test agents that belong to their group. Currently, there are four supervisor-agents: **group-level agent**[1] (*see* in Figure 3.9), **kernel-level agent**, **network-level agent**, and **application-level agent**. The group-level agent is the only exception to the above rule as it does not supervise any other agents, and conducts the test on its own. In addition, this agent automatically conducts its task on system start-up, as it is necessary for the system to know on which device it is running, and thus adjusts itself, accordingly.

Test agents can be generally grouped into two categories: continuous and preliminary. In the former, test agents run continuously in the background, aiming to monitor device's resources, while in the latter, test agents are usually executed once, aiming to benchmark the device's hardware characteristics. In terms of device's resource consumptions, continuous tests agents are light-weighted and are thus not overloading the device, however, preliminary test agents may need more processing power and resources to complete their tasks. Continuous agents include the memory and battery monitoring agent, the temperature variation monitoring agent, the Internet connectivity test agent, the error packets monitoring agent, the CPU, memory, and overall utilisation agent, the heap memory usage monitoring agent, and the Java threads monitoring agent. Preliminary agents include the memory test agent, the CPU bubble sort test agent, the CPU merge test agent, the client-server throughput agent, the proxy throughput agent, and the group-level agent.

Each supervisor-agent reads the test specification file and, accordingly, creates an instance of the required agents to deal with the demand. Supervision-level agents then pass the required parameters to performance agents that describe the frequency of the test execution, the delay between test iterations, and the specification test arguments. Supervisor-agents thus constantly look for changes in the test specification. Once changes have been made, the supervisor-agents kill any running test agents that are no longer necessary, and re-instantiate the required ones. When test agents have results from the conducted tests, they report them to supervisor-agents, which, in turn, forward them to the **test results gathering agent**. Once the test results gathering agent has sufficient amount of information, it forwards them to the **librarian agent** for database or file storage. The librarian agent is then responsible for forwarding the results to other authenticated agents, objects, and entities, such as the **metric calculation agent**, which initially produces a preliminary metric for each test, represented from one to 100 (the lower the value, the fitter it is for this test) and finally calculates the

routing metric of the device associated with available routing scenarios. This information is then made available to MARIAN's route discovery process, which builds up an overall routing metric representing the fitness of a route to accommodate various routing scenarios.

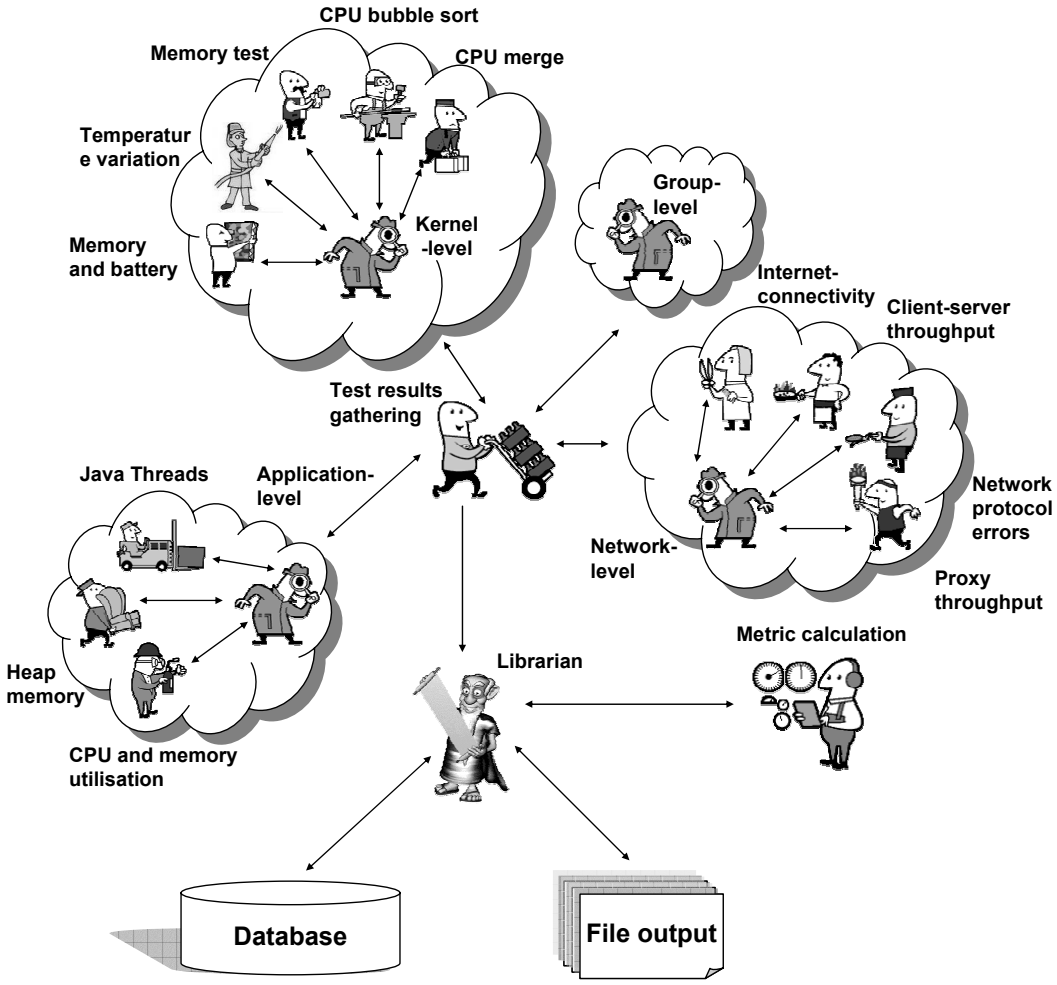

Figure 3.9: BASS Multi-agent model based on both intelligent stationary and mobile agents

Kernel-level agent can create an instance of the following agents: **memory and battery monitoring agent**, **temperature variation monitoring agent**, **memory test agent**, **CPU bubble sort test agent**, and **CPU merge test agent**. The memory and battery agent monitoring is responsible for extracting memory usage and battery levels when changes occur. Readings are then passed to the kernel-level agent, on request, or in a periodic fashion. The temperature variation monitoring agent records the variation in the device's internal temperature each time the battery drops by 1%. The memory test agent is responsible for conducting two tests, which both benchmark the buffering capabilities of the hardware system (applicable to PDAs

only) and the speed of the hard-drive (applicable to non-PDAs). For this, it measures the time taken to create varying number of files with constant file-sizes, and the time taken to create a constant number of files with varying file-sizes. The CPU bubble sort test agent is responsible for sorting a number of random integer values using a bubble sort algorithm. It aims to benchmark the strength of the CPU to perform complex calculations, allowing four levels of intensity (1, 2, 3, and 4-dimensional array of integer values), however, the 2D, 3D, and 4D, as shown in Section 5.3.2, do not provide any additional useful information in terms of ad-hoc routing, and thus only the 1D bubble sort test is used by the metric calculation process. The CPU merge test agent is similar to CPU bubble sort test agent, however, it uses a simpler algorithm to sort a number of random integer values. It is used to benchmark devices with limited processing capabilities, such as mobile phones, which are not capable of conducting the 1D bubble sort test.

The network-level agent can create an instance of the following agents: the **proxy throughput agent**, the **error packets monitoring agent**, the **client-server test agent**, and the **Internet connectivity test agent**. The proxy is light-weighted, and has been developed as an alternative to proper routing software, as it bases its functionality on simple mechanisms, such as Java *Sockets* and *Threads*, which enable resource-constrained devices to execute routing tasks. In particular, the agent constantly listens for incoming network traffic in a pre-defined port, and, once an incoming connection is established, it creates an outgoing connection on another pre-defined port, and forwards the data. In this manner, the agent is simple as it does not require complex routing calculations, which is often the case for proper routing software, as it bases its functionality on higher layers of the network protocol stack. The details of the process of benchmarking the routing capabilities of a proxy-enabled ad-hoc routing device, as well as the monitoring of resource-consumption rates, such as the battery discharge rate, CPU utilisation, and so on, can be found in Appendix A. The error packets monitoring agent is responsible for monitoring the current network state. It calls native code via the JNI to calculate current TCP, UDP, and IP data statistics and errors. This test is particularly useful in ad-hoc routing, where frequent errors can occur because of the limitations of the wireless medium, and can thus be used to determine the network reliability of a routing device. The client-server throughput agent is responsible for measuring the throughput between any pair of nodes in an ad-hoc, or, a fixed network. It requires a client node, and a server node, where raw data are passed from the client to the server, and then back to the client. This process allows the calculation of throughput between two nodes within communication range, and can be used by the routing protocol as an additional factor, which may assist in determining the network state between a node and its neighbouring nodes. The Internet connectivity test agent attempts to download an HTML page from a

remote server. If successful, it calculates the time to connect and the time to download, and shows that there is a connection. If unsuccessful, it shows that there is no TCP connection for HTTP, from the viewpoint of the device. Although this test is not related to ad-hoc routing, and is thus not included in the metric calculation process, it may provide information on the connection and download speeds of resource-constrained devices.

The application-level agent can create instances of the following agents: the **CPU, memory, and overall utilisation agent**, the **heap memory usage monitoring agent**, and the **Java threads monitoring agent**. The CPU, memory, and overall utilisation agent is responsible for obtaining current CPU and memory usage per process running in the system, as well as total system utilisation. It calls native code via JNI to achieve this. Results from this test can be used from the routing protocol in order to protect ad-hoc routing devices from over-utilisation, which can occur from frequent routing requests. The heap memory usage agent is slightly different than the CPU, memory, and overall utilisation agent in the respect that it extracts heap memory usage, which is utilised by Java *Objects*, instead of measuring the total memory utilisation. Even though test results from the heap memory usage agent are not used by the metric calculation process, as they are not relevant with ad-hoc routing, they may be useful in determining the performance of the JVM, which is used to interpret the Java-based proxy agent (*see* Section 5.2). The Java threads monitoring agent is responsible for obtaining the total number of Java *Threads* running on the device, and the amount of CPU they utilise, and, possibly, it can be used to detect malicious agents, which initiate denial of service attacks, and thus stop their execution. However, test results are not included in the metric calculation process.

3.13 Ad-hoc routing metrics and applied weighting for QoS support

This section describes the process of assigning a routing metric to an ad-hoc routing device, which is based on the test results that were previously described. Initially, for each test that participates in the metric calculation process, a preliminary metric is calculated, which is then appropriately weighted to suit various routing objectives, and is then averaged with the remaining weighted preliminary metrics. In this manner, a number of overall routing metrics is calculated, which represent the routing fitness of a device to achieve various routing objectives, such as to route synchronous network traffic, asynchronous network traffic, and so on. As previously mentioned, the nodes' overall routing metrics are gathered by MARIAN's on-demand route discovery and proactive network discovery processes. This information is used by a source node to determine the capability/incapability of each retrieved route to accom-

plish the source's routing scenario. Thus, the source estimates a final metric for each of the capable routes, which represents the QoS that the routes can offer, and thus it bases its route selection on the requirements imposed by its routing scenario.

Preliminary metrics calculation

The tests that participate in the preliminary metrics calculation process are outlined in Table 3.14. Briefly, the 1D bubble sort test is used to determine the processing strength of an ad-hoc routing devices, while the CPU merge, although serves the same purpose, it is intended to be used by limited devices, which cannot execute the intensive 1D bubble sort. The memory tests are used to benchmark the buffering capabilities of resource-constrained devices, as they typically use their RAM as persistent storage. The client-server throughput test is used to benchmark the available throughput between two neighbouring devices, while the proxy throughput is used to determine the routing fitness of a proxy-based ad-hoc routing devices, which is particularly important, as this information is used by the routing protocol to determine *optimal* routing paths for network traffic that requires low latency. The protocol error tests (T_7-T_9) are used to determine a device's current network state in terms of routing reliability, which is particularly important in ad-hoc routing, as frequent network errors can considerably reduce a device's routing ability. The CPU utilisation and memory usage tests are used to estimate a device's remaining resources, and thus protect the device from performing routing tasks that exceed its capability. The battery tests is, possibly, the most important one, as ad-hoc devices normally rely on battery power for operation, and thus this information can be efficiently used by the routing protocol to provide energy conservation.

Table 3.14: Test that count towards the preliminary metrics calculation process

Test	*Symbol*
1D Bubble sort	T_1
CPU Merge	T_2
memory 1 File	T_3
memory 1 KB	T_4
Client-Server throughput	T_5
Proxy throughput	T_6
TCP error	T_7
IP error	T_8
UDP error	T_9
CPU utilisation	T_{10}
Memory usage	T_{11}
Battery level	T_{12}

For each of the tests presented in Table 3.14, a preliminary metric (pm) is calculated, based on the results achieved by each test. The process of creating the pm(s) is based on either a function, or a threshold value:

- **Function**. Using this method, results acquired from a certain test are passed to a function which produces a preliminary metric. Each test is using a distinct function for its metric calculation.
- **Threshold value** (**TH**). The main element in this method is a threshold value, which represents the worst case scenario for a test. Using this value a preliminary metric can be calculated based on the test results.

The preliminary metrics for tests T_1 - T_6 are calculated based on a threshold basis while tests T_7 - T_{12} are calculated using functions. Table 3.15 presents the default threshold values for some of these tests.

Table 3.15: Threshold values for preliminary metric calculation

Test	Threshold value (TH)
1D Bubble sort	500
CPU Merge	100
memory 1 File	20
memory 1 KB	7000
Client-Server throughput	80
Proxy throughput	350

The threshold values were derived from the experimentation phase (*see* Sections 5.1-5.3), and represent the worst case test results, that an ad-hoc routing device can achieve. A device which achieves results equal to a threshold, or above, is determined incapable of routing, and is thus assigned an infinity routing metric (∞). The mathematical expression that is used to estimate the preliminary metric (PM), given the values of the test results (T_n) and threshold value (TH_n) for that test is:

$$PM = \left(\frac{T_n}{TH_n} \right) \times 100 \tag{3.1}$$

The preliminary metrics for tests T_5 - T_7 are calculated using the mathematical expressions in:

$$PM_5 = \frac{TCP_{error}}{TCP_{in} + TCP_{out}} \times 100 \quad (3.2)$$

$$PM_6 = \frac{IP_{error}}{IP_{in} + IP_{out}} \times 100 \quad (3.3)$$

$$PM_7 = \frac{UDP_{error}}{UDP_{in} + UDP_{out}} \times 100 \quad (3.4)$$

For T_{10} - T_{11}, the preliminary metrics are calculated based on equation 3.5, while for T_{12} is based on equation 3.6. In both cases, special sensitivity factors α, β, and γ are introduced. These factors differ on value for each of these tests, and thus allow for better adaptation of the output preliminary metrics accordingly. In particular, these factors refine the shape of the exponential curves (*see* equation 3.5-3.6), and their values were deduced through simulation, aiming to deliver the precise preliminary metric for each memory, CPU, and battery reading, respectively. For example, the equation in 3.6 in relation to the default sensitivity factors in Table 3.16, produce a low preliminary metric for battery readings above 60% of battery reserves, while for readings below 20% they produce a high preliminary metric. Although the default values of the sensitivity factors might not constitute the most accurate figures, they were shown to produce close proximity preliminary metrics.

$$f(x) = 100 \times e^{-\alpha \cdot \left(\frac{100-x}{100}\right)^{\beta} + \gamma} \quad (3.5)$$

$$f(x) = 100 \times e^{-\alpha \cdot \left(\frac{x}{100}\right)^{\beta} + \gamma} \quad (3.6)$$

Table 3.16: Sensitivity factors for each monitoring test

	α	β	γ
CPU utilisation	3	2	-4
Memory usage	3	2.5	-4
Battery level	6	2	0

An additional preliminary metric (T_m) is used to represent a device's mobility patterns, and is only used for clustering formation. Mobility results are assumed to be directly fed into this metric from MOBIC (Basu, P., et. al., 2001). Even though mobility is the most important

factor for cluster-head selection, other factors including processing power, utilisation status, and so on, which are not considered in (Basu, P., et. al., 2001) should also influence this decision. This is a direct result of a cluster-head being responsible for intra-cluster routing and location management. For this purpose, a cluster-head objective (O_{CH}) is defined, however distinguishes itself from the other, as it is based on a best effort approach. In this fashion, a node with the lowest cluster-head objective, that is, the lowest node-ID, it is elected as the cluster-head.

Once the preliminary metrics for all tests have been calculated, the system applies a distinct weighting (W) to each one of them according to various objectives and calculates an overall metric for each of the objectives. The mathematical expression used to calculate the overall metric (OM) is:

$$\mathrm{OM} = \frac{\sum_{n=1}^{n=12}(W_n \times PM_n)}{\sum_{n=1}^{n=12}(W_n)} \tag{3.7}$$

The system supports various objectives, including:

- **Energy efficient traffic** (**O_1**). This type of traffic typically favours the use of devices, which have high battery levels, or do not require batteries to operate.
- **Synchronous traffic** (**O_2**). This type of traffic typically requires a wide bandwidth and has high buffering requirements.
- **Asynchronous traffic** (**O_3**). This type of traffic typically has no special requirements.
- **Critical traffic** (**O_4**). This type of traffic may contain critical information that requires reliable transmission to the destination.
- **Secure traffic** (**O_5**). This type of traffic typically requires authentication, encryption/decryption, and thus requires typically good processing capacity.
- **Burst traffic** (**O_6**). This type of traffic has high buffering requirements.

In addition to these, the routing protocol can dynamically adapt to newly defined objectives, as long as their specification concerning their weighting requirements is provided. In particular, trusted reconfiguration agents can extend the network by carrying updates, such as these in their payloads. This can be advantageous in situations where an ad-hoc network requires extra configuration and finer tuning.

The weighting system is different for each of these objectives, and thus each of these scenarios is treated differently according to requirements. In this way, more weighting can be applied to battery preliminary metric for energy efficient traffic, while less can be applied to asynchronous traffic. The values of the weighting system were deduced through experimental work, and were shown to produce the desired outcome through simulations (*see* Section 5.4). Although the values of the weighting system for each test and for each objective, which are presented in Table 3.17, may not be the most accurate, they were shown to provide a good configuration.

Table 3.17: The weighting system for each predefined objective

	Energy	Synch	Asynch	Critical	Secure	Burst	Clustering
Bubble sort	1	1	0	1	5	1	5
CPU merge	1	1	0	1	5	1	5
Memory test 1 File	1	5	0	1	1	10	10
Memory test 1KB	1	5	0	1	1	10	10
Client-server throughput	3	3	1	15	1	1	10
Proxy throughput	3	3	1	15	1	1	10
TCP error	3	3	1	15	1	1	10
IP error	5	10	0	2	1	1	15
UDP error	5	25	0	2	1	1	15
CPU utilisation	25	20	1	10	3	10	25
Memory utilisation	34	20	3	12	4	12	20
Battery level	50	40	4	20	3	10	35
Mobility	N/A	N/A	N/A	N/A	N/A	N/A	50

Final route metric calculation

The final stage includes the translation of the devices' overall metric along a source route, to a meaningful expression, which indicates the ability of the route to accomplish the objective in question. For this purpose, five grades were defined including:

- **Excellent**. A device assigned this grade is the most suitable device to accomplish the particular objective.
- **Very Good**. The device can accomplish the objective very efficiently.
- **Good**. The objective will be accomplished adequately.
- **Average**. A boundary performance is expected.
- **Poor**. The device should only be used for this type of objective, only if there are no alternatives.

Figure 3.10 illustrates the process of assigning an overall routing metric to an ad-hoc routing device based on a number of pre-defined objectives, as well as calculating the final metric of the route on which the device is situated. Initially, when a source node receives the preliminary metrics of each node along a source route, which is achieved by initiating a route discovery process, it calculates the capability/incapability determination of each node along the source route in relation to the intended routing objective, and based on this information it calculates the capability/incapability of the source route. Table 3.18 provides a look-up table on which a node is based to determine the capability/incapability of each node along a source route, as it defines the desired overall metric ranges, in which a device must fall to be determined as capable. These values were deduced through experimentation, and were verified through simulations (*see* Section 5.4).

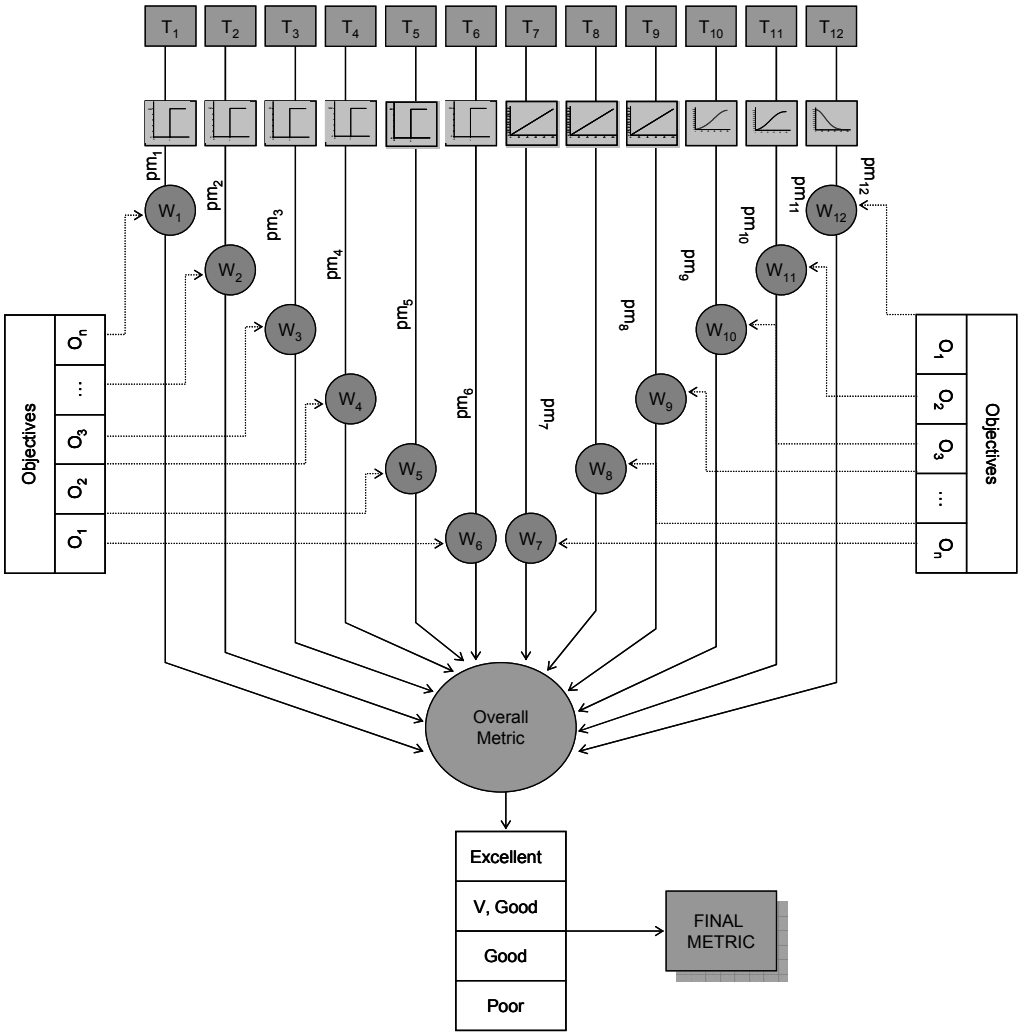

Figure 3.10: The process of calculating the overall routing metric for certain device types and objectives

Table 3.18: Desired ranges for each predefined objective

Objective	Desired metric ranges
Energy efficient network traffic	0-30
Synchronous network traffic	0-25
Asynchronous network traffic	0-50
Critical network traffic	0-20
Secure network traffic	0-15
Burst network traffic	0-20

Table 3.19 is used as a look-up table by a source node in order to translate the capable retrieved routes into a final metric. The average (AV) of the devices' overall routing metrics, as well as the standard deviation (SD) of these metrics, is used to determine the QoS-level that a route can provide. The average represents the routing fitness of the route, while the standard deviation represents the difference in routing fitness that the nodes along a route may have. Thus, a route with low average and standard deviation is likely to provide high-levels of QoS, and is thus preferable for routing objectives that impose high requirements. As an example, for asynchronous network traffic, in order for a route to be determined as *Excellent*, the average and standard deviation must be below, or, equal to 10, whereas, for synchronous network traffic, the average must be below, or, equal to five, and the standard deviation must be equal to zero.

Table 3.19: Final route metric look-up table

AV & SD	Energy	Synch	Asynch	Critical	Secure	Burst
AV≤3 & SD=0	Excellent	Excellent	Excellent	Excellent	Excellent	Excellent
AV≤5 & SD=0	Excellent	Excellent	Excellent	Excellent	V.Good	Excellent
AV≤5 & SD≤4	Excellent	V.Good	Excellent	Excellent	V.Good	V.Good
AV≤5 & SD≤5	Excellent	V.Good	Excellent	Excellent	Good	Good
AV≤10 & SD≤4	Excellent	V.Good	Excellent	V.Good	Good	V.Good
AV≤10 & SD≤5	Excellent	V.Good	Excellent	V.Good	Good	Good
AV≤10 & SD≤8	V.Good	Good	Excellent	V.Good	Good	Good
AV≤10 & SD≤10	V.Good	Good	Excellent	V.Good	Poor	Poor
AV≤15 & SD≤9	V.Good	Good	Good	Good	Poor	Good
AV≤15 & SD≤10	V.Good	Good	Good	Good	Poor	Poor
AV≤15 & SD≤15	V.Good	Poor	Good	Good	Poor	Poor
AV≤20 & SD≤15	Good	Poor	Good	Poor	Poor	Poor
AV≤30 & SD≤20	Poor	Poor	Good	Poor	Poor	Poor
AV≥30 & SD≥0	Poor	Poor	Poor	Poor	Poor	Poor

An example is provided in Section 5.6 that extensively demonstrates the process of calculating the device's preliminary metrics, the device's overall metric, the route's

capability/incapability determination, and the route's final metric, for each of the predefined objectives and in relation to various discovered routes.

3.14 Chapter Summary

This chapter introduced the detailed MARIAN routing protocol specification, which was specifically designed for multi-hop wireless ad-hoc networks, and for resource-constrained devices, such as PDAs. MARIAN is a hybrid routing protocol that utilises a clustering structure for the purpose of minimising the flooding traffic. It combines an on-demand route discovery based on stationary agents, and a proactive approach based on mobile agents. The reactive route discovery process is similar to CBRP, where a node can reactively discover a route to a destination by means of flooding, however the propagation is significantly reduced in a way that the packets are traversed only through key nodes, such as cluster-heads, gateways, and distributed gateways. The proactive route discovery process is initiated only from cluster-head nodes, with purpose of gathering the network's topology of non-adjacent cluster-heads. The novelties this protocol is aiming to achieve is to provide redundancy by discovering multiple routes, support QoS by assigning a capability/incapability routing-metric, reduce latency by allowing each cluster-head to maintain a routing table provided by the proactive approach, support reconfigurability, and provide the framework for enhanced security. Efforts are now concentrated on submitting the specification to the Internet Engineering Task Force (IETF) for Multi-hop Ad-hoc Networks (MANETs).

4 Implementation

4.1 Introduction

This chapter presents specific details on the implementation decisions involved in configuration and execution of the experimentation work that has been performed for this research. In addition, certain obstacles which this work came across are also presented and discussed, along with the way that they were overcome. Each section presents the experimentation setup for each experimentation cycle, together with details on the organisation of participating devices, whereas the hardware and software used is presented in Appendix A.

4.2 Preliminary experimentation - Implementation decisions

Experiments can be grouped into three categories: the protocol stack; the fixed network; and the wireless network. Experiments in the first category only used a single device, such as a PDA or a laptop. The fixed network experiments required two or more devices connected to an Ethernet network (10/100 Mbps). Similarly, the wireless network experiments required two or more devices, however, connected wirelessly using the IEEE 802.11b standard. Each of these experiments can be sub-divided into experiments that only use the TCP transmitter/receiver agents (*see* Appendix A) and experiments that, in addition, use the TCP proxy throughput agent (*see* Appendix A). Thus, in the absence of the TCP proxy throughput agent, the devices were operating in client-server mode, while, in its presence, the intermediate device acts as a gateway linking the other two devices.

The transmitting and receiving buffer size was set to 8KB for every experiment mentioned above. Also, throughout the experimentation, the memory and battery monitoring agent (*see* Appendix A) was used in order to record battery discharge rates experienced by the PDA. Specifically, the agent was recording the time duration between two consecutive battery levels, until the experiment was finished or the battery reserves were exhausted. However, the minimum allowed battery level was set to 15%, and the PDA was fully charged before the start of the experiments.

4.2.1 The Protocol Stack

Two experiments were conducted for each device type (*see* Appendix A): the client-server and the client-proxy-server. In the first experiment, the transmitter and receiver were running as separate processes in the same device, where the transmitter was sending data traffic from the application layer down to the protocol stack and immediately back up to the application. In the second experiment, an additional process was involved, the proxy. The difference is that the data was travelling through a proxy process before being pushed back up to the application layer. Specifically the proxy accepts data from a transmitter process on a predefined TCP port, and then pushes data down to the protocol stack and back up again, where the data was finally delivered to the server process on another predefined TCP port (*see* Appendix A). In both cases, the data was not intended to reach the physical network.

For the client-server experiment the buffer size was set to 8KB, and 4,750 buffers were routed through by the workstation, laptop, and PDA devices. The experiment was iterated a total of 20 times. In the client-server-proxy experiment, the buffer size remained the same, while the number of buffers reduced to 512 for the workstation and laptop, whereas, for the PDA this figure was considerably reduced to 85. This is due to the PDA requiring an unrealistic amount of time to conduct the proxy experiment with large amounts of data.

4.2.2 The Ethernet evaluation

This cycle of experiments was conducted only by the workstation and laptop (*see* Appendix A). In the first class of experiments, the laptop was sending data to another laptop, and the throughput achieved by the pair was measured, over 20 iterations. Following the same procedure, the throughput of a pair of workstations was also measured. In the second class, two workstations were used, which were acting in a client-server mode, while the network traffic was being routed through a workstation in the first experiment and through a laptop in the second.

In the client-server experiment, 1,024 buffers were transmitted and received by the workstation, while 512 buffers were used for the laptop. In the client-server-proxy experiment, 128 buffers were routed through by the workstation, while 64 buffers where used for the laptop. Each experiment was conducted 20 times in order to guarantee accuracy of results, and the buffer size was set to 8KB. The workstation was tested with double the amount of data that was used for the laptop, as the workstation was shown to achieve almost half more throughput than the laptop.

4.2.3 The Wireless evaluation

This cycle of experiments aims to benchmark the throughput of wireless devices, operating in client-server mode, as well as operating as proxies (*see* Section 5.2 and Appendix A). The wireless protocol used, in all cases, was the IEEE 802.11b, which can theoretically provide a maximum throughput of 11Mbits/s. In the client-server category, three device pairs where used: a pair of workstations, laptops, and PDAs (*see* Appendix A). In the client-proxy-server experiment, a pair of workstations was used, which was transmitting network traffic to each other, initially, through the workstation, then through the laptop, and finally through the PDA.

In the client-server experiment, 6,128 buffers where transmitted and received by the workstations, laptops, and PDAs pairs. In the client-proxy-server experiment, 256 buffers were routed through the workstation, while 128 buffers routed through the laptop, and only eight buffers were routed through the PDA. The number of buffers routed through the PDA was significantly reduced, as the PDA requires an unrealistic amount of time to conduct the proxy experiment with large amounts of data, whereas when in client-server mode it performs relatively well. The experiment was iterated a total of 20 times, and the buffer size was set to 8KB.

4.2.4 Obstacles presented in the preliminary experimentation phase

There were a few difficulties in this phase, mainly in the experimentation setup. In particular, most difficulties concerned the PDA's support and installation of a suitable JRE and mobile agent system, as well as the implementation and successful execution of agent-based software targeted for this platform. The operating system, usually PocketPC 2002 or 2003 (Microsoft, Corporation, 2004), restricts Java support, and consecutively, the installation of the majority of available mobile agent systems (developed in Java). Grasshopper v2.2.4 is one out of the few mobile agent systems that provides a mobile edition (ME) tailored for PocketPC platforms. Even though Grasshopper is a well documented system it provides insufficient documentation for the ME edition, resulting in major installation problems because of the uneven support of various JREs targeted for these platforms (J2ME) (*see* Appendix A). A common problem in Grasshopper's ME execution is an exception raised by most J2MEs implementations, concerning the use of an unsupported class in the Abstract Windows Toolkit (AWT). This problem was not solved, as IKV++ holds the source code private. However, an alternative solution was found. This involves switching off Grasshopper's Graphical User Interface (GUI), and thus only using the provided Textual User

Interface (TUI). In this way, the exception could be bypassed, and the agent platform can properly operate.

4.3 Proxy experimentation - Implementation Decisions

Routing in ad-hoc networks is a complex task, often requiring high throughput, CPU-intensive calculations, and large memory usage, which can result in relatively high power consumption. Resource-constrained devices, such as PDAs are the principle candidates for these networks due to their portability and small size. However, their strength of carrying out routing tasks is highly restricted because of their limited capacities.

The aim of this experimentation cycle is to benchmark the available throughput that may be offered by PDAs while routing, and determine the resource consumption rates required in terms of CPU utilisation, battery consumption, heap memory, and temperature. Experiments were conducted for various Operating Systems (OSs), Java Virtual Machines (JVMs), and buffer sizes. In particular four JVMs especially designed to target handheld devices have been tested: Insignia Jeode (Insignia, 2004); IBM J9 (IBM, 2004); NSIcom CrEme (NSICom, 2004); and Blackdown JRE 1.3 (Blackdown, 2004) (*see* Appendix A). In addition, three operating systems have also been tested including: PocketPC 2002, PocketPC 2003, and Familiar Linux (*see* Appendix A). Moreover, buffer sizes tested include: 1KB; 2KB; 4KB; 8KB; and 16KB.

Figure 4.1, presents the organisation of the hardware devices and the software used. The handheld was situated 10 meters apart from each of the two workstations, while they were situated 20 meters apart from each other. A possible limitation of the experiment is that the selected distance does not represent a real-life scenario, as 20 meters apart is a relatively short distance, as devices can communicate directly. On the other hand, the simulation of a real-life experiment requires long distances, perhaps, 200 meters apart, which is unrealistic in a laboratory environment.

The first workstation was used as a transmitter, the second as a receiver, and the handheld as a proxy. The TCP transmitter agent has been instructed to send fixed amounts of TCP network traffic, at fixed time intervals, to the handheld's proxy throughput agent, which, in turn, was forwarding it to the TCP receiver agent (*see* Appendix A). All communications were point-to-point and were using wireless as their communication medium.

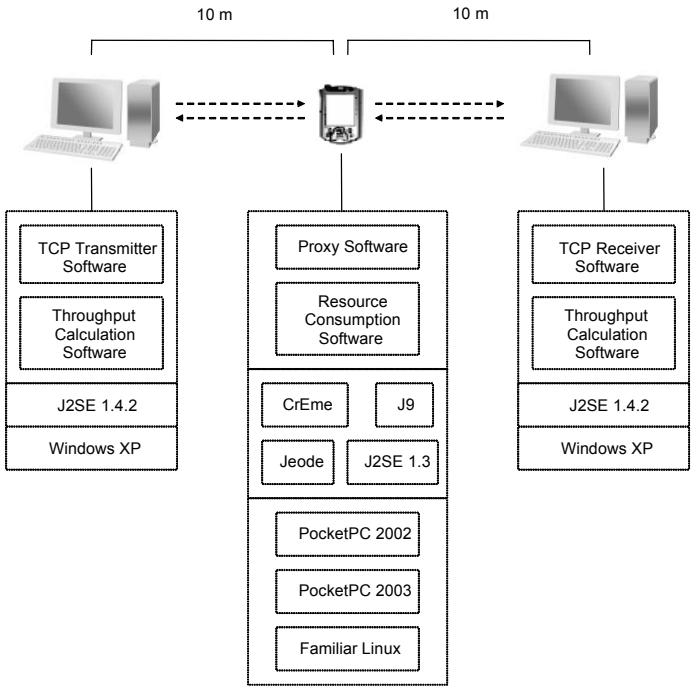

Figure 4.1: The organisation of devices for the proxy experimentation cycle

This experimentation cycle is very similar, in nature, to the proxy experiments conducted in the preliminary experimentation phase, with the only difference being that throughout, in this occasion, was measured using various JVMs and OSs combinations in order to identify the differences, and, possibly, the underlying reasons for these differences. Furthermore, three supplementary monitoring agents were used, in addition to the battery discharge agent, in order to monitor the proxy's resource consumptions in terms of CPU utilisation, heap memory usage, and temperature variation, which was implemented for Familiar Linux devices (*see* Appendix A). The tested JVMs and OSs combinations include the following:

- PocketPC 2002 with Jeode, J9, and CrEme.
- PocketPC 2003 with J9 and CrEme.
- Familiar Linux with JRE 1.3 and J9.

All other combinations were not possible, as the available JVMs are not supported by all OSs, that is, JRE 1.3 cannot be installed to PocketPC 2002/2003, nor, can Jeode be installed in PocketPC 2003 or Familiar Linux. The purpose of experimenting with different OSs and JVMs is to investigate if there are any significant differences in the routing performance,

which may be associated with the OS and/or the JVM, and identify the issues involved, which can further assist in building-up a robust model.

4.3.1 Proxy PDA with PocketPC 2002 - Setup

As previously mentioned, this experimentation cycle involved three different JVMs designed specifically for PDAs running PocketPC 2002. The tested JVMs included the Insignia Jeode, IBM J9, and NSIcom CrEme, which are all fully certified from Sun Microsystems and conform to J2ME Personal Profile specification (Sun, Microsystems, 2004c).

In the first experiment, the source transmitted 128 buffers of size 1KB each, a total of 128KB to the proxy, which forwarded them to the destination. Once the data was received by the destination, the throughput was calculated in both the transmitter and receiver, using the time taken for the data to arrive at the destination through the proxy. This experiment has been repeated fifteen times and the average throughput has been calculated, a total of 1.8311MB were routed through the proxy device. The multiple repetitions were necessary in order to measure the average throughput.

Another four experiments were conducted using the same principles, however, the buffer's size, and the number of buffers were altered in such a way that the total amount of data remained constant. The aim was to investigate whether buffer size has a significant role in routing performance, or not. In more detail, in the second experiment the source transmitted 64 buffers of size 2KB each, in the third 32 buffers of size 4KB each, in the fourth 16 buffers of size 8KB each, and in the fifth 8 buffers of size 16KB each, thus a total of 128KB for each experiment. In this way, at each experiment, the total amount of data routed through the device remained the same, which is necessary in order to measure the effect of the buffer size.

The experiments described above were performed for Jeode, J9, and CrEme. Finally, in addition to measurements concerning the throughput achieved by the proxy device, the battery consumption rate, CPU utilisation, and heap memory usage (see Appendix A) have been also monitored throughout this experimentation cycle.

4.3.2 Proxy PDA with PocketPC 2003 - Setup

The same series of experiments were conducted here as described in Section 4.3.1, with the only difference being the OS used, which was PocketPC 2003, a more recent version of Microsoft's OS than PocketPC 2002. The hardware, software, and organisation of devices remained the same. At present, Jeode is not available for PocketPC 2003, and thus the experimentation had to be restricted to J9 and CrEme only. The purpose of this series of

experiments is to investigate whether the OS can significantly influence the performance of a proxy-PDA in an ad-hoc environment.

Similarly to Section 4.3.1, the source device was used to transmit a fixed number of data to the proxy-PDA, which, in turn, was used to forward them to the destination device. All communications were point-to-point and used wireless as their communication medium. The same buffer sizes were tested against available throughput, including: 1KB; 2KB; 4KB; 8KB; and 16KB. A total of 128KB were thus routed for all buffer sizes. In addition, this experiment has been repeated fifteen times for each buffer size and the average throughput has been calculated, with a total of 1.8311MB for each experiment. Moreover, the throughput, battery consumption rate, CPU utilisation, and heap memory usage were monitored throughout this experimentation cycle.

4.3.3 Proxy PDA with Familiar Linux v0.7.2 - Setup

The purpose of these experiments is to measure the strengths and weaknesses of different JVMs when executing proxy software on Familiar Linux handheld devices. They are designed in the same way as described in Section 4.3.1, with the only difference being the OS installed. A non-commercial OS targeted for handheld devices, such as PDAs, is Familiar Linux. It can be freely downloaded and installed in any of the supporting handhelds by replacing the original OS, or creating a dual-band booting. For the purpose of this research, two JVMs were selected due to their different nature: Blackdown's Java 1.3; and IBM's J9 Personal profile for Zaurus. Both JREs are fully certified by Sun Microsystems, however Blackdown 1.3 conforms to Sun's Java 1.3 specification, while J9 conforms to Sun's J2ME Personal Profile specification. Thus, the full Java 1.3 onto PDAs possibly improves the development on these platforms, but, on the other hand, it imposes a heavy burden due to its large footprint. On the contrary, J9 Personal profile for Zaurus is a light-weighted JRE, which is restricted to a smaller scale of classes and libraries that comply with the Sun's J2ME specification (*see* Appendix A).

Blackdown Java 1.3 has been especially designed for Familiar Linux, and it is thus not supported by PocketPC platforms. IBM J9 Personal Profile has been designed for Zaurus handhelds running Embedded Linux, that is, a small-footprint Linux OS targeted for Zaurus handhelds, and therefore there is no official support for any other handhelds, or platforms. However, experimentation demonstrated that IBM J9 Personal Profile can be successfully ported on other handheld devices, such as iPAQs, which are running Familiar Linux. Similarly to Section 4.3.1, the throughput, battery consumption rate, CPU utilisation and heap memory usage, as well as temperature variation were monitored throughout this series of ex-

periments. These additional measurements may provide evidence on whether proxy devices suffer from significant changes in their internal temperatures, and if so, to which degree, and whether the variation in temperature significantly varies between different JVMs. The temperature variation test was specifically designed for Familiar Linux, as the supported JVMs, that is, Java 1.3 and IMB J9 for Zaurus, have major differences in the resources they utilise while routing, and may thus provide a suitable environment for temperature monitoring.

4.3.4 Obstacles presented in the proxy experimentation phase

This experimentation phase was inspired by efforts aiming in the optimisation of the proxy-based PDA. In particular, the assumption was that the OS or/and the JVM may be two of many factors that could significantly improve the throughput provided by the proxy. In order to test this assumption, a variety of OSs and JVMs, especially designed for PDAs, had to be found, installed, and tested. There were no obstacles involved in this experimentation phase, as there are notes available for PocketPC and Familiar Linux platforms, as well as for each tested JVM.

4.4 BASS Experimentation – Implementation decisions

BASS is a multi-agent benchmarking system, which aims to determine the routing fitness of ad-hoc routing devices, and was especially designed to be light-weighted, and thus allow its execution on resource-constrained devices. Its purpose is to conduct various tests, such as intensive algorithmic calculations, routing throughput, utilisation monitoring, and pass the results to a metric-driven routing protocol, which can then translate them to metrics that describe the fitness of a device to perform routing tasks (*see* Section 3.10).

This section presents various tests that were conducted in order to identify the most appropriate ones, which can be used to describe the routing fitness of an ad-hoc device. The tests that are directly related to ad-hoc routing are: CPU-intensive sorting algorithms, memory tests (applicable to PDAs), client-server and proxy throughput, network protocol error, CPU utilisation, memory usage, and battery reserves monitoring (*see* Section 3.10). Thus, these tests can be used to determine the routing fitness of various ad-hoc devices, as these factors are directly linked to ad-hoc routing (*see* Section 3.10). In addition, a few other tests, that are not directly related to ad-hoc routing, and thus are not involved in the metric calculation process, are presented in this section. These include: the group-level test; the Internet connectivity test; and the Java threads utilisation test. The group-level test is used to gather the devices' system-level information, such as the OS and JVM version, which can be used to

group similar devices, and thus allow the dispatch of routing updates to specific groups, if necessary, which can extend the reconfiguration flexibility of the ad-hoc network. The Internet connectivity test was designed to provide insight information on the connection and download speeds of resource-constrained devices. The Java threads utilisation test was designed to gather statistical utilisation information for each Java *Thread* running in the device, and could possibly used to detect denial of service (DoS) attacks, which may be launched by malicious agents.

The battery monitoring agent was used to monitor the devices' battery consumption, while the devices were executing the tests. Even though BASS is equipped with more resource-consumption monitoring agents, such as the CPU utilisation, the heap memory usage, and the temperature variation, their operation was skipped, as they were used in the proxy experimentation phase (*see* Section 4.3). Thus, the group-level, bubble sort, CPU merge, memory test, Internet-connectivity, error packets monitoring, Java threads, and battery reserves monitoring agents were executed on each of the hardware devices (*see* Appendix A). The following sections present specific details on their setup and execution.

The group-level agent does not require any addition configuration, or supervision, on runtime, and were executed only once on the laptop device (*see* Appendix A). The details of this agent are shown in Appendix A.

4.4.1 Bubble sort agent – Setup

In this experiment, the bubble sort test was requested to sort 30,000 random integers. Various levels of depth were selected, including the 1D, 2D, 3D, and 4D. Each test was required to sort approximately the same amount of integer values, i.e. 30,000 for the 1D, 173×173=29929 for the 2D, 31×31×31=29791 for the 3D, and 13×13×13×13=28,561 for the 4D. The purpose of experimenting with in-depth dimensions rather than a single one is to identify the CPU's response on increased algorithmic depth, and compare the results to the effect that ad-hoc routing has on the CPU utilisation. Each test was executed on each device, for 20 iterations, and the battery discharge rate was recorded throughout the sorting process.

4.4.2 Memory test - Setup

This agent has the capacity of performing two similar tests: creation of varying number of files with constant file-sizes and creation of a constant number of files with varying file-sizes. For each test two experiments were carried out. Specifically, for the first test and in the first experiment one file was used, while in the second the file number increased to 16. Their sizes

were ranging from 0 (KB) to 2000 (KB), in increments of 100 (KB). For the second test and in the first experiment the size of the files used was 1 (KB), while in the second it was set to 16 (KB). The number of files was ranging from 100 to 2000, in increments of 100 files. Each experiment was iterated 20 times, and the battery discharge rate was recorded throughout these tests.

4.4.3 CPU merge agent – Setup

This test is an alternative to bubble sort, designed to impose a lighter CPU utilisation, and thus provide a simpler and faster execution for limited devices, such as mobile phones. Even though both tests can be requested to short the same number of random integer values, it is guaranteed that the CPU merge agent will always finish first, as it bases its functionality to a simpler sorting algorithm than the one used by the bubble sort agent. In addition, this test can be used in situations were the user wishes to perform a quick benchmarking of the handheld's processing power, and thus avoid long-lasting processes. Thus, this test is a preferable alternative to the bubble sort in cases where a quick benchmarking of a device's processing speed is necessary, or, the device is incapable of conducting the 1D bubble sort. In the actual experiment, the agent was requested to sort 80,000 random integers, and the experiment was iterated 20 times.

4.4.4 Internet connectivity agent – Setup

This test attempts to download a single HTML page of 26,823 (bytes). It can be used to determine whether there is an Internet connection or not, assuming that the web-site address is always available. In order to minimise the possibility of the site being actually unavailable, multiple addresses may be passed to this agent at runtime. If the agent senses that there is Internet connection it calculates the following:

- Time taken to connect.
- Time taken to download.
- Total time taken.

In this particular experiment, all devices were configured in such a way so as to have a connection to the Internet and attempted to download a single web-site from the same source. The connection was provided wirelessly from an access point. Although this test is not related to ad-hoc routing, and is thus not involved in the metric calculation process, it might

provide insight information on the connection, and download, which is required by resource-constrained devices.

4.4.5 Error packets monitoring test – Setup

The purpose of this test is to monitor the network traffic generated at an ad-hoc routing device, and to gather statistical information on the amount of traffic being processed by the device, as well as to capture any network errors that may have occurred. Thus, this test can constantly monitor the incoming, outgoing, and error data packets, of each of the supported network protocols, that is, IP, TCP, and UDP, and thus allow the ad-hoc routing protocol to determine the device's network state. This information can then be used by the ad-hoc routing protocol to determine the device's reliability, and thus base its routing decisions accordingly. This test was executed a few times for a relatively short period of time on the laptop device. The instance, in which network errors occurred, was captured, and presented in Appendix A.

4.4.6 CPU utilisation and memory usage monitoring test – Setup

As previously mentioned, the purpose of this test is to monitor the utilisation of a device in terms of CPU and memory. In general, the test can greatly assist in the determination of the routing capabilities of a device. Unlike most other tests, the results can be highly dynamic, as they can widely vary, depending on the current state of the device. Although a device may have achieved good results in all previously described tests, if results from this test prove to be inadequate, then the device is most likely to be determined as poor in terms of routing capabilities. Therefore this test is extremely sensitive in relation to the overall fitness calculation, as CPU and memory are usually the two most important factors.

This test was used throughout the proxy experimentation phase, and results were mainly concentrated on the current CPU utilisation of the proxy-enabled devices. In addition to CPU utilisation, this test can provide information, including: the total memory; the available memory; the virtual memory used; and so on. A snapshot of the test's output is presented in Appendix A.

4.4.7 Java threads monitoring test – Setup

The purpose of this test is to monitor all Java *Threads* in the system, and store information including the following:

- **threadID**. A unique ID identifying this running thread.

- **cpuPercentUsed**. The CPU percentage used by this thread.
- **cpuTimeUsed**. The CPU time used by this thread.
- **lastUpdated**. Information corresponds to the last updated time.

Since the BASS system, the Grasshopper agent-platform, and the stationary and mobile agents themselves are all written in Java, this test could be especially useful in determining their overall CPU utilisation, and memory usage, activities. Thus, it provides the basic infrastructure for tolerating denial of service attacks, originating from foreign mobile agents, where stationary guard agents can monitor their activities, and kill or depart them in case of any suspicion. However, this test is not directly related to ad-hoc routing, and thus it does not participate in the metric calculation process.

4.4.8 Obstacles presented in the BASS experimentation phase

The main difficulty of this experimentation phase was to determine the suitability of each test in terms of ad-hoc routing, and thus rely only on the tests that are directly related to determine the routing fitness of various ad-hoc devices. As previously discussed, tests, such as the group-level, Java threads utilisation, Internet connectivity, and heap memory usage were left out from the metric calculation process, due to their lack of relevance to ad-hoc routing. On the other hand, the group-level test provides system-level information for each participating device, which may be used by the routing protocol to disseminate routing updates to the group of devices, which need them, for example, all PocketPC platforms. In the same manner, the heap memory usage test was used throughout the proxy experimentation phase in order to measure heap memory, which was used by proxy-enabled devices. However, the Java threads utilisation and Internet connectivity tests have little or, no validity, in the context of this research, and, thus, they were presented here, as added features that could only provide interesting results from the perspective of each device type.

4.5 Experimentation of mobile agent migration – Implementation decisions

This experimentation cycle aims to determine the migration times involved with mobile agents and their ability to reduce network overhead. Initially, two groups of devices with different hardware characteristics were selected, namely, the superiors and inferiors (*see* Appendix A). Each group consisted of five workstations, which were wirelessly connected using the IEEE 802.11b standard. A mobile agent was implemented in such a way so as to

migrate from its home platform, hop to each device of the same group, in sequence, and return back to its origin, along the same route, which was used by the agent for its propagation. Thus, the total migrations were designed to be of exactly eight hops. In addition, the agent was equipped with a timer, used to measure its round-trip time (RTT), and the migration time, along each hop. This experiment could provide insight information in determining the average migration time of a mobile agent, and further provide evidence on whether devices' hardware characteristics can, or cannot, affect migration times, in general.

The last phase of this experimentation cycle aims to show whether data gathering mobile agents, with intelligent filtering capabilities, could, or could not, improve upon static approaches. For this purpose, a database application scenario was implemented in such a way so as to enable both static and mobile agent retrieval approaches to a fair comparison. In particular, a database holding information about research articles was maintained by a laptop, while another nearby laptop was maintaining a region registry. A remote PDA required specific information from the database, and was situated beyond the database's reachability. An intermediate proxy-PDA device was used to facilitate forwarding services for both the static and mobile agent approaches. Initially, the time taken to retrieve the requested data using the pure static agent approach was compared to the pure mobile agent approach, whereas in the second phase the comparison was made between the pure static agent approach and the mobile agent approach which was equipped with intelligent data filtering.

Figure 4.2 presents the organisation of devices used to conduct the experimentation of the database application scenario. As shown, every device is in direct communication range with each other, apart from the client PDA and the database laptop (nodes A and D), which belong to different wireless domains (WD_B and WD_A respectively). The application scenario required the registration of each agency, agent, and service, in the local region registry. Thus, an agent could locate other objects, in the distributed environment, by simply requesting their location information from the local region registry.

According to the static agent approach (*see* Figure 4.3), the client agent initially contacts the region registry and requests the contact information of the database agent. The region registry provides the information to the client agent, which then tries to contact the database agent directly by passing its user's query. However, since the client PDA and the database laptop are not in communication range, this attempt is designed to fail. Thus, the client senses the failure and decides to contact the region registry again, only that this time it requests the proxy agent's contact information. Once the details arrive at the agent, it contacts the proxy agent and passes its user's query. The proxy agent realises that the client requires a database proxy service, and thus contacts the region registry to retrieve the contact information of the database agent. Then, it passed the original query to the database agent and waits

for the agent to process the request. Once the resulting data is available, the proxy agent forwarded it to the client agent.

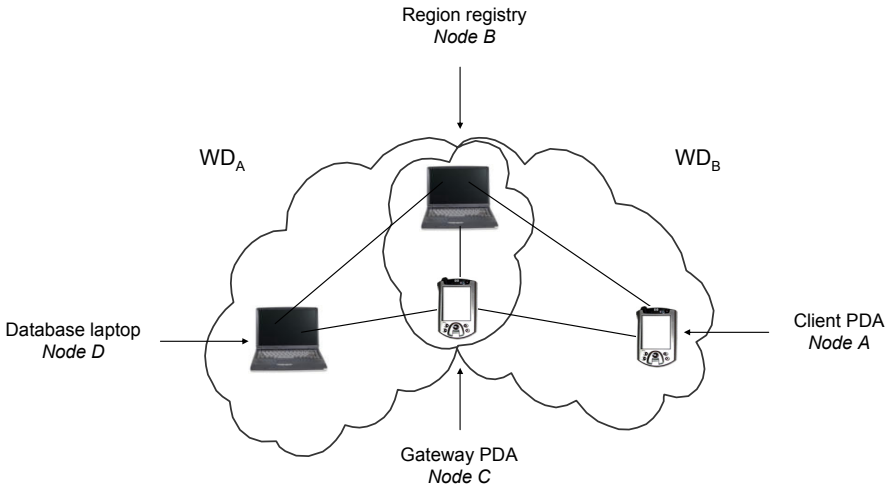

Figure 4.2: Organisation of devices for the database retrieval experiment

According to the mobile agent approach (*see* Figure 4.4), the client mobile agent initially contacts the region registry and requests the location information of the database agent, that is, the agency location information in which the database agent lives. Then, it tries to migrate to that location, however, the client PDA and the database laptop are not in communication range, and thus this attempt fails, similarly to the static agent approach. The agent senses the migration failure, and decides to migrate to the intermediate node, that is, the gateway. Thus, it contacted the region registry and retrieves the gateway's location information. Next, it initiates its self-migration with destination the gateway node. Once there, it tries to migrate to the database laptop and succeeds, as the proxy and database nodes are within communication range. The agent then senses its arrival on the database node and initiates communication with the database agent. It passes the query to the agent, and stores the results in its payload. Then, it inverted its itinerary and migrates to the gateway node, and, finally, to its home-platform, that is, the client node.

The filtering mobile agent approach follows the same principles, as the pure mobile agent data gathering, however, in this case the client mobile agent maintains preference information, on articles that its user requires. Once the client mobile agent retrieves the results from the database, instead of just storing them to its payload, it first filters the data locally, according to its user's preference criteria, and thus stores only a small amount of the total data. For instance, according to the current implementation of this application scenario, the agent

bases its decision onto knowledge acquired by the user upon its creation. Specifically, the agent learned that its user was only interested in recent articles published in journals only, written by a precise set of authors, and supplied in a *pdf* format. The agent interprets the word *recent* to papers written between the years 2004 and 2005, and follows its user's instructions.

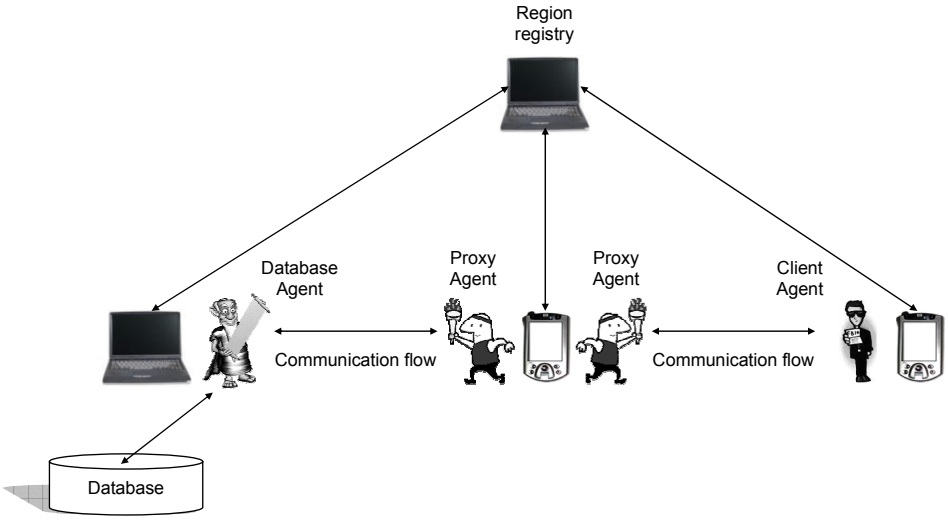

Figure 4.3: Retrieving database records using the static agent approach

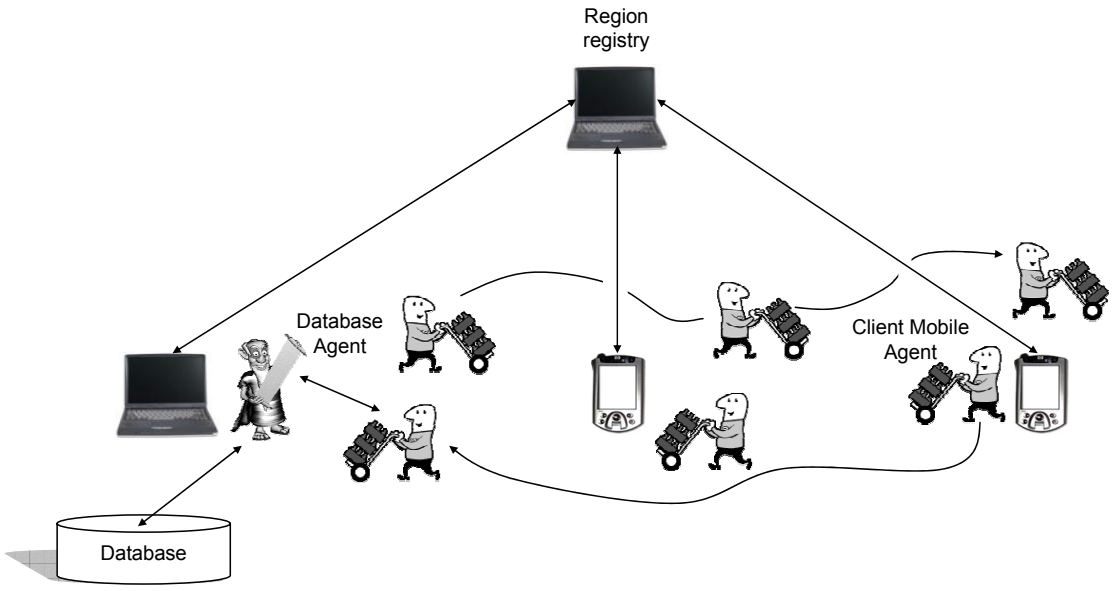

Figure 4.4: Retrieving database records using the mobile agent approach

4.5.1 Data gathering using the static agent, mobile agent, and mobile agent with filtering approaches - Setup

Initially, the results from the database were set to be 100Kbits, which then increased to 200Kbits, and, finally, to 300Kbits. The static and mobile agent approaches were tested against these amounts, and, in respect to time. Each experiment was iterated 20 times in order to allow accurate measurements. Then, the results from the database query were set to be 15Mbits, where the static agent and mobile agent with filtering approaches, were tested for this data size. Similarly, this experiment was iterated 20 times. Each device was set to be approximately 5 meters apart from the remaining. The PDAs and laptops were fully charged at the start of the experiment.

4.5.2 Obstacles presented throughout the experimentation of mobile agent migration cycle

In this experimentation cycle a few obstacles were presented mostly in relation to the possession and set-up of the hardware equipment, since only the first phase required two groups with each group consisting of five identical devices. In addition, the software installation and control was another issue, since it had to be spawn over a large number of devices, however, it was dealt efficiently. Furthermore, concerns were also raised with the definition of *intelligent filtering*. Specifically, the term *intelligent filtering* is reasonably ambiguous, and highly relates to the context used, thus, it can be implemented in many possible diverse ways. A reasonable decision was then taken, that is, to allow the mobile agent to filter the data, as if it performed a second search, on the originally returned data. The search criteria were based on its user's preferences, which were fed into the agent on creation. Even thought some database search components offer sophisticated search facilities, this is not the case for most of them, and thus it makes sense to utilise mobile agent technology as to an extension of their services.

4.6 Metrics simulation – Implementation decisions

This experimentation cycle involves simulation experiments which aim to prove the correctness, adaptability, and fast convergence of the proposed metric-driven routing protocol, in various changes of the devices' main routing elements. Specifically, conditions such as the rapid increase in a device's utilisation, sudden drop of a device's battery capacity, and rapid decrease in a device's available memory, have been tested in relation to their effect in the device's overall routing metric. For the purposes of this simulation, six distinct device types have been defined, including the following:

- **Average strength iPAQ PDA** (DT_1). These devices are the most common in mobile ad-hoc networks.
- **iPAQ PDA with high utilisation** (DT_2). These devices inherit all the characteristics of the first category, however, the utilisation was set to constantly be considerably high.
- **iPAQ PDA with good network throughput** (DT_3). These devices also inherit all the characteristics of the first category, however, the proxy throughput was set to a valid maximum for the standards of these devices.
- **iPAQ PDA with poor throughput** (DT_4). These devices provide a valid minimum proxy throughput.
- **iPAQ PDA with high errors in the network protocols** (DT_5). These devices are prone to network protocol errors.
- **iPAQ PDA with low battery** (DT_6). These devices were set to have low battery capacity.
- **Average strength laptop** (DT_7). An average strength mobile device, however, highly capable when compared to the first category devices.
- **Good strength laptop** (DT_8). A highly capable mobile device.
- **Powerful workstation** (DT_9). An exceptionally strong device, which is not battery-driven.

Each device is assigned to a set of preliminary metrics (*see* Section 5.5.2), which are deduced from the test results that are later presented throughout Sections 5.1 - 5.3. The purpose of the simulation experiments is to vary key preliminary metrics from zero to 100 for each defined device type, and measure the effect that this could have on the device's overall metric, and capability/incapability determination criteria, in relation to each predefined objective (*see* Section 3.13). This experimentation phase has not faced any problems or obstacles.

4.7 Chapter Summary – Experimentation setup

This chapter presented a thorough analysis and provided implementation details for each experiment conducted for the purposes of this research. Specifically, this chapter presented the configuration of each experiment, where the hardware and software used is presented in Appendix A. In addition, it provided details on the obstacles presented and how these were efficiently dealt. Each section provided implementation detailed concerning the setup for each experimentation cycle, together with details on the organisation of participating devices. It should be noted that this chapter did not include implementation details for the experimentation cycle presented in Section 5.6, as it was especially designed for demonstration

purposes and mainly provides a demonstration on various application scenarios of the proposed research work.

5 Results

5.1 Introduction

The purpose of this chapter is to outline the experimentation approach, and to presents the results obtained, which justify the design and verification of the model. One of the key factors is to determine the characteristics of various device types, which can be used in simulators, such as *ns-2*, and provide a methodology for automatically assessing devices for ad-hoc routing. In this way, the devices' responsibilities towards ad-hoc routing is balanced on their capacity to perform key tasks, such as buffering, processing, data routing, as well as their current status, such as battery life, CPU utilisation, and memory usage, which overall introduces fairness from the device's perspective, and most importantly improves the network's overall reliability and performance. This chapter demonstrates that routing is a resources-consuming process, which has a significant negative effect on resource-constrained devices, especially in terms of battery discharge rate and CPU utilisation. In addition, high-end devices typically provide sufficient throughput, which supports high-requirement routing scenarios, such as synchronous and burst network traffic, whereas resource-constrained devices can normally support network traffic with no special requirements, such as asynchronous chat. This chapter also shows that the introduction of the proxy, which is a simple and efficient way of routing, allows a wide range of devices to perform routing tasks, even handhelds, which are possibly the weakest ad-hoc devices. In addition, a number of optimisation strategies are shown to improve the throughput that handhelds can achieve, such as the selection of the Operating System, and Java Runtime Environment.

5.1.1 Device tests

This experimentation phase was conducted to provide answers for the following fundamental questions:

- Can proxy devices be used to route traffic over fixed and wireless networks?
- Are mobile devices fit enough to route network traffic?
- Are there any restrictions on the type of traffic PDAs can handle?

Multi-hop ad-hoc routing inherently suggests that each mobile device along a routing path should equally participate in the routing process, as needed. Thus, the aim of the experimentation is to identify the fitness of devices with various hardware characteristics,

as routing elements in terms of maximum throughput and battery discharge rate.

5.1.2 The protocol stack

This is the simplest routing experiment, where a device is sending data from the application layer down to the protocol stack and immediately back up to the application layer, however, the data does not actually reach the physical network. This test is useful in providing an estimation of the average time which is necessary for data to reach the physical network, having been processed by various protocols of the protocol stack. The implementation details of this experiment are described in Section 4.2.1.

Figure 5.1 presents the throughput of the protocol stack experiment achieved by the workstation, laptop, and PDA, over multiple iterations. In more detail, the workstation achieved the best results, with an average of approximately 400Mbits/s, while the laptop achieved an approximate average of 95Mbits/s, and finally the PDA achieved an approximate average of 1.75Mbits/s. By comparing these results, it can be deduced that the workstation performed the experiment almost four times faster than the laptop, and a considerable 230 times faster than the PDA. These results clearly indicate the difference in processing power between high and low performance devices. Further evidence of the difference in processing power between low and high performance devices is presented in Figure 5.2. The results were obtained by conducting the protocol stack experiment, using a proxy in-between the sender and receiver processes. The figures suggest that the workstation may route data internally up to 10 times faster than the PDA, however achieving better, but similarly, to the laptop. Consecutively, devices with similar hardware characteristics to a PDA may struggle when dealing with high routing requirements, such as the ones imposed by real-time traffic. In addition, a resource-constrained device, such as a PDA, could become over-utilised while routing, and may even become temporarily unavailable to perform user-driven tasks.

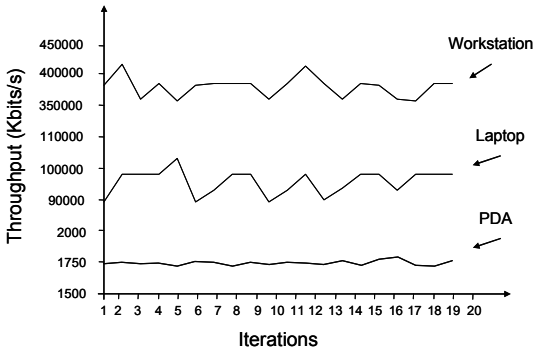

Figure 5.1: The protocol stack throughput

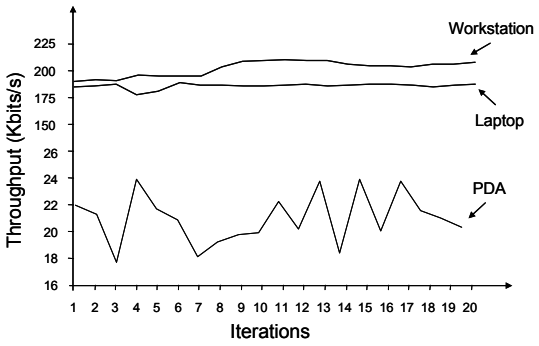

Figure 5.2: The protocol stack throughput, through a proxy

Another equally important factor to throughput, especially for mobile devices, is battery discharge rate. In other words, the average time (s) in which the battery-level drops by 1% (delta 1%), while the device is conducting an experiment. Figure 5.3 presents the battery discharge rate for the PDA, throughout the internal client-server experiment. The *x-axis* shows the time taken for the complete experiment, while *y-axis* shows the time taken for the battery to drop by 1%. These measurements are important so as to determine the resources, in terms of battery, which is required by resource-constrained devices to push data down the protocol stack and back up again. In this way, this experiment provides a valuable insight in the consumption rates experienced by a device when routing. According to Figure 5.3, the average battery discharge rate for the PDA while conducting the client-server experiment is approximately 141s, whereas the average battery discharge rate for the PDA while conducting the client-proxy-server experiment is approximately 127s (*see* Figure 5.4). Specifically, battery discharge at almost 10% faster for the client-proxy-server experiment than the client-server. Thus, that the added proxy process causes the battery to discharge at a higher rate, which infers that more intensive tasks consume the battery at a higher rate.

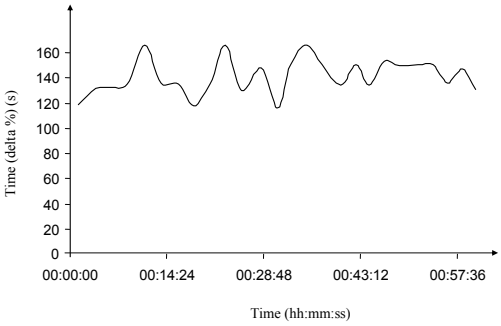

Figure 5.3: Battery discharge rate experienced by the PDA throughout the protocol stack experiment (client-server)

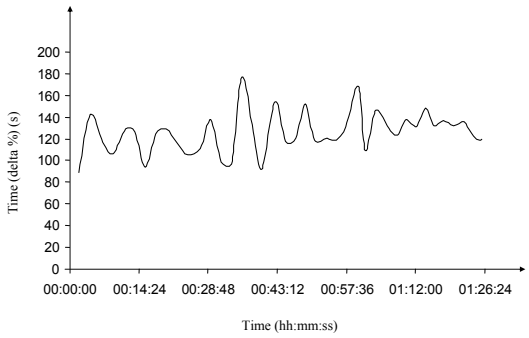

Figure 5.4: Battery discharge rate experienced by the PDA throughout the protocol stack experiment (client-proxy-server)

5.1.3 Throughput over an Ethernet 10/100 (Mbits/s)

This experiment's description and implementation details are thoroughly presented in Section 4.2.2. Figure 5.5 presents the throughput provided by the transmitting and receiving workstations and laptops. The workstation pair achieved an average throughput of approximately 9.4Mbits/s, while the laptop pair achieved approximately 4Mbits/s. In addition, the workstation results were similar for all 20 iterations, while, in case of the laptop, several inconsistencies were observed. This could be attributed to the fact that the laptop is caching the data to virtual memory, as its physical memory capacity was limited. In terms of throughput, the workstations pair achieved approximately twice as good as the laptops pair, which is attributed to the overall fitness of the workstation device, and

especially to its Ethernet interface. Figure 5.6 presents the throughput provided by the proxy-workstation and proxy-laptop. The proxy-workstation achieved an average throughput of approximately 610KBits/s, while the proxy-laptop achieved an average throughput of approximately 190KBits/s. In comparison to results presented in Figure 5.5, it can be clearly seen that the introduction of the proxy had a considerable effect on the overall throughput for both device types. Specifically, the throughput through the proxy-workstation was reduced by 93.6%, when compared to the workstation client-server experiment, while the throughput through the laptop-workstation was reduced by 95.25%, when compared to the laptop client-server experiment. In addition, the proxy-workstation routed the same amount of data, as the proxy-laptop, within almost one-third of the time. This further suggests that higher performance devices can actually increase throughput, and thus route data in less time.

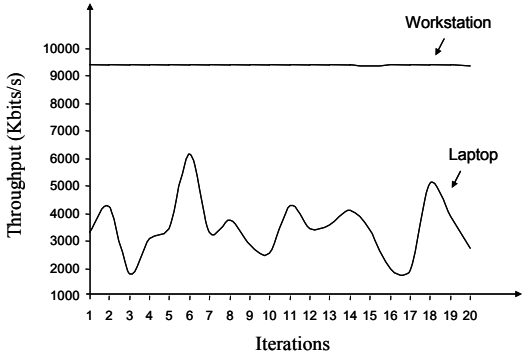

Figure 5.5: Client-Server throughput, over an Ethernet network, 10/100 (Mbits/s)

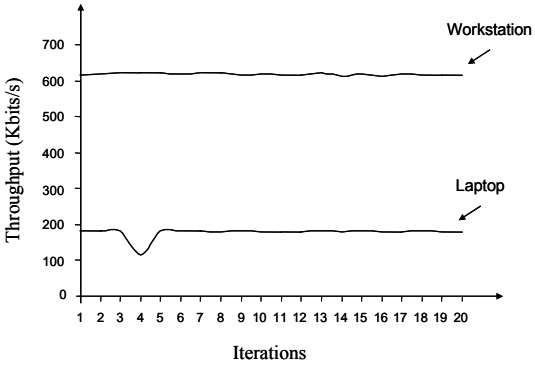

Figure 5.6: Client-Server throughput, through a proxy, over an Ethernet network 10/100Mbits/s

5.1.4 Throughput over a wireless IEEE 802.11b network (11Mbits/s)

In this category, devices were wirelessly connected (IEEE802.11b standard) and tested as presented in Section 4.2.3. Figure 5.7 presents the throughput results obtained by a pair of workstations, laptops, and PDAs operating in client-server mode, while Figure 5.8 presents throughput results when these devices were acting as proxies. According to Figure 5.7, the workstations achieved the best results, almost twice as much throughput as the laptops, and almost three times more throughput than the PDAs. Similarly, according to Figure 5.8, it can be seen that the proxy-workstation achieved the best results. In particular, the workstation achieved twice as much throughput as the laptop, and 100 times more than the PDA. The difference between the proxy-workstation and the proxy-PDA is significant, and may be addressed to the fact that PDAs are limited devices, in terms of processing power, memory capacity, buffering capabilities, and so on. Thus, it may be

safe to conclude that PDAs are not appropriate for routing heavy network traffic, such as real-time multimedia traffic with Quality of Service (QoS) requirements. Instead, these devices should only be used to route asynchronous network traffic, such as message exchange.

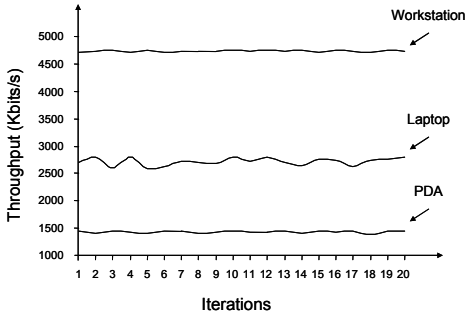

Figure 5.7: Client-Server throughput over a wireless network IEEE 802.11b (11Mbits/s)

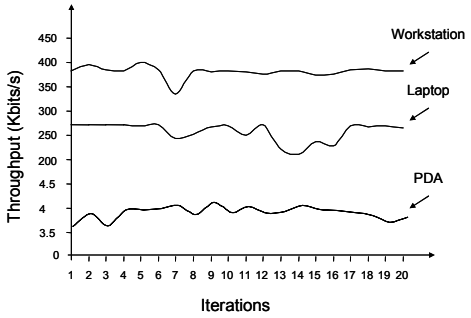

Figure 5.8: Client-Server throughput, through a proxy, over a wireless network IEEE 802.11b (11Mbits/s)

Figure 5.9 presents the battery discharge rate of the receiving PDA, while conducting the client-server experiment. As it can be seen from the graph, the average discharge rate for the receiving PDA was approximately 74s, which is a significant reduction of approximately 50%, compared to the corresponding results in the protocol stack experiment. This is possibly a direct effect of the wireless being enabled and used, which obviously requires a relatively sufficient amount of energy. Figure 5.10 presents the battery discharge rate of the receiving PDA, while conducting the client-proxy-server experiment. As it can be seen, the average battery discharge rate was approximately 70s, which is a slight difference from the results presented in Figure 5.9, specifically a reduction of approximately 5.5%. Thus, results do not prove that data routing has a significant negative effect on battery for resource-constrained devices, which may be attributed to the limitations of this small-scale experiment.

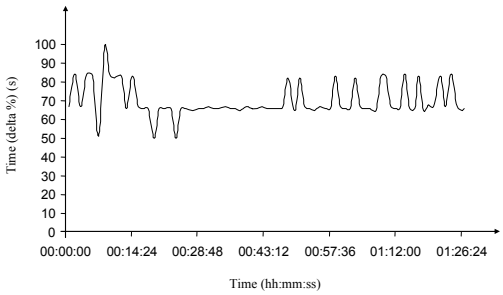

Figure 5.9: Battery discharge rate experienced by the PDA throughout the wireless experiment (client-server)

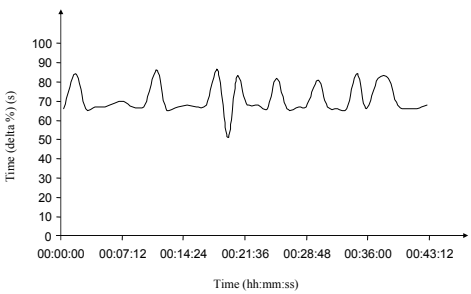

Figure 5.10: Battery discharge rate experienced by the PDA throughout the wireless experiment (client-proxy-server)

5.1.5 Outcomes and evaluation of preliminary experimentation

The introduction of the proxy device was shown to reduce network throughput, and, in addition, increase the proxy's battery discharge rate. However, the proxy allows many different types of devices, such as workstations, laptops, and handhelds to route data, whereas proper routing software would possibly prohibit its use by handhelds, as it requires the execution of intensive processes. The benefit of using a proxy is that it does not require devices to execute additional functions, other than routing data, as it only requires the creation of two *Threads*, which are used as tunnels for data exchange, and thus it does not impose additional processing.

High performance devices were shown to provide significantly more throughput than resource-constrained devices, when used as proxies. In particular, in the client-server protocol stack experiment, the workstation achieved 230 times more throughput than the PDA, while, in the client-proxy-server experiment of the same category, the difference was 100 times. In the wireless client-server experiment, a pair of workstations achieved an average throughput of approximately four times more than the corresponding pair of PDAs, however, the difference between the proxy-workstation and the proxy-PDA, was significantly higher. Specifically, the proxy-workstation accomplished an average throughput of approximately 100 times more than the proxy-PDA. Throughput results suggest that resource-constrained devices may be unable of routing heavy network traffic, because of the significantly narrow throughput provided, and may thus be more suitable of routing network traffic with no specific requirements, such as asynchronous chat.

In terms of battery discharge rate, it was shown that when a PDA has its wireless on it consumes almost 50% more battery that without wireless. As ad-hoc routing devices typically use wireless for data routing, the PDA may struggle to maintain the battery discharge rate, at a reasonable pace, if it is required to constantly have its wireless on. In addition, data routing was shown to have a slight effect on the PDA's battery discharge rate, which may be attributed to the small-scale experiment.

5.2 Proxy Experimentation

Following the significantly negative results of the proxy-PDA presented previously, this section investigates the significance of the proxy element, such as on the selected buffer size, the Operating System (OS), and the Java Virtual Machine (JVM) used to interpret the Java-based proxy bytecode. Thus, for the selected device, which is an iPAQ h5450 (*see* Appendix A) all possible OS and JVM combinations were tested. In addition, each experiment was conducted for five different buffer sizes, at: 1KB, 2KB, 4KB, 8KB, and 16KB. For each experiment, various devices' important factors were monitored, includ-

ing the battery discharge rate, CPU utilisation, heap memory usage, and temperature. The implementation details of this experimentation cycle are described in Section 4.3.

The preliminary experimentation provided several evidence of the inability of resource-constrained devices, as routing elements, especially in wireless networks. The purpose of this set of experiments is to further benchmark the minimum and maximum throughput that may be provided by a proxy-enabled PDA, and, in addition, to measure the battery consumption rate, CPU utilisation, heap memory usage, and temperature variation. The aims are to:

- Benchmark throughput offered by a proxy-based PDA, in an ad-hoc network.
- Identify the degree in which the installed OS may affect throughput.
- Identify the degree in which the installed JVM has on throughput.
- Identify the degree in which the selected buffer size has on throughput.
- Monitor the resource consumption rates of the proxy-based PDA, while routing.
- Monitor the resource consumption rates of a PDA, while in idle state.
- Benchmark the imposed overhead, in terms of resources used by the proxy-based PDA, and deduce the type of traffic which may be suitable of routing.

5.2.1 Throughput of a PDA running PocketPC 2002

The details of the experiment are presented in Section 4.3.1. Figures 5.11 - 5.14 present the throughput measurements, for each supported JVM, over 15 iterations, using buffer sizes of 1KB, 2KB, 4KB, 8KB, and 16KB. Insignia's Jeode JVM is represented by a solid line, IBM's J9 is represented by a dashed line, and NSIcom CrEme is represented by a dotted line. It can be seen that the measured throughput using Jeode and J9 overlap, while in the case of CrEme the throughput is approximately four times larger. Similar conclusions can be drawn by examining Figure 5.15, with the only difference that J9 achieves slightly more throughput than Jeode. The average throughput values are summarised in Table 5.1. It can be seen that CrEme seems to be able to provide significantly higher throughput in PDAs running PocketPC 2002. The underlying reason is that, although each supported JRE fully satisfies the J2ME specification (*see* Appendix A), there are no strict guidelines on the actual implementation, and is thus possible for one system to achieve better performance than another. This is possibly achieved by utilising the device's wireless capability more intensively, as well as occupying more resources. This argument is examined further, in later sections. As far as the buffer size is concerned, there is no noticeable difference as all tested buffer sizes produced approximately the same throughput. Although CrEme may be the best JVM candidate for interpreting the Java-based proxy software, the throughput provided seems mostly suitable for routing small amounts of network traffic, such as e-mails and text, rather than heavy network

traffic, such as real-time audio and video.

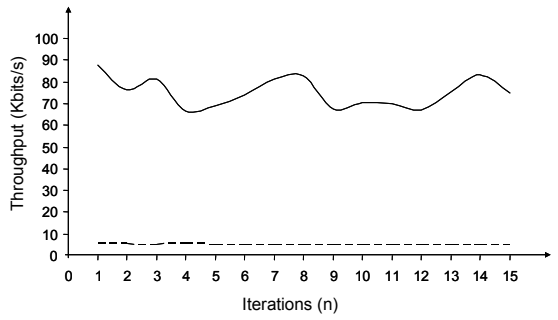

Figure 5.11: Throughput using Jeode, J9, and CrEme with buffer size of 1KB

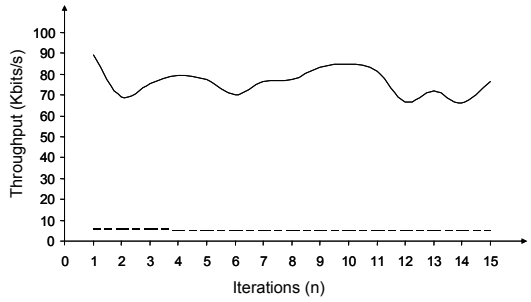

Figure 5.12: Throughput using Jeode, J9, and CrEme with buffer size of 2KB

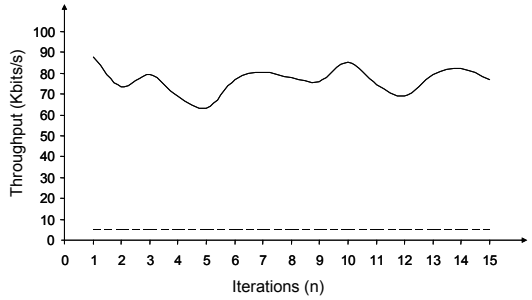

Figure 5.13: Throughput using Jeode, J9, and CrEme with buffer size of 4KB

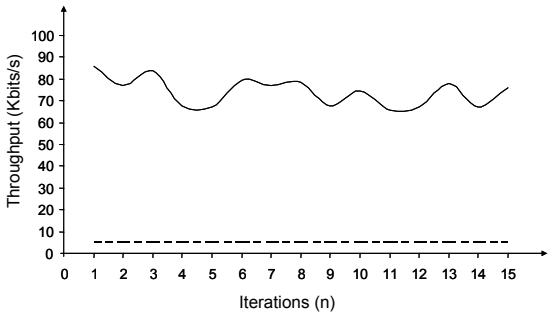

Figure 5.14: Throughput using Jeode, J9, and CrEme with buffer size of 8KB

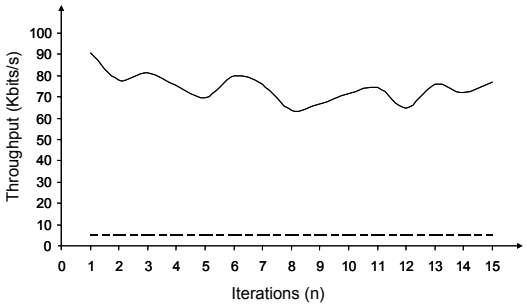

Figure 5.15: Throughput using Jeode, J9, and CrEme with buffer size of 16KB

Table 5.1: Average throughput for the PocketPC

PocketPC 2002 throughput (Kbits/s)			
	Jeode	J9	CrEme
1 KB	3.44	3.44	13.36
2 KB	3.49	3.45	13.41
4 KB	3.70	3.47	12.37
8 KB	3.38	3.44	13.23
16 KB	3.43	4.05	14.23

5.2.2 Battery discharge rate for PocketPC 2002

The PDA's battery discharge rate was measured while the PDA was routing network traffic from the source to the destination, as described in Section 4.2. Briefly, this was performed by measuring the time taken for the battery to discharge by 1%, until the experiment was finished, or until the battery reached 15% of its remaining capacity. Figures 5.16 - 5.20 present the battery discharge rate, for the selected buffer sizes and JVMs. Jeode is represented by a solid line, J9 with a dashed line, and CrEme with a dotted line. It can be seen that the battery discharge rate is similar for all tested JVMs and buffer sizes. The average battery discharge results are summarised in Table 5.2. With this, the battery discharge rates recorded using Jeode and CrEme were almost identical for all buffer sizes. In the same way, J9 accomplished an almost identical discharge rate for buffer size of 1KB, however, for the remaining buffer sizes it allowed a slower battery discharge of approximately 6s, that is, almost 10% more time for the battery to discharge by 1%. This means that the PDA can stay alive for 10 additional minutes, before the battery gets fully discharged Although J9 seems to slightly improve on battery life, it provides the worst routing throughput.

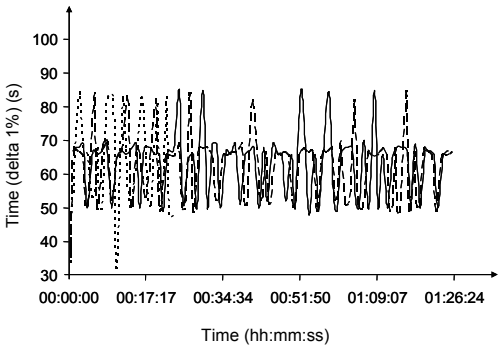

Figure 5.16: Battery discharge rate for buffer size of 1KB using Jeode, J9, and CrEme

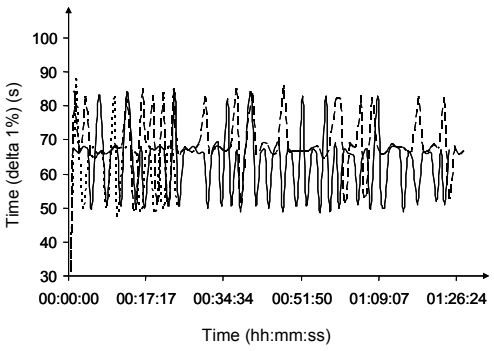

Figure 5.17: Battery discharge rate for buffer size of 2KB using Jeode, J9, and CrEme

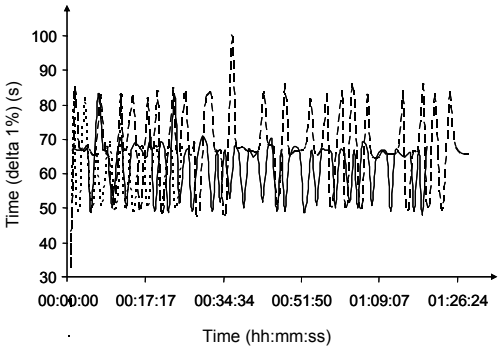

Figure 5.18: Battery discharge rate for buffer size of 4KB using Jeode, J9, and CrEme

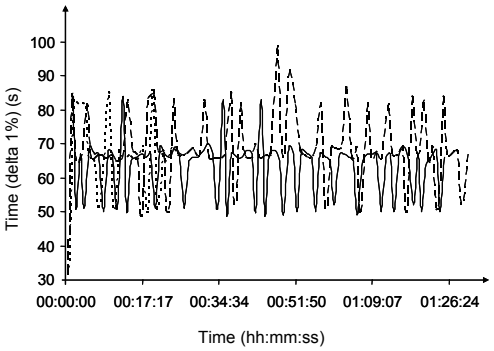

Figure 5.19: Battery discharge rate for buffer size of 8KB using Jeode, J9, and CrEme

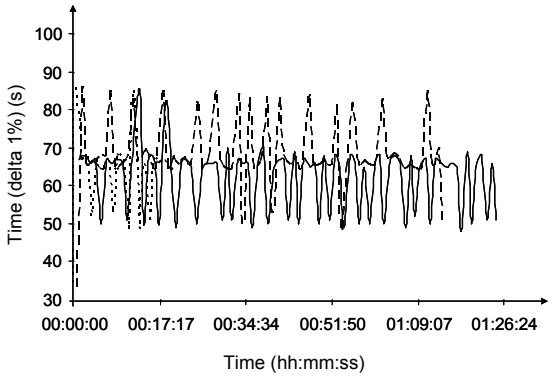

Figure 5.20: Battery discharge rate for buffer size of 16KB using Jeode, J9, and CrEme

Table 5.2: Average battery discharge rates for the PocketPC 2002

PocketPC 2002 Battery discharge rate Average time to change 1% (s)			
	Jeode	J9	CrEme
1 KB	62.24	62.80	61.52
2 KB	62.19	68.64	60.70
4 KB	61.41	68.50	60.36
8 KB	62.60	68.83	63.04
16 KB	61.32	68.49	61.57

5.2.3 CPU utilisation for PocketPC 2002

This section presents the utilisation of the PDA's CPU, while routing. It was estimated, that while the PDA was in an idle state, the CPU utilisation would be low, however, while the PDA was forwarding large amounts of network traffic, the CPU utilisation recorded was extremely high, reaching up to 97%. In addition to the proxy, processes managing the wireless features, where shown to further occupy CPU time, making routing over a wireless ad-hoc network even more CPU intensive.

Each CPU measurement for Jeode, J9, and CrEme was taken at fixed time intervals of 60s, 60s, and 20s respectively. The difference in time scale is justified by the fact that CrEme required approximately four times less time to complete the experiments (*see* Section 5.2.2). Thus, by taking CPU measurements more frequently for CrEme than the rest JVMs, it is guaranteed to obtain similar amount of data, which can assist in comparing the CPU utilisation imposed by each JVM. Figures 5.21 - 5.25 present the CPU utilisation for all tested buffer sizes and JVMs. Jeode is represented by a solid line, J9 with a dashed line, and CrEme with a dotted line. It can be seen that Jeode and J9 utilised the processor at a very similar rate, however CrEme's utilisation was of a higher order. The average CPU utilisation rates are summarised in Table 5.3. It can be seen that Jeode and J9 imposed similar CPU utilisation, approximately 64%. In contrast, CrEme

utilised the processor at a significantly higher order than the other two JVMs, which was approximately by 15%. The buffer size had no real significant influence on CPU utilisation.

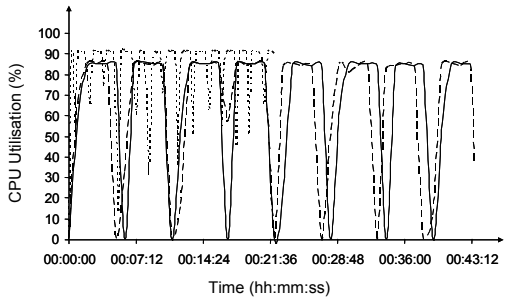

Figure 5.21: CPU utilisation for buffer size of 1 KB using Jeode, J9, and CrEme

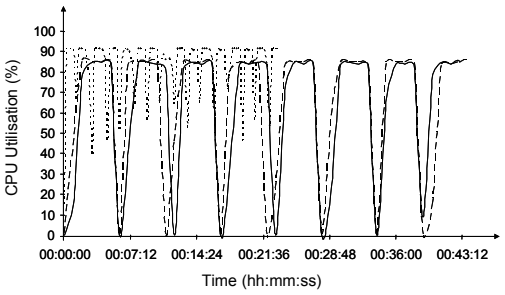

Figure 5.22: CPU utilisation for buffer size of 2 KB using Jeode, J9, and CrEme

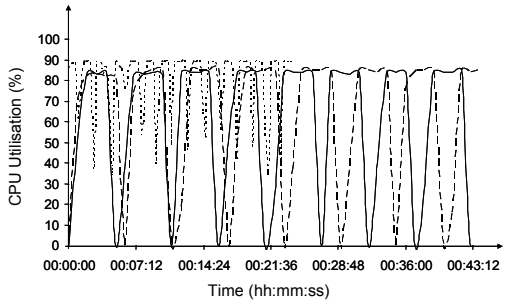

Figure 5.23: CPU utilisation for buffer size of 4KB using Jeode, J9, and CrEme

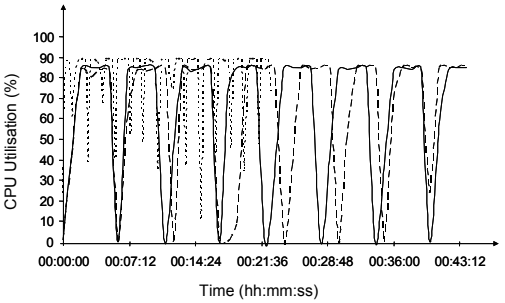

Figure 5.24: CPU utilisation for buffer size of 8KB using Jeode, J9, and CrEme

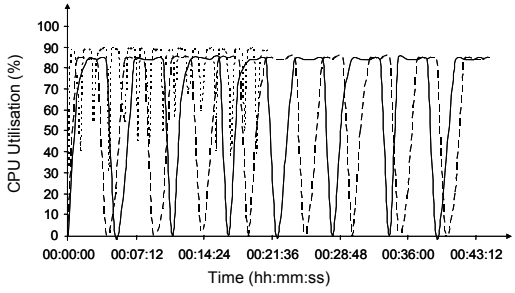

Figure 5.25: CPU utilisation for buffer size of 16KB using Jeode, J9, and crème

Table 5.3: Average CPU utilisation for the PocketPC 2002

PocketPC 2002 CPU utilisation (%)			
	Jeode	J9	CrEme
1 KB	64.96	63.92	80.01

2	KB	63.30	64.04	81.47
4	KB	63.15	64.26	78.13
8	KB	65.02	64.65	78.13
16	KB	64.31	59.47	76.81

5.2.4 Heap memory usage for PocketPC 2002

This section presents the heap memory usage, which was recorded throughout the routing process. Results are focused on the amounts of memory used by Java *Objects* while routing, rather than the amounts of memory used by the JVM as a process, which normally remains almost constant. In this way, the experiment may provide insight into the memory allocation and management, which is performed by each JVM, and possibly provide a means of explaining their difference performance in terms of throughput. Measurements were taken at fixed time intervals of 1 (s) for all JVMs. Figures 5.26 - 5.30 present the memory usage for all buffer sizes using Jeode, J9, and CrEme. Jeode is represented by a solid line, J9 with a dashed line, and CrEme with a dotted line. It can be clearly seen that Jeode and J9 employed similar amounts of heap memory, for all buffer sizes. In contrast, CrEme employed significantly higher heap memory, almost three times more than the other two JVMs. Buffer size did not influence the heap memory usage. The average values are summarised in Table 5.4.

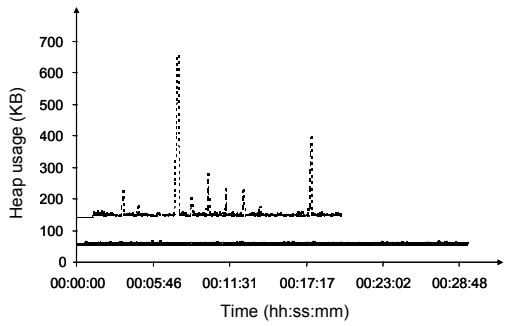

Figure 5.26: Memory usage for buffer size of 1KB using Jeode, J9, and CrEme

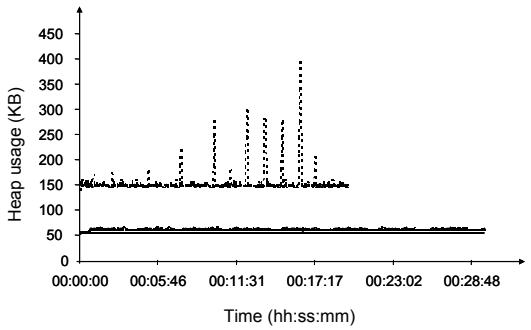

Figure 5.27: Memory usage for buffer size of 2KB using Jeode, J9, and CrEme

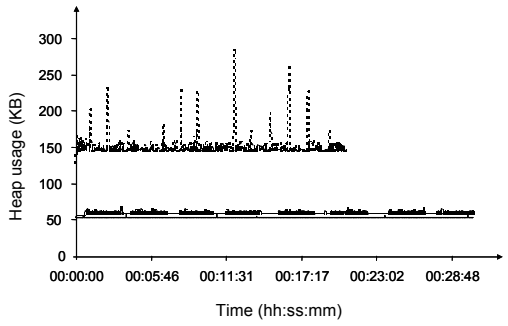

Figure 5.28: Memory usage for buffer size of 4KB using Jeode, J9, and CrEme

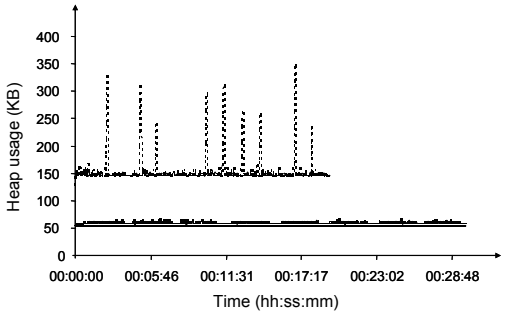

Figure 5.29: Memory usage for buffer size of 8KB using Jeode, J9, and CrEme

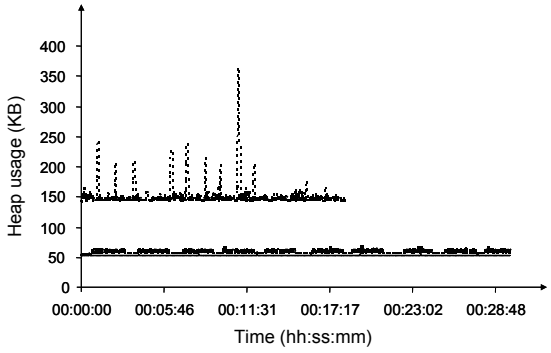

Figure 5.30: Memory usage for buffer size of 16KB using Jeode, J9, and CrEme

Table 5.4: Average heap usage for PocketPC 2002.

PocketPC 2002 Heap memory usage (KB)			
	Jeode	J9	CrEme
1 KB	53.95	59.11	151.71
2 KB	53.93	59.06	149.25
4 KB	53.92	58.95	148.58
8 KB	53.95	59.06	150.28
16 KB	53.95	59.05	148.67

5.2.5 Evaluation of PocketPC 2002 results

The purpose of this section is to summarise results presented for the PDA, and evaluate their significance. As previously stated, the main aim of the experiments was to identify whether the JVM, which provides the runtime environment for the Java-based proxy software, has any significant importance on the routing device's performance, especially in terms of throughput, battery discharge rate, CPU utilisation, and heap memory usage. An additional aim was to investigate, whether buffer size could influence the device's performance.

According to Tables 5.1 - 5.4, it can be observed that all average values obtained for Jeode and J9 in relation to throughput, battery consumption rate, CPU utilisation, and heap usage, were indeed similar to each other, for all buffer sizes. Thus, it may be safe to conclude that a wireless proxy-based PDA running PocketPC 2002, with either Jeode or J9, performs equally the same, in all aspects. Major differences were observed by experimenting with CrEme on PocketPC 2002. These differences mostly concerned throughput, CPU utilisation, and heap memory usage. In more depth, the offered throughput, while the PDA was routing heavy network traffic, was of a magnitude of approximately three times more than its counterparts. In addition, the CPU utilisation was increased by a value of 14%, and three times more heap memory was utilised by Java *Ob-*

jects. In contrast, the battery discharge rate was similar to the other two JVMs. This factor is especially important for devices that depend on battery power.

The data presented above make CrEme the most powerful candidate for wireless proxy-based PDAs running PocketPC 2002, as far as routing is concerned. However, the smaller footprints of Jeode and J9 may make them more appropriate for tasks that do not require higher rates of throughput.

5.2.6 Throughput of a PDA running PocketPC 2003

The throughout measurements for J9 and CrEme, with all tested buffer sizes are presented in Figures 5.31 - 5.35. CrEme is represented by a solid line and J9 is represented by a dashed line. It can be seen that the throughput measurements for J9 and CrEme significantly differed. CrEme achieved a significantly higher throughput than J9 of approximately 13 times more. This observation was repeated for all buffer sizes. The average throughput values are summarised in Table 5.5. It can be seen that J9 achieves a fairly stable throughput for all buffer sizes which is approximately 5Kbits/s. Along the same line, CrEme achieves the same throughput for all buffer sizes which is approximately 75Kbits/s. However, CrEme and J9 have a significant difference in terms of throughput, where CrEme achieves 13 times more throughput than J9. In other words, if a CrEme-enabled proxy-PDA requires 10 seconds to route a certain amount of network traffic, a J9-enabled proxy-PDA requires more than two minutes to route the same amount of network traffic. In addition, CrEme may be capable of routing low-bandwidth real-time traffic such as voice and video data, since it provides nearly one and a half times more throughput than a 56 Kbits/s modem. J9 cannot offer as high throughput, and thus it may be more suitable for e-mail, text, and low-resolution graphics network traffic rather than real-time voice, or video.

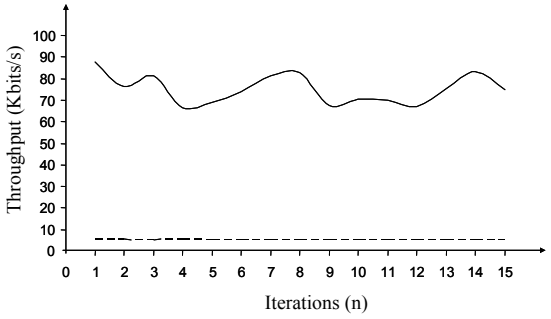

Figure 5.31: Throughput using J9 and CrEme with buffer size of 1 KB

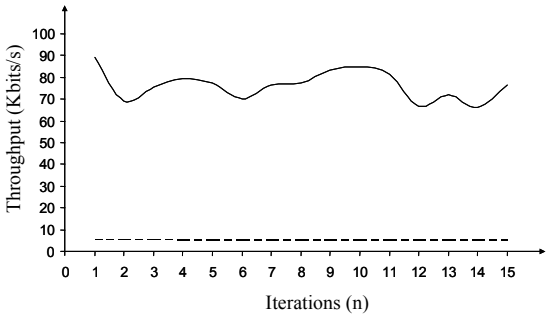

Figure 5.32: Throughput using J9 and CrEme with buffer size of 2 KB

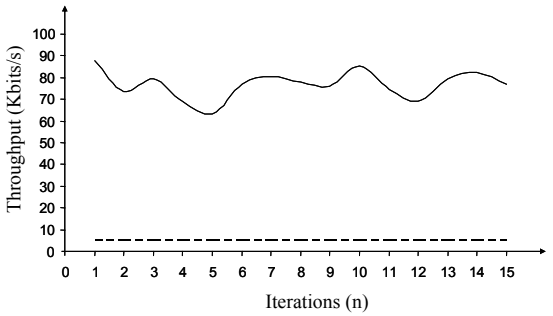

Figure 5.33: Throughput using J9 and CrEme with buffer size of 4 KB

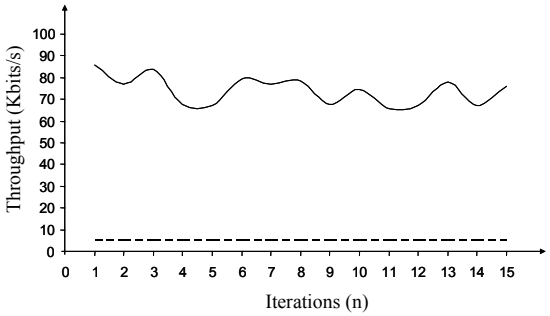

Figure 5.34: Throughput using J9 and CrEme with buffer size of 8 KB

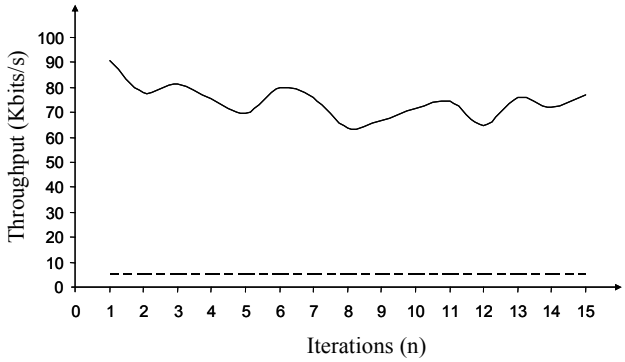

Figure 5.35: Throughput using J9 and CrEme with buffer size of 16 KB

Table 5.5: Average throughput values for PocketPC 2003

PocketPC 2003 Throughput (Kbits/s)		
	J9	CrEme
1 KB	5.08	75.10
2 KB	5.08	76.34
4 KB	5.00	76.80
8 KB	4.99	74.14
16 KB	4.99	74.55

5.2.7 Battery discharge rate for the PocketPC 2003

The PDA's battery discharge rate was measured using the same procedure, as described in section 5.2.3. Figures 5.36 - 5.40 present the battery discharge rate experienced by the PDA for CrEme and Jeode, and all tested buffer sizes. J9 is represented by a dashed line while CrEme by a bolded solid line. It can be clearly seen that the whole range of experiments finished much faster with CrEme than J9. This is due to higher throughput offered by CrEme. The battery discharge rate is similar for both J9 and CrEme for all tested buffer sizes. The average values are summarised in Table 5.6. It can be seen that the battery discharge rates achieved using J9 are approximately the same for all buffer

sizes, which is on an average approximately 70s. Similarly, the battery discharge rates achieved using CrEme are approximately the same for all buffer sizes, which is on an average approximately 60 (s). Therefore it can be seen that CrEme forces the battery to discharge at a higher rate than J9. On average, the battery requires 10 additional seconds to discharge by 1% using J9 than what it requires using CrEme. As a result, the battery life of the J9-enabled proxy-PDA can stay alive for approximately 17 minutes more than a CrEme-enabled proxy-PDA, before the battery gets fully discharged.

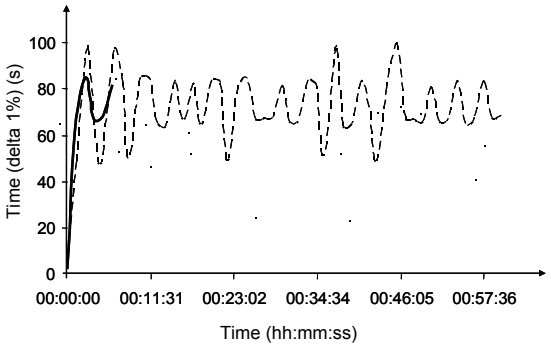

Figure 5.36: Battery discharge rate for buffer size of 1 KB using J9 and CrEme

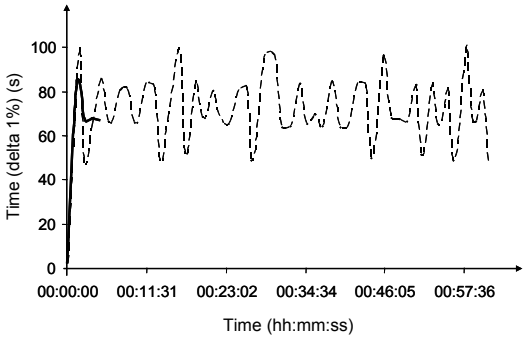

Figure 5.37: Battery discharge rate for buffer size of 2 KB using J9 and CrEme

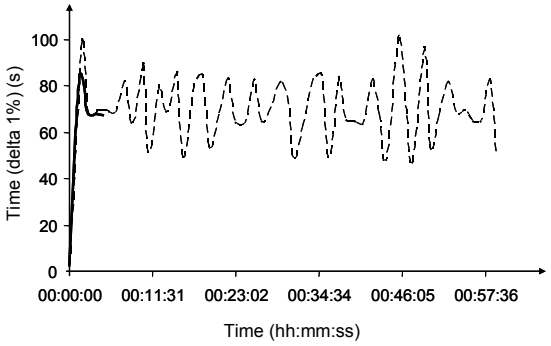

Figure 5.38: Battery discharge rate for buffer size of 4 KB using J9 and CrEme

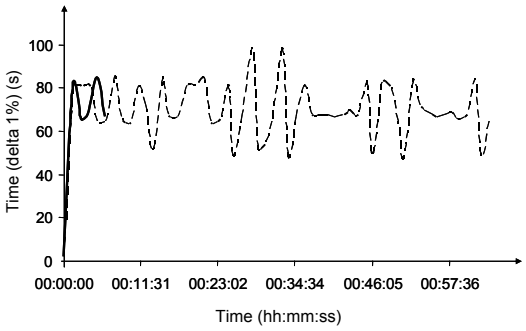

Figure 5.39: Battery discharge rate for buffer size of 8 KB using J9 and CrEme

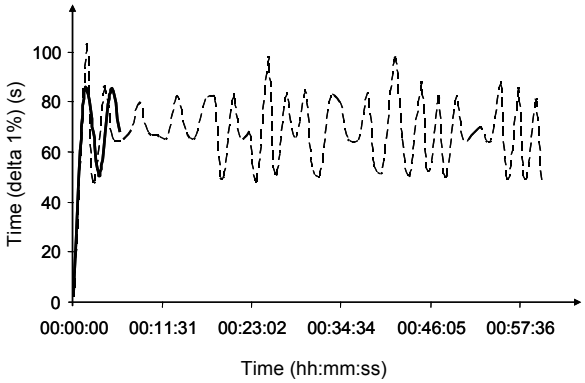

Figure 5.40: Battery discharge rate for buffer size of 8 KB using J9 and CrEme.

Table 5.6: Average battery discharge rates for Pocket PC 2003

PocketPC 2003 Battery discharge rate Average time to change 1% (s)		
	J9	CrEme
1 KB	71.86	61.83
2 KB	71.94	57.60
4 KB	69.68	57.80
8 KB	69.27	61.66
16 KB	68.49	59.66

5.2.8 CPU utilisation for the PocketPC 2003

This section presents the CPU utilisation of the proxy-PDA. Measurements were taken at fixed time intervals of 40s for J9 and 3s for CrEme. This difference in the measurements scale is justified by the difference in the time taken to complete the experiments (*see* Section 5.2.7). Figures 5.41 - 5.45 present the CPU utilisation for all tested buffer sizes with J9 and CrEme. J9 is represented by a dashed line while CrEme by a solid line. It can be clearly seen that CrEme had finished the whole range of experiments much faster due to its faster throughput, as explained in Section 5.2.7. CrEme utilised the CPU in a higher intensity that J9, and also in less time. The average CPU utilisation values are summarised in Table 5.7. It can be seen that J9 utilises the CPU similarly for all buffer sizes, which is on an average of approximately 70%, while CrEme utilises the CPU on an average of approximately 78%. Although CrEme's CPU utilisation for most buffer sizes slightly varies, this does not present a significant difference and may be considered as an oversight. In addition, CrEme utilises the CPU by 8% more than J9, on average. It was observed throughout the experimentation that the CrEme-enabled proxy-PDA managed to connect instantly, with the transmitting and receiving devices, at the start of each iteration, whereas the J9-enabled proxy-PDA suffered considerable delays, which are illustrated throughout Figures 5.41-5.45, as its CPU utilisation drops low at various points.

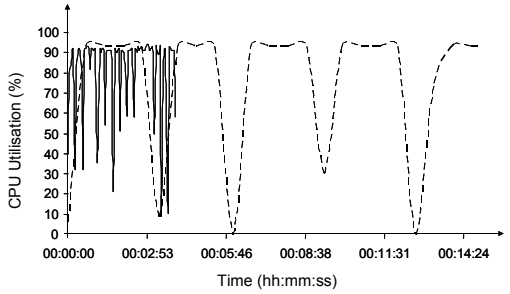

Figure 5.41: CPU utilisation for buffer size of 1KB using J9 and CrEme

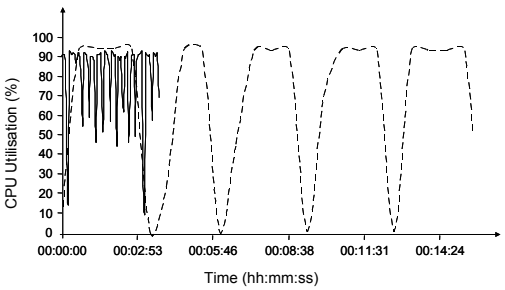

Figure 5.42: CPU utilisation for buffer size of 2KB using J9 and CrEme

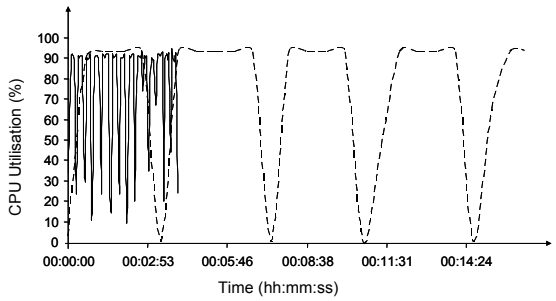

Figure 5.43: CPU utilisation for buffer size of 4KB using J9 and CrEme

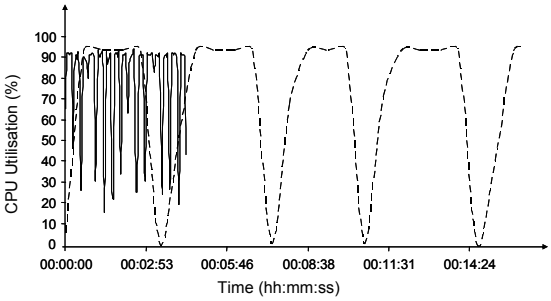

Figure 5.44: CPU utilisation for buffer size of 8KB using J9 and CrEme

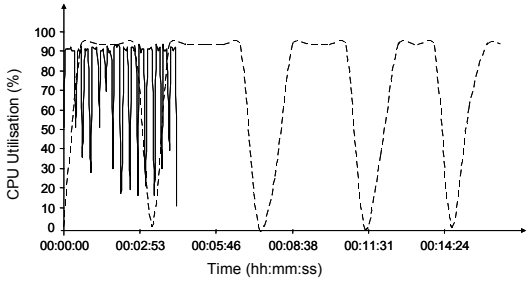

Figure 5.45: CPU utilisation for buffer size of 16KB using J9 and CrEme

Table 5.7: Average CPU utilisation for the PocketPC 2003

PocketPC 2003 CPU utilisation (%)		
	J9	CrEme
1 KB	70.86	80.29
2 KB	68.11	80.08
4 KB	71.28	74.25
8 KB	70.70	76.88
16 KB	69.67	77.89

5.2.9 Heap memory usage for the PocketPC 2003

This section presents the heap memory usage recorded throughout the routing process. Measurements were taken at fixed time intervals of 1 (s) for both J9 and CrEme. Figures 5.46 - 5.50 present the heap memory usage for all buffer sizes using J9 and CrEme. J9 is represented by a dashed line and CrEme by a solid line. It can be seen that CrEme's Java *Objects* employ significantly more heap memory than J9's *Objects* in order to route the data. CrEme allows Java *Objects* to use up to 10 times more heap memory. The average heap memory usage results are summarised in Table 5.8. It can be seen that J9 *Objects* used the same amount of heap memory for all buffer sizes, which is approximately 60KB, on average. CrEme *Objects* used similar amounts of heap memory for all buffer sizes,

which is approximately 615KB, on average. This is a significant difference, as CrEme *Objects* may use up to 10 times more heap memory than J9 *Objects*, which means that CrEme *Objects* can buffer more data as they arrive from the transmitting device, which may be a factor that enforces the higher rates of throughput. Furthermore, it seems that buffer size has not any significant effect on heap memory usage.

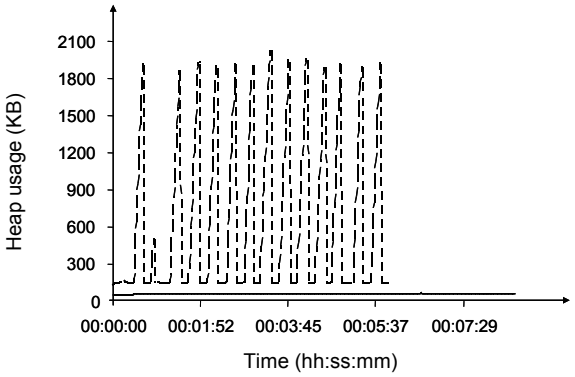

Figure 5.46: Memory usage for buffer size of 1KB using J9 and CrEme

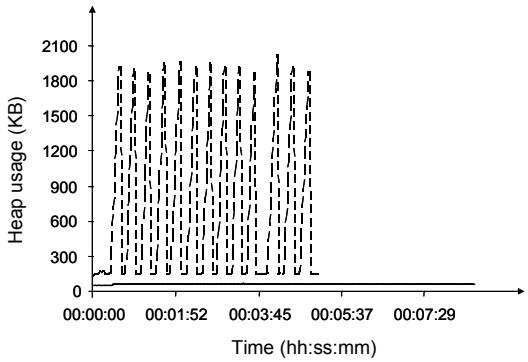

Figure 5.47: Memory usage for buffer size of 2KB using J9 and CrEme

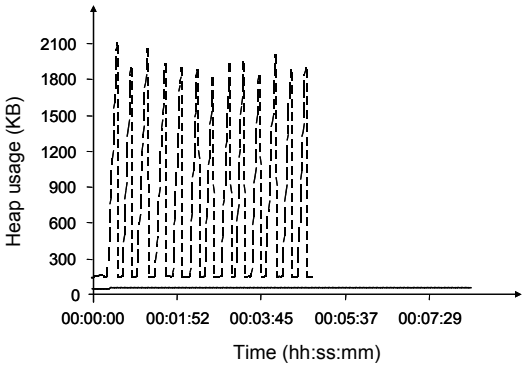

Figure 5.48: Memory usage for buffer size of 4KB using J9 and CrEme

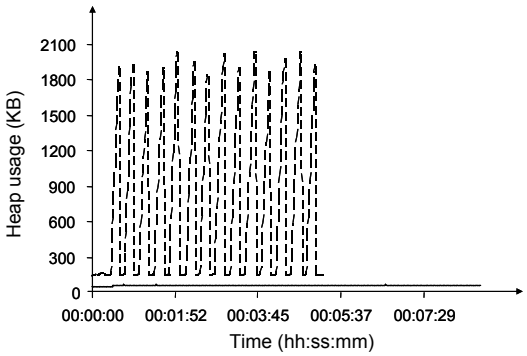

Figure 5.49: Memory usage for buffer size of 8KB using J9 and CrEme

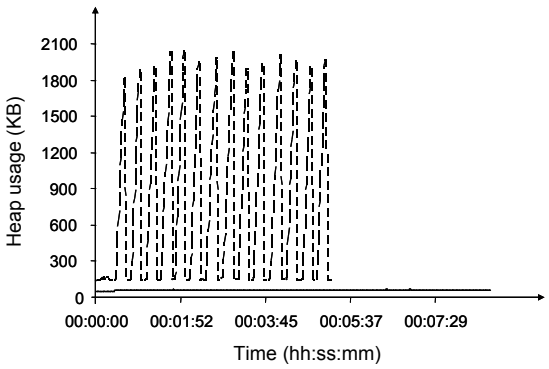

Figure 5.50: Memory usage for buffer size of 16KB using J9 and CrEme

Table 5.8: Average heap usage for PocketPC 2003

PocketPC 2003 heap memory usage (KB)		
	J9	CrEme
1 KB	60.22	547.18
2 KB	60.04	601.42
4 KB	60.31	621.75
8 KB	60.10	648.62
16 KB	60.18	648.23

5.2.10 Evaluation of PocketPC 2003 results

This section summarises results presented in previous sections, concerning PocketPC 2003, and evaluates their significance. CrEme officially supports both PocketPC 2002 and PocketPC 2003 OSs, although J9 officially supports only PocketPC 2002. However, it has been seen that J9, running on top of PocketPC 2003, provided significantly more throughput, than on top of PocketPC 2002. The same was also observed in the case of CrEme, and thus it seems that Microsoft's PocketPC 2003 considerably improved upon the earlier version.

In relation to Table 5.5 and Table 5.8, it can be clearly seen that the JVMs used can have enormous differences concerning throughput and heap memory usage. However, in terms of battery consumption rate and CPU utilisation the measurements are shown to be fairly similar. CrEme can increase the available throughput up to 13 times in comparison to J9. The high demand of throughput in ad-hoc networks makes CrEme, on top of PocketPC 2003, the strongest candidate amongst its counterparts so far. However, it has to be noted that IBM J9 does not officially support PocketPC 2003 platforms.

The heap memory usage may be a crucial factor assisting CrEme to perform exceptionally well. As shown, CrEme *Objects* can use up to 10 times more heap memory than J9 *Objects*. In this way, the CrEme-enabled PDA moves data faster, as it can buffer large amounts of received data before they are transmitted, which seems that enhances its proxy capability, especially if a device cannot send and receive data at the same time. However, other factors could have contributed, including: better implementation practices specifically aiming PocketPC 2003 platform; better utilisation of the wireless interface; and so on.

Furthermore, by examining throughput results presented so far, it can be deduced that, on average, J9 increased the throughput by 31%, when PocketPC 2003 platform was used instead of PocketPC 2002. This fact was consistent among all tested buffer sizes. Along the same line, CrEme on top of Pocket 2003 improved the provided throughput by 558% than on top of PocketPC 2002. Again, this was true for all tested buffer sizes. Finally, it was shown that buffer size did not significant influence throughput, battery consumption rate, CPU utilisation, or heap memory usage in either J9 or

CrEme results.

5.2.11 Throughput of a PDA running Familiar Linux

In a similar direction as to the one described in Sections 5.2.1 and 5.2.6, the Familiar Linux PDA was used as a proxy, forwarding a fixed amount of data from a source to a destination device. All communications were point-to-point and were using wireless as the communications medium. This set of experiments was repeated 15 times, with varying buffer sizes, and the average throughput was measured. The implementation details of the experiment are presented in Section 4.3.3. Figures 5.51 - 5.55 illustrates the throughput measured for both Java 1.3 and J9 and for all tested buffer sizes. Java 1.3 is denoted by a dashed line, while J9 is denoted by a solid line.

It can be seen that throughput measurements for Java 1.3 were considerably lower than J9. It arises that J9 can provide up to approximately three times more throughput than Java 1.3. This is true for all buffer sizes presented above, and the average throughput values are summarised in Table 5.9. It can been seen that buffer size does not have any significance in either Java 1.3 or J9 throughput measurements. However, there are some insignificant differences in the throughput that may be caused by the unreliable nature of the wireless medium. The average throughput of all tested buffer sizes for Java 1.3 is 13.428Kbits/s, while for J9 is 41.542Kbits/s, which means that J9 can be approximately up to three times faster than Java 1.3. The throughput achieved by J9 is almost equal to a 56Kbits/s modem, while Java 1.3 just reaches the one third of that. According to these results, J9 could be used to route moderate network traffic such as text, e-mails, low quality real-time voice, sounds, and graphics. On the contrary, Java 1.3 may be more appropriate for low network traffic such as text, e-mails, low quality graphics, and so on.

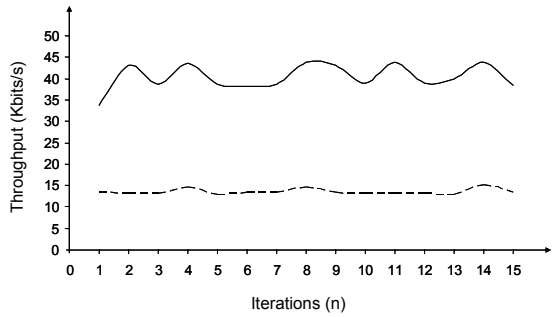

Figure 5.51: Throughput using Java 1.3 and J9 for buffer size of 1KB

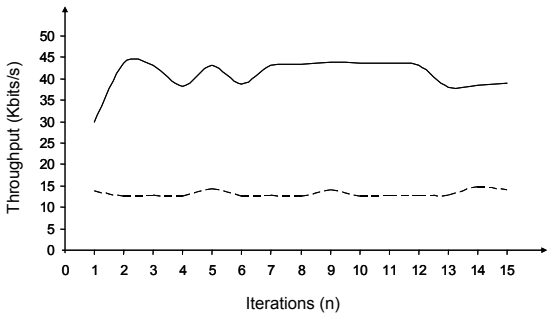

Figure 5.52: Throughput using Java 1.3 and J9 for buffer size of 2KB

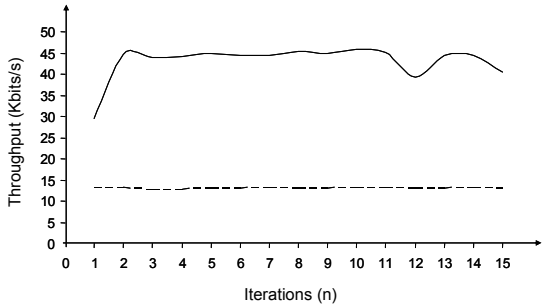

Figure 5.53: Throughput using Java 1.3 and J9 for buffer size of 4KB

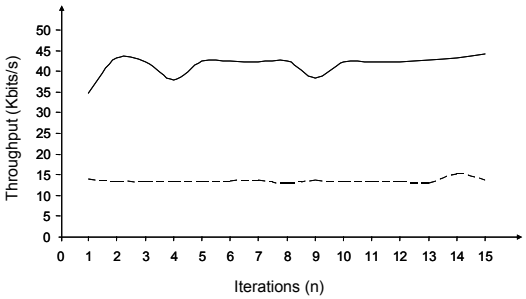

Figure 5.54: Throughput using Java 1.3 and J9 for buffer size of 8KB

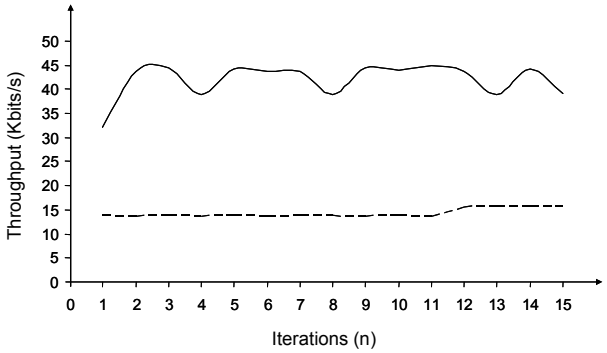

Figure 5.55: Throughput using Java 1.3 and J9 for buffer size of 16KB

Table 5.9: Average throughput for the Familiar Linux PDA

Familiar Linux throughput (Kbits/s)		
	Java 1.3	J9
1 KB	13.428	40.371
2 KB	13.115	40.862
4 KB	13.022	43.047
8 KB	13.397	41.493
16 KB	14.182	41.939

5.2.12 Battery discharge rate for the Familiar Linux

The PDA's battery discharge rate was measured throughout the series of experiments presented in the previous section. The purpose of these measurements was to investigate whether the large footprint of Java 1.3 or the light-weighted footprint of J9 (*see* Section 4.3.3) can have any significant difference in battery discharge rate. Figures 5.56 - 5.60 present the battery discharge rate, measured in seconds, against the total time taken to complete the whole range of experiments. The dashed line denotes Java 1.3 and the bolded solid line denotes J9.

It can be clearly seen that more measurements were taken for Java 1.3 than J9, which is a direct result of the fact that J9 completed the experiment much faster than Java 1.3

(*see* Section 5.2.11). In addition, it can be seen that the battery discharge rate measurements for Java 1.3 are more stable than the respective ones for J9. Overall, while Java 1.3 maintains a quite solid line for all presented buffer sizes, J9 experiences sudden drops of the battery. It has to be noted that IBM J9 does not officially support J9 for Familiar Linux platforms, and as it seems J9 does not fully adapts to this platform. However, as the results suggest the overall average for both JVMs is approximately equal. These average battery discharge rates are summarised in Table 5.10. It can be seen that Java 1.3 discharges the battery at a stable rate across all tested buffer sizes, which is on average approximately 83s. J9 discharges the battery at a very similar rate which is approximately 82s, even though the rate varies across different tested buffer sizes. For instance, buffer size of 16KB produces the lowest discharge rate of 68.142s, while a buffer size of 8KB produces the highest of 92.857s. However, these observed anomalies do not provide strong evidence that buffer size can influence the battery discharge rate, and may thus be attributed to the problematic power management offered by Familiar Linux. Despite this, both JVMs forced the battery to discharge at the same rate and may thus be considered equal in this aspect.

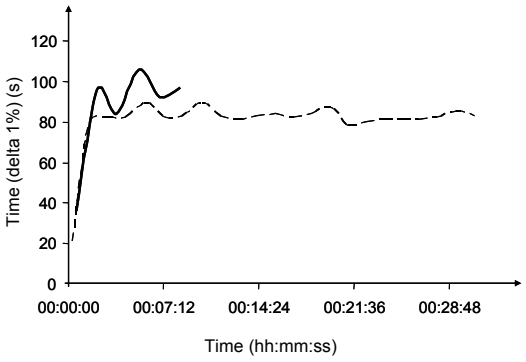

Figure 5.56: Battery discharge rate for buffer size of 1KB using Java 1.3 and J9

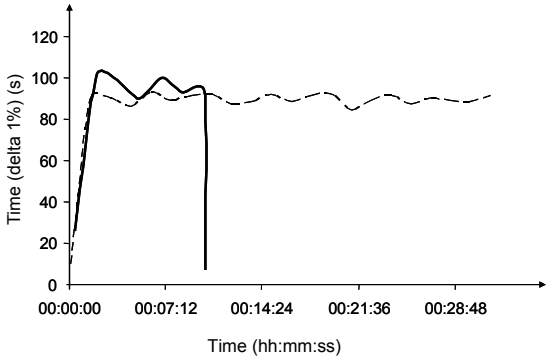

Figure 5.57: Battery discharge rate for buffer size of 2KB using Java 1.3 and J9

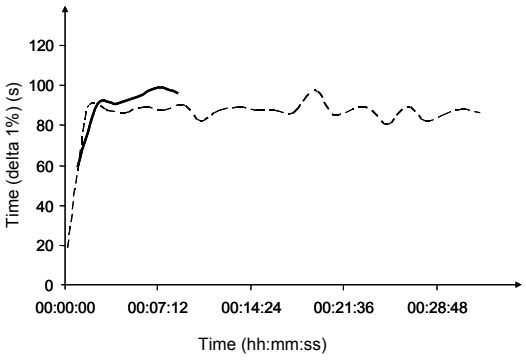

Figure 5.58: Battery discharge rate for buffer size of 4KB using Java 1.3 and J9

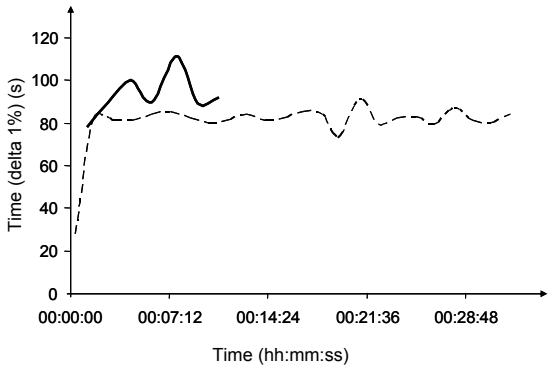

Figure 5.59: Battery discharge rate for buffer size of 8KB using Java 1.3 and J9

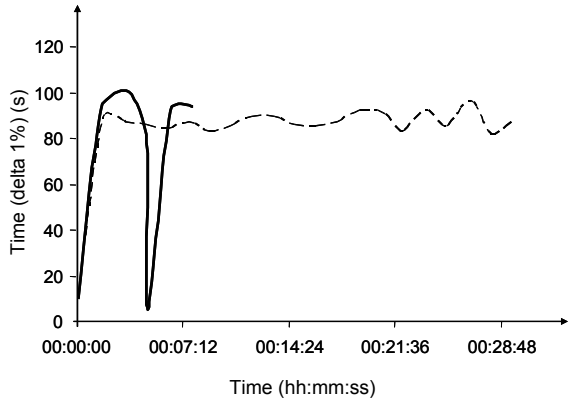

Figure 5.60: Battery discharge rate for buffer size of 16KB using Java 1.3 and J9

Table 5.10: Average battery discharge rates for Familiar Linux

Familiar Linux battery discharge rates Average time to change 1% (s)		
	Java 1.3	J9
1 KB	80.13	85.16
2 KB	85.81	76.25
4 KB	84.04	88.33
8 KB	80.00	92.85
16 KB	84.28	68.14

5.2.13 CPU utilisation for the Familiar Linux

This section presents the CPU utilisation of the Familiar Linux PDA experienced by the PDA throughout this series of experiments. Measurements were taken for both Java 1.3 and J9 at fixed time intervals of 20s and 5s respectively. The difference in the time scale is justified by the fact that J9 completed the whole range of experiments much faster than Java 1.3, and thus CPU measurements should have been scheduled at shorter time intervals, in order to record approximately the same CPU values for both JVMs. Figures 5.61 - 5.65 presents the CPU utilisation for all tested buffer sizes using Java 1.3 and J9 against the time taken to complete the experiments. Java 1.3 is represented by a dashed line, while J9 by a solid line.

It can be clearly seen that J9 had finished the whole range of experiments much faster due to its higher rate of throughput, as explained in Section 5.2.11. Another important issue is that J9 had some of its values close to zero, while Java 1.3 maintained all of its measurements above 20%. The reason for this is that J9 experienced short connection delays, at the start of each iteration, in a similar way to J9 (*see* Section 5.2.8). These average CPU utilisation values are summarised in Table 5.11. It can be seen that Java 1.3 utilised the CPU evenly across all tested buffer sizes, which was approximately 87%, while for J9 it was approximately 66%. These results highlight a significant difference in

terms of CPU utilisation. Java 1.3 used almost 21% more CPU time than J9, while it provided approximately one-third of the throughput that J9 achieved. This proves the initial hypothesis that the large footprint of Java 1.3 could have a negative effect on performance, and further suggests that an optimised smaller JVM may be more suitable for resource-limited devices (*see* Section 4.3.3). In addition, it shows that increased CPU utilisation, at least in the case of Java 1.3, does not have a positive effect on throughput (*see* Section 5.2.11). However, the full version of Java 1.3 is not restricted to the smaller subset of available classes, which is supported by Sun's J2ME (*see* Appendix A), and thus provides more useful features. Furthermore, it was shown that even though Java 1.3

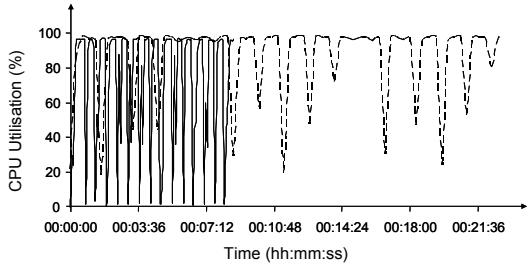

Figure 5.61: CPU utilisation for buffer size of 2 KB using Java 1.3 and J9

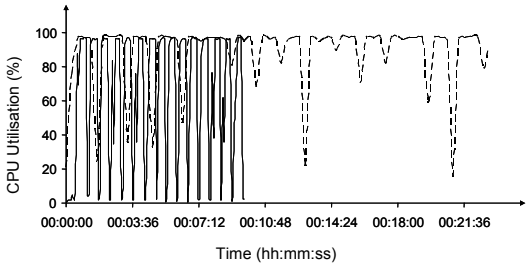

Figure 5.62: CPU utilisation for buffer size of 2 KB using Java 1.3 and J9

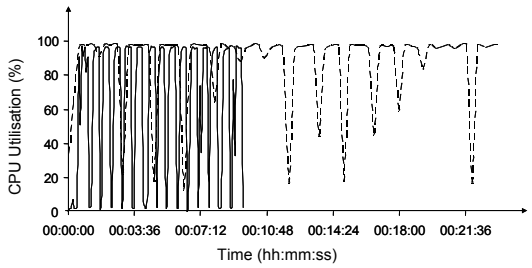

Figure 5.63: CPU utilisation for buffer size of 4 KB using Java 1.3 and J9

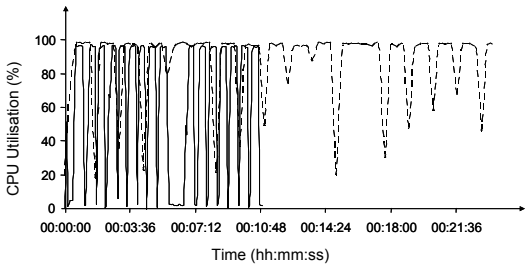

Figure 5.64: CPU utilisation for buffer size of 8 KB using Java 1.3 and J9

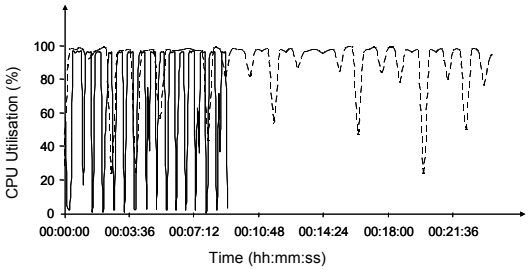

Figure 5.65: CPU utilisation for buffer size of 16 KB using Java 1.3 and J9

Table 5.11: CPU utilisation for Familiar Linux

Familiar Linux CPU utilisation (%)		
	Java 1.3	J9
1 KB	86.684	71.751
2 KB	86.120	65.047
4 KB	86.021	63.669
8 KB	86.611	62.937
16 KB	88.106	68.963

5.2.14 Heap memory usage for Familiar Linux

This section presents the heap memory usage required by the Java 1.3 and J9 *Objects* throughout the data routing process. As in Sections 5.2.4 and 5.2.9, results are focused on the amounts of memory used by Java *Objects* rather than the amounts of memory used by the JVM as a process. Measurements were taken at fixed time intervals of 1s for both Java 1.3 and J9. Figures 5.66 - 5.70 presents the heap memory usage throughout the whole series of experiments, including all tested buffer sizes. J9 is represented by a dashed line and Java 1.3 by a solid line. It can be clearly seen that Java 1.3 required significantly more heap memory than J9, almost double the amount. Also, Java 1.3 heap memory measurements were steady across all tested buffer sizes. In contrast, J9 measurements were similar across all tested buffer sizes, however, values seemed to vary between relatively low and high end-points. The average heap memory results are summarised in Table 5.12. It can be clearly seen that the heap memory used by Java 1.3 *Objects* was more stable across all tested buffer sizes, which was on average approximately 211KB. The J9 heap memory usage was on average approximately 77.192KB. By comparing these two JVMs, it appears that Java 1.3 *Objects* require approximately three times more heap memory that J9 *Objects*. These results further suggest that the full version of Java 1.3 utilises more resources than an optimised JVM targeted for handheld devices, such as J9.

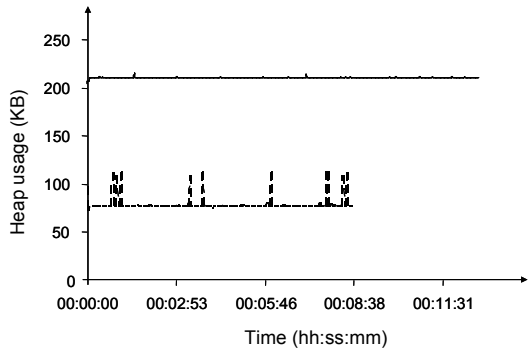

Figure 5.66: Memory usage for buffer size of 1 KB using Java 1.3 and J9

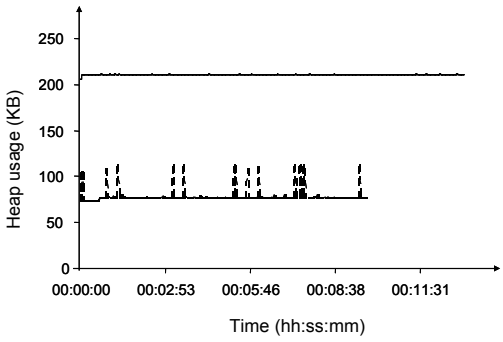

Figure 5.67: Memory usage for buffer size of 2 KB using Java 1.3 and J9

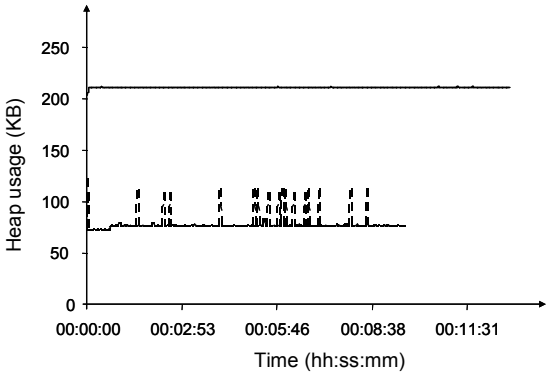

Figure 5.68: Memory usage for buffer size of 4 KB using Java 1.3 and J9

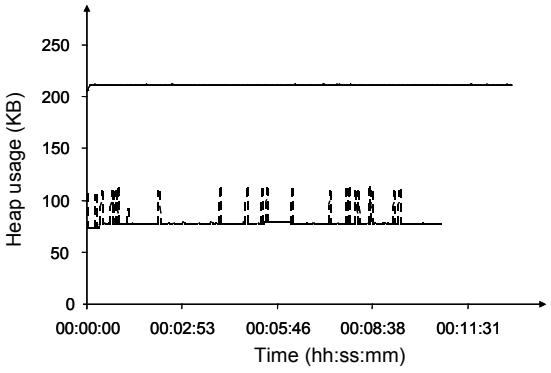

Figure 5.69: Memory usage for buffer size of 8 KB using Java 1.3 and J9

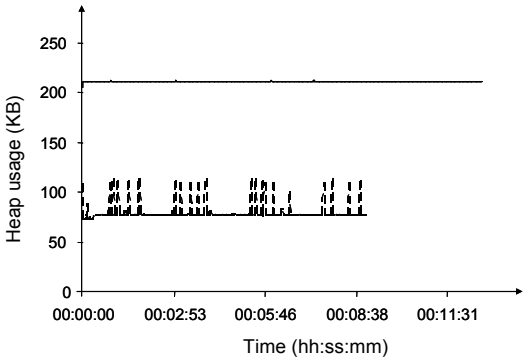

Figure 5.70: Memory usage for buffer size of 16 KB using Java 1.3 and J9

Table 5.12: Average heap memory usage for Familiar Linux

Familiar Linux memory heap usage (KB)		
	Java 1.3	J9
1 KB	210.925	71.751
2 KB	210.929	65.047
4 KB	210.929	63.669
8 KB	210.923	62.937
16 KB	210.937	68.963

5.2.15 Temperature for Familiar Linux

Unlike PocketPC 2002 and 2003, the temperature monitoring under Familiar Linux was trivial. Thus, this platform was preferable for the implementation of a temperature monitor. This section presents the temperature variation that occurred during the routing process. Readings were taken at the exact time when the battery capacity was being reduced by 1%. Thus, the temperature readings are as many as the battery readings, as they were taken at exactly the same time. Figures 5.71 - 5.75 present the temperature variation

experienced by Java 1.3 and J9 for all tested buffer sizes. Temperatures were measured in Celsius (°C) and are plotted against the total time taken to complete the experiments. Java 1.3 is represented by a dashed line, while J9 is represented by a bolded solid line.

It can be seen that the temperature is rising for most of the time, remaining steady for less, and rarely reached negative values during which the PDA was cooling down. Similar patterns were observed for both Java 1.3 and J9. These average temperature variation results are summarised in Table 5.13. It can be seen that Java 1.3 increased the internal temperature of the PDA across all buffer sizes, apart from 2 KB where the total difference in temperature was zero. The average increase in temperature was approximately 0.082 °C for every reduction of the battery's capacity by 1%. Similarly, J9 increased the internal temperature of the PDA across all buffer sizes, which on average was approximately 0.128 °C. Therefore, J9 was shown to have a more intense temperature effect on the PDA than Java 1.3, which may be associated with the increased throughput provided by it. However, for Java 1.3, the PDA started at an internal temperature of 32 °C and finished with a temperature of 39.875 °C, while for J9 the PDA started at 28.625 °C and finished at 33.25 °C. Thus, Java 1.3 completed the whole range of experiments and experienced an increase in the temperature of 7.875 °C, while J9 of 4.625 °C. Unfortunately, part of this difference is associated with the more time taken by Java 1.3 to conduct the experiments.

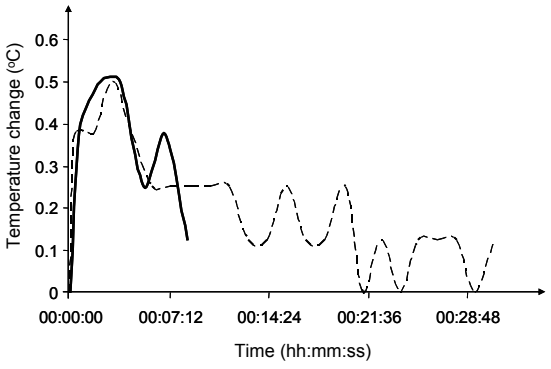

Figure 5.71: Temperature variation using Java 1.3 and J9 for buffer size of 1 KB

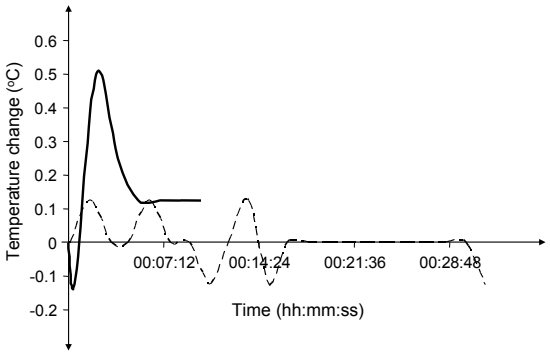

Figure 5.72: Temperature variation using Java 1.3 and J9 for buffer size of 2 KB

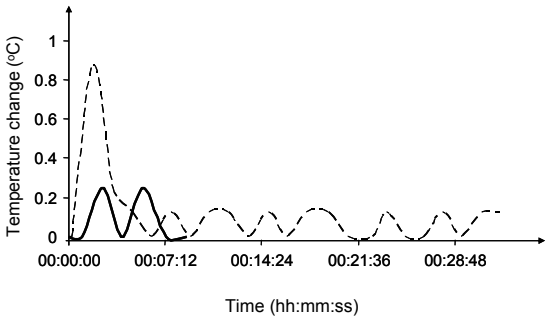

Figure 5.73: Temperature variation using Java 1.3 and J9 for buffer size of 4 KB

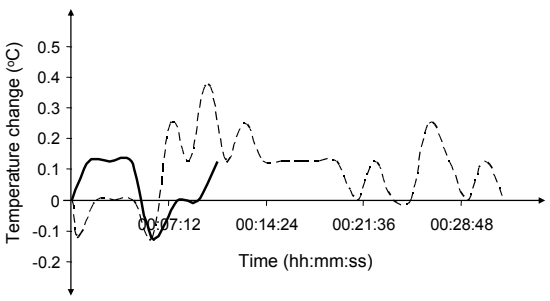

Figure 5.74: Temperature variation using Java 1.3 and J9 for buffer size of 8 KB

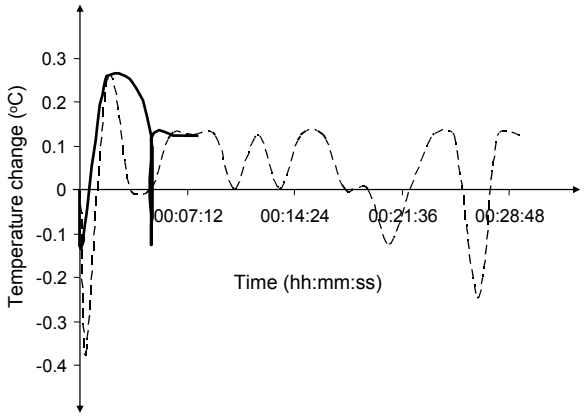

Figure 5.75: Temperature variation using Java 1.3 and J9 for buffer size of 16 KB

Table 5.13: Temperature variation for Familiar Linux

Average temperature variation for Familiar Linux (°C)		
	Java 1.3	J9
1 KB	0.187	0.303
2 KB	0	0.140
4 KB	0.104	0.071
8 KB	0.085	0.046
16 KB	0.034	0.083

5.2.16 Evaluation of Familiar Linux results

Java 1.3 from Blackdown has been designed specifically for handheld devices running Familiar Linux OS. J9 from IBM has been designed specifically for the Zaurus handhelds running Embedded Linux, and thus there is no official support for iPAQs running Familiar Linux. However, experimentation has shown that J9 is an optimised small-scale JVM that executes fairly well on Familiar Linux. In addition, as shown in Tables 5.9 - 5.13, J9 provided up to three times more throughput than Java 1.3, for various buffer sizes. Furthermore, J9 utilised the CPU on an average of approximately 21% less than Java 1.3 while routing. Also, the heap memory usage was up to three times less than Java 1.3. The temperature increase rate experienced by Java 1.3 was lower than J9, however, the total increase in temperature experienced by Java 1.3 was approximately double the amount of J9, as Java 1.3 required approximately three times more time than J9 to route the same amount of data. Finally, battery discharge rate seemed to have no effect on both JVMs, and was also almost equal across all tested buffer sizes.

These figures strongly suggest that J9 is an optimised JVM for handheld devices which is ideal due to its small footprint. In contrast, Java 1.3 requires more resources and provides less throughput. Nevertheless, it provides many more useful features, and can thus

execute a vast number of Java applications, without having to alter the application's source code in order to make it compatible with Sun's J2ME specification (*see* Appendix A).

Comparing throughput results presented for J9 on top of Familiar Linux to throughput results presented for J9 on top of PocketPC 2002 and 2003, there was a constant, significant improvement of the available throughput in the degree of 1164% and 826%, respectively, for all buffer sizes. Finally, it has been proven that buffer size does not have any significant importance in throughput, battery consumption rate, CPU utilisation, heap memory usage, or temperature variation.

5.2.17 Battery discharge rate of an idle PDA

This section presents results obtained by measuring the PDA's battery discharge rate while it was in an idle state. Measurements were taken for PocketPC 2002, PocketPC 2003, and Familiar Linux. The aim is to compare the power management of these OSs. Figure 5.76 presents the time taken for the battery to discharge by 1% for the PocketPC 2002, PocketPC 2003 and Familiar Linux, which is drawn against the total time taken for the battery to fully discharge. However, in PocketPC platforms data are stored in RAM, which provides requires minimal power to maintain them, thus if the battery is fully discharged it is likely that the information stored will get lost. Thus, to avoid this from happening, measurements were taken until the battery reached 25% of its capacity and then the experiment was stopped. In addition, all standby modes and power saving features were turned off. PocketPC 2002 is represented by a solid line, PocketPC 2003 by a dashed line, and Familiar Linux by a dotted line.

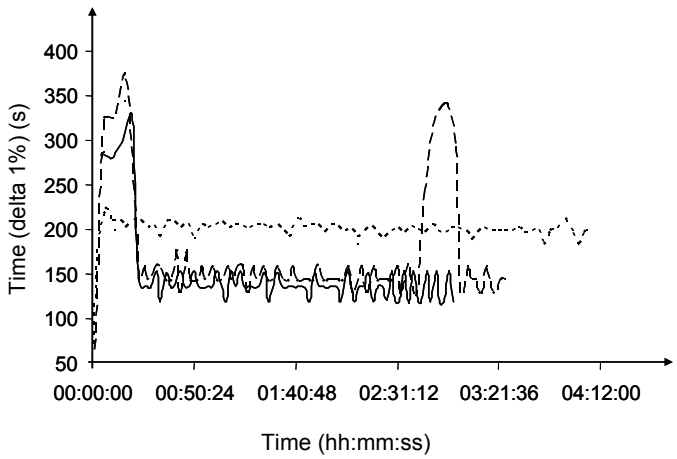

Figure 5.76:
Battery discharge rate of an idle PDA with PocketPC 2002, PocketPC 2003, and Familiar Linux

According to Figure 5.76, it can be clearly seen that Familiar Linux achieved the best results, followed by Pocket PC 2003, and, finally, PocketPC 2002. In more detail, the

average discharge rate for PocketPC 2002 was 143s, for PocketPC 2003 was 163s, and for Familiar Linux was 195s. Thus, the battery can last the longest with Familiar Linux, and longer with PocketPC 2003 than PocketPC 2002. In fact, according to these results, the battery with Pocket PC 2003 could last up to 33 minutes more than PocketPC 2002, and with Familiar Linux 1 hour and 27 minutes more. Thus, it is safe to conclude that Familiar Linux, applies better power management and thus the battery is discharged at a lower rate.

5.2.18 Battery discharge rate of an idle wireless PDA

This series of experiments is similar to experiments described in Section 5.2.17, with the only difference being that the PDA's wireless features were turned on during the experiment. The aim was to compare the ability of PocketPC 2002, PocketPC 2003, and Familiar Linux to maintain a low discharge rate while the wireless features are turned on. The PDA was in an idle state without sending or accepting network traffic. Figure 5.77, presents the time taken for the battery to discharge by 1% for the PocketPC 2002, PocketPC 2003 and Familiar Linux. As mentioned in the previous section, measurements were taken until the battery reached 25% of its capacity and then the experiment was stopped. Similarly, all standby modes and power saving features were turned off. PocketPC 2002 is represented by a solid line, PocketPC 2003 by a dashed line, and Familiar Linux by a dotted line.

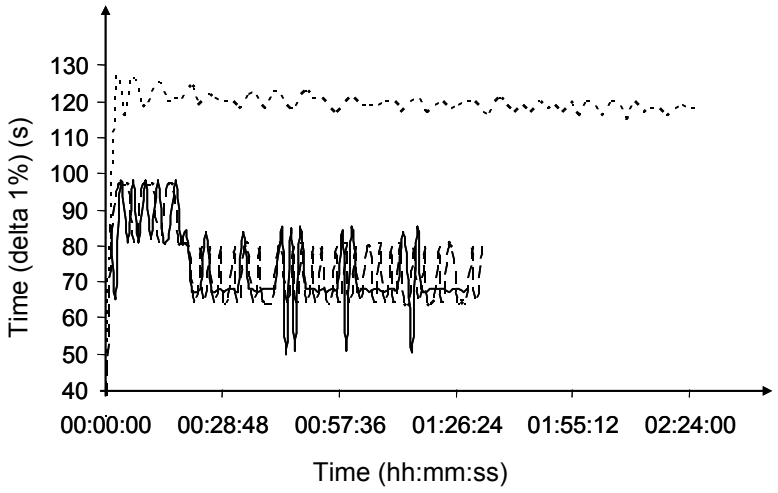

Figure 5.77:
Battery discharge rate of an idle wireless PDA with PocketPC 2002, PocketPC 2003, and Familiar Linux

According to Figure 5.77, PocketPC 2002 and PocketPC 2003 perform equally well, while Familiar Linux performs significantly better. In more detail, the average discharge rate for PocketPC 2002 was 71s, for PocketPC 2003 was 74s, and for Familiar Linux was 115s. Thus, the battery can last the longest with Familiar Linux, and approximately the

same with PocketPC 2002 and PocketPC 2003. In fact, according to these results, the battery with Pocket PC 2003 could last up to five minutes more than PocketPC 2002, and with Familiar Linux up to 1 hour and 13 minutes. Thus, it is safe to conclude that Familiar Linux applies better power management when the wireless is turned on and thus the battery is discharged at a significantly lower rate. Table 5.14 summarises the results for the idle PDA with and without its wireless on.

Table 5.14: Average battery discharge rate of an idle PDA with and without its wireless turned on

Average battery discharge rate for each supported OS Average time taken to change 1%		
	Idle (s)	Wireless (s)
PocketPC 2002	143	71
PocketPC 2003	163	74
Familiar Linux	195	115

5.2.19 Outcomes and evaluation of the proxy experimentation

This proxy experimentation phase presented extensive experimentation with an iPAQ proxy-based PDA in relation to throughput, battery discharge rate, CPU utilisation, heap memory usage, and temperature variation. In particular, the throughput experimentation revealed the following significant conclusions:

- The JVM can play a dramatic role in terms of the throughput offered by the routing device. In fact, CrEme offered 15 times more throughput than J9 when running on top of PocketPC 2003.
- The OS can also play an important role in terms of the throughput offered by a routing device. As an example, CrEme running on top of PocketPC 2003 can offer up to five times more throughput than running on top of PocketPC 2002. Similarly, J9 running on top of Familiar Linux can offer up to 10 times more throughput than running on top of PocketPC 2003.
- Buffer size does not have a significant role in the throughput offered by the routing device. This fact has been throughout the experimentation process.
- The best combination of OS and JVM in terms of throughput is PocketPC 2003 with CrEme, while the most unfavourable combination is PocketPC 2002 with either Jeode or J9.

Significant conclusions involving the battery discharge rate, which where revealed throughout the experimentation process are:

- OS's power management capability can assist in providing a longer battery life to

handhelds. For instance, when the PDA is idle, Familiar Linux allows an average discharge rate of 195s, while for the PocketPC 2002 it is 143s. This can be interpreted as 1 hour and 27 minutes longer battery life for the PDA running Familiar Linux instead of PocketPC 2002. The same can be observed when the wireless features are turned on. Familiar Linux offers 1 hour and 13 minutes longer battery life to a wireless handheld in comparison to PocketPC 2002.

- Routing reduces the battery capacity. Comparing the average discharge rates while routing to rates while the wireless features are turned on, it can be clearly seen that, in most cases, the difference is not significantly large. For instance, consider CrEme running on top of PocketPC 2003, which provides the highest throughput. On average the battery capacity of the routing PDA would get exhausted 23 minutes faster than the PDA which is idle but has its wireless features turned on. However, once compared with the PDA which is in an IDLE state without the wireless turned on, significant differences can be observed. In fact, battery capacity of the routing PDA would get exhausted two hours and 35 minutes faster than the idle PDA that has its wireless features turned off. In that respect, routing is a heavy process due to the requirement that the wireless must be switched on, and also that ad-hoc routing, itself, adds an extra overhead.
- Buffer size does not play any significant role in the PDA's battery discharge rate. This fact is proven throughout the experimentation process.
- The most efficient combination in terms of battery capacity is J9 running on top of Familiar Linux, which achieves an average discharge rate of 82s. The most unfavourable combination is Jeode or J9 running on top of PocketPC 2002, which achieves a discharge rate of approximately 62s. This can be interpreted as 30 minutes of more routing time by J9. In addition, J9 running on top of Familiar Linux can finish a routing task up to 10 times faster than Jeode running on top of PocketPC 2002 due to its higher throughput. Thus, it can save battery life by completing the task faster and also forcing a lower discharge rate to the battery's capacity.

The CPU utilisation conclusions obtained throughout the experimentation process are:

- Routing is a CPU intensive task. In all cases, the CPU utilisation ranges from a minimum point of 59% to a maximum point of 86.68%. The over utilisation of the CPU while constantly routing heavy network traffic may prohibit handhelds from performing other concurrent tasks efficiently.
- The JVM has a significant effect on the CPU utilisation of the handheld while performing a routing task. As an example, consider the utilisation of J9 running on top of PocketPC 2002 and the utilisation of CrEme running on top of the same OS. The

average utilisation of J9 is 64%, while CrEme's is 79%. Thus, CrEme requires more CPU time while routing than Jeode, however its broader throughput compensates for it.

- Buffer size does not play any significant role in the CPU utilisation of the PDA. This fact is proven throughout the experimentation process.
- The least CPU intensive combination is J9 running on top of PocketPC 2002. However, it provides the lowest throughput. A slightly more CPU-intensive combination is J9 running on top of Familiar Linux that utilises the CPU by 3% more, however it provides up to 10 times more throughput. This may be considered as evidence that the OS has a significant effect on the CPU utilisation, or that J9 for Linux is highly optimised compared to the implementation of J9 for PocketPC 2002.

The following briefly presents the conclusions obtained throughout this chapter concerning heap memory usage:

- The magnitude of heap memory used by Java *Objects* is linked to the JVM used. For instance, CrEme running on PocketPC 2002 allows its *Objects* to use up to three times more heap than Jeode and J9 running on PocketPC 2002. Another example is Java 1.3 and J9 running on top of Familiar Linux. Java 1.3 *Objects* can use up to three times more heap memory than J9 *Objects*.
- The magnitude of heap memory usage is related to the throughput offered by the JVM. For instance, CrEme running on top of PocketPC 2002 which allows its *Objects* to use three times more heap memory than Jeode and J9, can achieve up to four times more throughput. Another example is CrEme running on top of PocketPC 2003 which allows its *Objects* to use 10 times more heap memory than J9, and can achieve up to 15 times more throughput. The only exception to this rule is Java 1.3, which is based on Sun's Java 1.3 specification, and thus has a large footprint, which is probably why it uses large amounts of heap, while achieving lower throughput than J9, which uses much less (*see* Section 5.2.14).
- Buffer size does not play any significant role in the heap memory usage. This fact is proven throughout the experimentation process.

Relative, temperature variation readings were much less compared to throughput, battery discharge rate, and so on, due to the fact that these figures were recorded for only the Familiar Linux platform. Conclusions that may be drawn from these figures include the following:

- J9 increased the proxy device's internal temperature at a higher rate than Java 1.3,

which may be due to the fact that J9 achieved higher throughput, and thus utilised its wireless interface more intensively than Java 1.3, which may have caused this difference.

- The limitation of this experiment is that the execution time for J9 and Java 1.3 was different, as J9 finished first due to its broader throughput, and thus the difference in the overall temperature increase may be partially associated with the more time taken by Java 1.3 to conduct the experiment.

In addition, the limitation of the experiments lie on the fact that, resource-consumption measurements, were not taken at a constant throughput. Thus, it is possibly unfair for devices that achieved good throughput, and consequently consumed more in terms of battery, CPU utilisation, and heap memory usage. In this way, a possible more accurate future experiment would be allow a constant throughput to flow through each device, and measure these resource-consumptions.

5.3 BASS Experimentation

Results presented in the previous chapters clearly demonstrated that the fitness of various routing devices significantly varies, and may be related to a wide range of parameters, such as processing power, buffering capability, memory capacity, utilisation status, and furthermore the OS and JVM combination used to execute the routing software. This section investigates the effect of these parameters in-depth, by presenting performance results acquired by running a set of tests in a number of device types (*see* Appendix A), ranging from a low-performance device, such as a PDA, to a powerful workstation. The software used for conducting these tests was BASS, which is fully described in the Sections 3.10 - 3.12 and 4.4. Briefly, BASS is a Java-based multi-agent performance acquisition system, especially designed for resource-constrained devices, which can be also executed in high-end devices, and has the ability to perform various tests and apply a preliminary metric for each test which collectively represent the strength of a routing device. These tests are grouped into preliminary and continuous, since the first involves tests that may be run only once, while the latter requires tests to continue monitor system's resources. Preliminary tests include the group-level, bubble sort, CPU merge, memory test, client-server and proxy throughput, and Internet connectivity, while continuous tests include the error packets monitoring, CPU, memory, and overall utilisation, Java threads monitoring, and battery monitoring.

The purpose of this cycle of experiments was to benchmark the strength of various device types as routing elements, ranging from low-to-high performance. The aims were:

- Identify the strengths and weaknesses of devices with different hardware characteristics, for the purpose of ad-hoc routing.
- Produce a preliminary metric which corresponds to the fitness of a device to carry-out a certain task, related to ad-hoc routing.
- Investigate the need for metric-driven routing in ad-hoc networks.
- Analyse and compare these preliminary metrics in relation to their importance in ad-hoc routing.
- Determine the tests that best fit into the derivation of an overall metric of a routing device.

5.3.1 Group-level agent test results

This test is aiming to extract system information, such as the IP address of the device, the OS name, architecture and version, the JRE version and supported classes, and so on. Table 5.15 presents the system information gathered by having run this test to the laptop device (*see* Appendix A). The importance of this test lies in the ability of the routing protocol to issue routing updates to a group of devices, without affecting the remaining devices in the network. For example, a mobile agent carrying a routing updated targeted for devices running Familiar Linux, could install the update based on system information found at each node, and thus avoid other devices running OSs, such as PocketPC.

Table 5.15: Sample data captured by the group-level test, executed on the laptop

Group information	Sample data
MainIP	192.168.0.11
MainHostname	Laptop
AltIP	127.0.0.1
AltHostname	localhost
OSName	Windows NT
OSArch	x86
OSVersion	5.0
JRuntimeName	NULL
JVMVersion	1.2
Username	me
UserCountry	NULL
UserLang	en
JavaVMRuntime	Java Virtual Machine Specification
JavaRuntimeVers	NULL
JavaVersion	1.2
Graphics	sun.awt.Win32GraphicsEnvironment
SunArcData	NULL
SunCPUEndian	NULL
SunUnicode	UnicodeLittle
SunCPUISA	NULL
Last Updated	12/14/03 14:09

5.3.2 Bubble sort agent test results

The aim of this test was to benchmark the processing power of a device, by performing a bubble sort test of a large amount of random integers. In particular, the bubble sort agent was requested to sort 30,000 random integers based on four different levels of intensity, which were: one, two, three, and four-dimensional array. Figures 5.78 - 5.81 presents the time taken for each device to complete the test, based on 20 iterations, and on multiple degree of intensity.

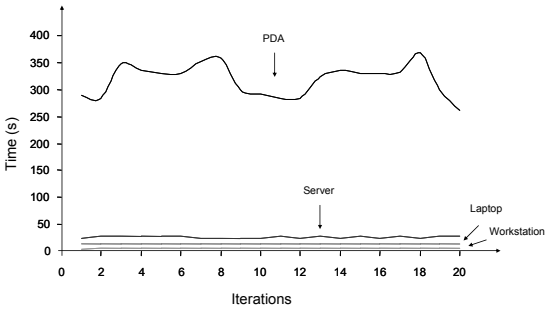

Figure 5.78: Time taken to execute the bubble sort test (1D) for the PDA, server, laptop, and workstation, for each one of the 20 iterations

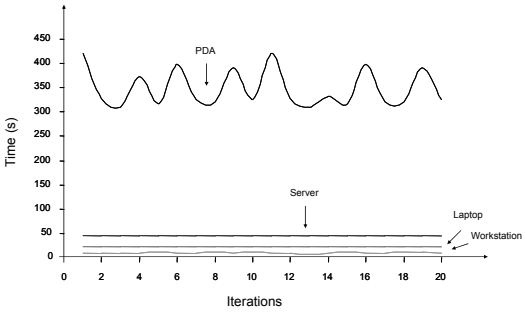

Figure 5.79: Time taken to execute bubble sort test results (2D) for the PDA, server, laptop, and workstation, for each one of the 20 iterations

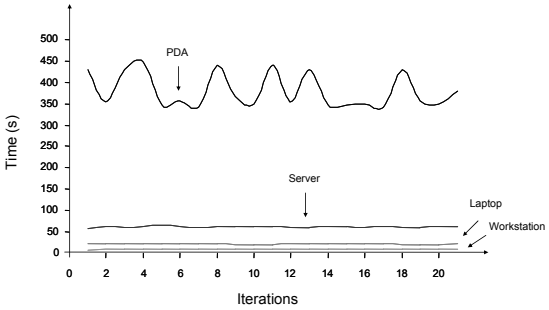

Figure 5.80: Time taken to execute the bubble sort test (3D) for the PDA, server, laptop, and workstation, for each one of the 20 iterations

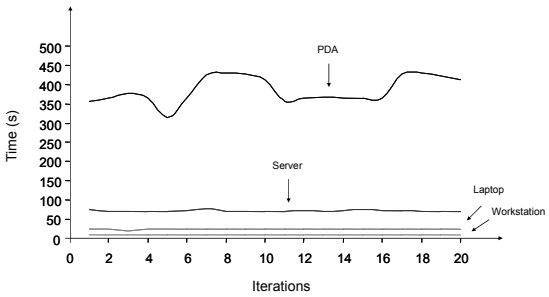

Figure 5.81: Time taken to execute the bubble sort test (4D) for the PDA, server, laptop, and workstation, for each of the 20 iterations

According to Figures 5.78 - 5.81, it can be observed that the PDA was above 50 times slower than the workstation, in most cases, and approximately 20 times slower than the laptop. In addition, all devices performed better when fewer dimensions were used, even though the amount of integers required by the devices to sort was approximately the same for all algorithmic-depths. In particular, as shown in Section 4.4.1, the 4D bubble sort algorithm was requested to sort less numbers than the 3D, and so on, which is probably why the difference of the tests is not clearly visible. However, this experiment shows that as the complexity of a sorting algorithm increases, the execution speed decreases, and is very similar for all device types. In more detail:

- The PDA required 9.54% more time to execute the 2D algorithm than it needed for the 1D, 19.37% more time for the 3D, and 21% more time for the 4D.
- The laptop required 59.46% more time for the 2D, 56.24% more time for the 3D, and 83.42% more time for the 4D.
- The server required 70.43% more time for the 2D, 136.53% more time for the 3D, and 175.53% more time for the 4D.
- Workstation required 78.57% more time for the 2D, 58.32% more time for the 3D, and 67.85% more time for the 4D.

Table 5.16 summarises the average time taken for each device to complete the bubble sort test with increased intensity. It can be seen that the workstation achieved the worst time in the 2D test, which may be a result of some internal processing that was required by the OS, and was hidden at the level of the test. Figures 5.82 - 5.83 present the time taken for the PDA's battery to discharge by 1% throughout the bubble sort experiment with 1D complexity and 3D respectively. The battery measurements taken during the tests were continuous. In particular, at the beginning of each test the battery was fully charged, and when the battery dropped below 10% the test was stopped.

Table 5.16: Average values for the Bubble sort experiment with in depth complexities

Bubble sort test depth	PDA time (s)	Laptop time (s)	Server time (s)	Workstation time (s)
1D	318.32	13.94	26.11	5.60
2D	348.71	22.23	44.50	10.01
3D	379.99	21.78	61.76	8.90
4D	385.18	25.57	71.91	9.41

A couple of insignificant anomalies can be observed in Figure 5.83, where the battery discharged by 1% in approximately 20s. However, the battery discharge rates, in both tests were normally in the range of 80s to 120s. The discharge rate was, on average, very similar in both tests, and in particular 84s per 1% battery drop for the 1D bubble sort and 83s per 1% battery drop for the 3D. This suggests that although the battery was shown to discharge at a higher rate while the resource-constrained device was performing routing tasks, which is clearly a CPU-intensive task (*see* Section 5.2), the battery does not necessarily discharge at a higher rate while the device is performing a less intensive algorithm, such as the 3D bubble sort.

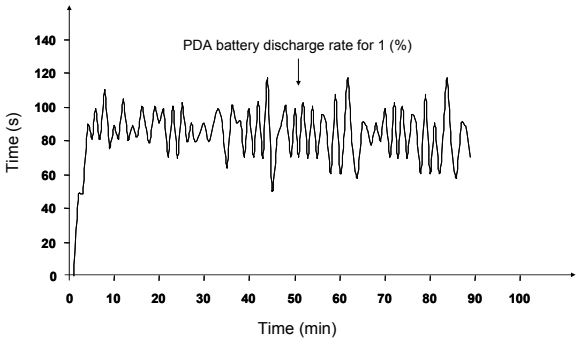

Figure 5.82: Battery discharge rate for the PDA executing the 1D bubble sort test

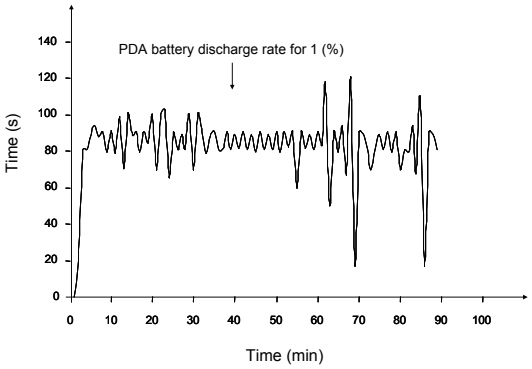

Figure 5.83: Battery discharge rate for the PDA executing the 3D bubble sort test

5.3.3 Memory test results

Figures 5.84 - 5.87 present the results gathered by each device executing the creation of one and 16 files with constant file-sizes, and the creation of a constant number of files with 1KB and 16KB file-sizes tests. Specifically, results show the time taken by each device for each of the total 20 iterations. The memory tests could reveal insight information concerning the buffering capabilities of devices that use their RAM as a persistent storage, whereas for devices that use a hard-drive as a persistent storage, the test is probably limited, as the speed of accessing and storing data to the hard drive does not relate to ad-hoc routing. Thus, test result for devices equipped with a hard-drive, such as the workstation and laptop (*see* Appendix A), are presented here only for comparison reasons.

Results show the differences in the performance capabilities of each device in relation to their specifications. The difference between the PDA and the other devices is, however, not nearly as great as in the bubble sort tests. For example, for the test of creating 16 files per size the PDA is only, on average 12.68 times slower than the workstation. This could be mainly due to the fact that the PDA uses electronic memory to store its files, which has a far quicker access time than a hard drive. This pattern is slightly different for the second test, which demands more processing power from the device, as it creates more files. In the case of the second test, the PDA is, on average, 30 times slower than the workstation. A possible reason for this is that the PDA runs low on memory as it creates the files in its RAM. For example, for the PDA to create 2000 files of 16KB, it requires nearly 32MB of its memory. This could impair performance as it has only a total of 64MB, which almost half of it is occupied by the OS. Thus, the PDA should not be used to buffer large amounts of information, which is often caused when a device is frequently used for ad-hoc routing.

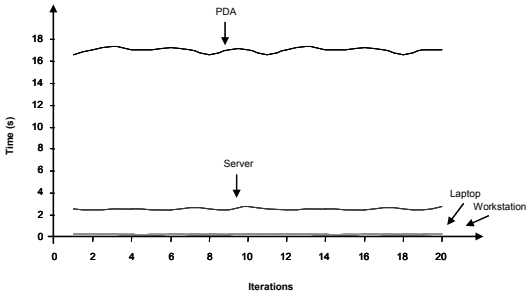

Figure 5.84: Time taken to create one file of varied sizes for each one of the 20 iterations

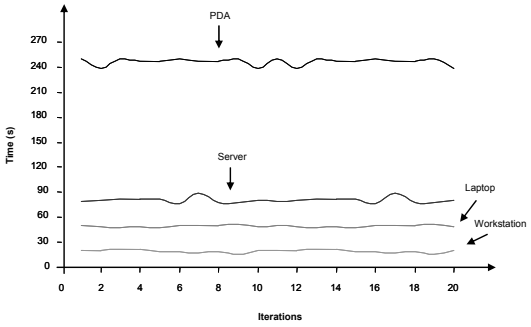

Figure 5.85: Time taken to create 16 files of varied sizes for each one of the 20 iterations

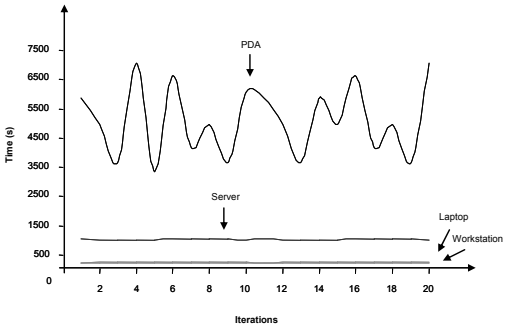

Figure 5.86: Time taken to create varied number of files of 1KB size, for each one of the 20 iterations

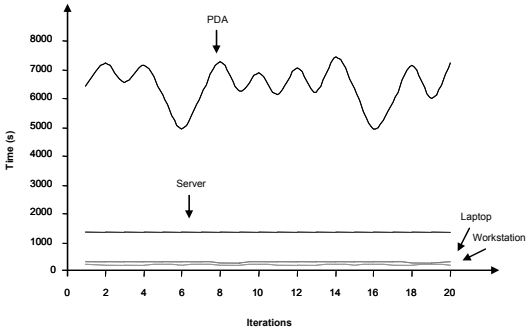

Figure 5.87: Time taken to create varied number of files of 16KB size, for each one of the 20 iterations

Figures 5.88 - 5.91 present the time taken for the PDA to discharge by 1%, while conducting the memory tests. Figures 5.88 and 5.89 present the results obtained during a continuous file creation test, which creates fixed files of a varied sizes. Figures 5.90 and 5.91 present the results obtained during a continuous file creation test, which instead, creates varied number of files of fixed sizes.

The same data size of 21,000KB was used in the following tests: fixed number of files (one file) of varied sizes; and varied number of files of fixed sizes (1KB). Similarly, a total of 336,000KB was used for the second set, respectively. In the first case, the battery discharge was approximately equal to 119s and 114s, while, in the second case, the discharge was 93s and 62s, respectively. This shows that the discharge of the battery was raised by 33.33% when more files of fixed sizes were created, even though the same total amount of data was used. Overall, the experiments prove that resource-constrained devices consume increased amounts of energy, when they constantly store large amounts of information to their memory, which may be related to the buffering process while routing. Thus, a resource-constrained device should not be regularly used to route heavy network traffic because of its limited buffering speed, and the increased battery consumption which is required by this process.

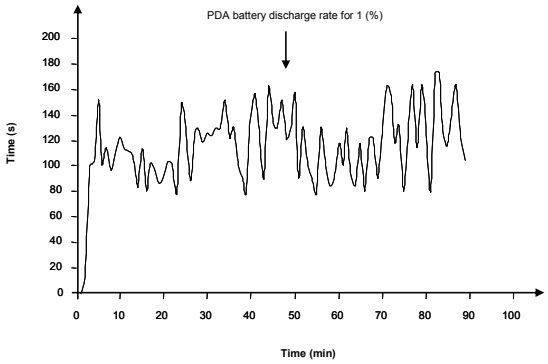

Figure 5.88: Time taken for the battery to discharge by 1% for 1 file of varied sizes

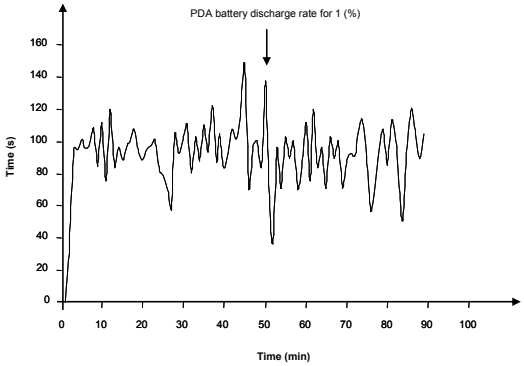

Figure 5.89: Time taken for the battery to discharge by 1% for varied number of files of 1KB sizes

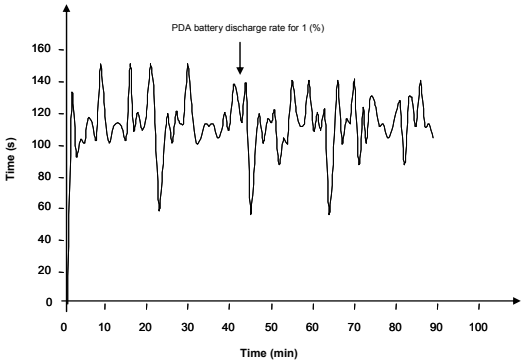

Figure 5.90: Time taken for the battery to discharge by 1% for 16 files of varied sizes

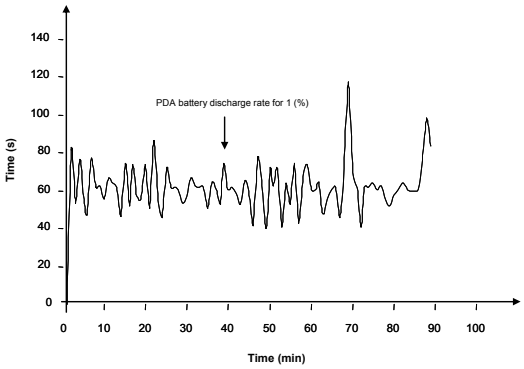

Figure 5.91: Time taken for the battery to discharge by 1% for varied number of files of 1KB sizes

5.3.4 CPU merge test results

As previously mentioned, this test is provided as an alternative to the bubble sort test, as it less CPU-intensive, and can thus, possibly, be executed by limited devices, such as mobile phones. Figure 5.92 illustrates the results from running the CPU merge test on all devices. In this, the PDA achieved the worst results, as it was 11 times slower than the server, 45 times slower than the laptop, and 103 times slower than the workstation. In addition to results obtained by bubble sort test execution, merge sort results further indicate the difference in processing power between PDAs and other higher-end devices. Although 80,000 integers were used for sorting by the merge sort algorithm, which is 50,000 more than the ones used in the bubble sort test, the time taken was significantly less. This is because merge sorting is using a less intense algorithm. Figure 5.93 illustrates the time taken for the PDA's battery to discharge by 1%, while executing the merge sort test, in a continuous manner. The average discharge time was 94s, which when compared with Figure 5.82, it can be seen that the battery discharge rate in this experiment was al-

most 10% less than the respective 1D bubble sort. This further suggests that less CPU-intensive algorithms cause the battery to discharge at a lower rate.

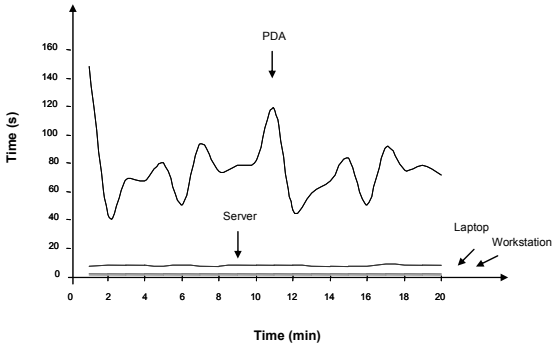

Figure 5.92: Time taken to sort 80,000 integers using the CPU merge sort agent for all devices

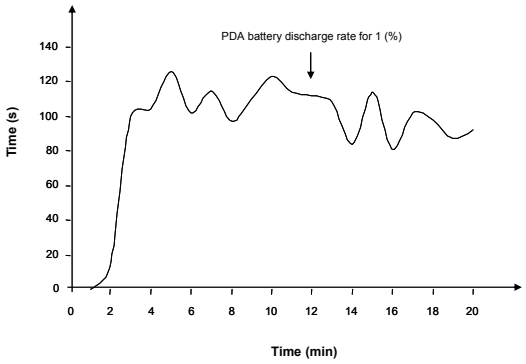

Figure 5.93: Time taken for the PDA's battery to discharge by 1 (%) while conducting the merge sort test for 80,000 integers

5.3.5 Internet connectivity test results

As previously mentioned, results from this test do not participate in the metric calculation process, as the test is not related to ad-hoc routing (*see* Section 3.11). The details of this experiment can be found in Section 4.4.4. The results from running the Internet connectivity test in each device are illustrated in Figures 5.94 - 5.96. The results show consistent connection, download, and total times for all devices apart from the PDA. Table 5.17 presents the average, connection, download and total times per device. It can be seen that the time taken for the PDA to conduct the experiment is nearly 50 times more than the workstation and laptop.

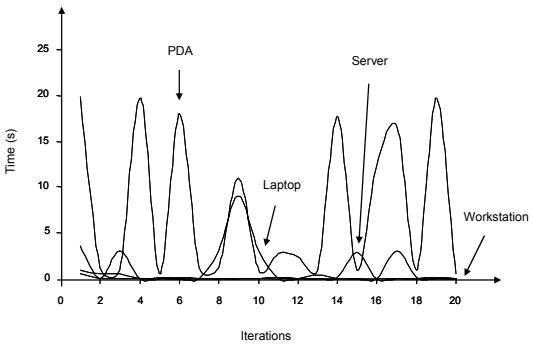

Figure 5.94: Remote page connection times

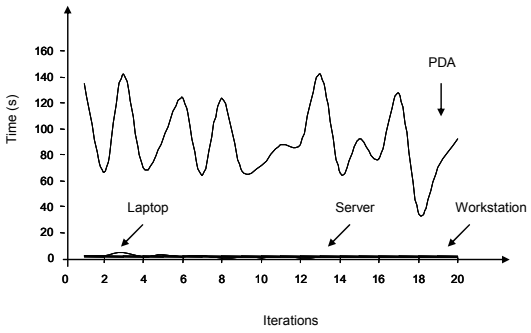

Figure 5.95: Remote page download times

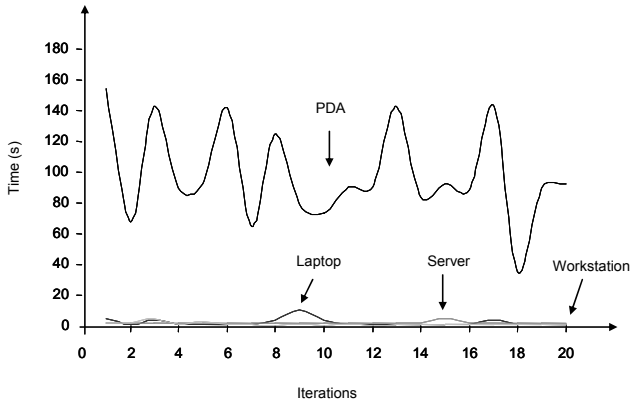

Figure 5.96: Remote page total times

Table 5.17: Average connection, download and total times for each device

Internet Connectivity test	PDA	Laptop	Server
Connect (s)	7.46	1.27	0.22
Download (s)	91.47	1.18	1.63
Total (s)	98.93	2.46	1.85

5.3.6 Summary of battery power consumed by each test

Table 5.18 summarises the total battery power consumed by each test, over an average of 20 iterations, as described in previous sections. It highlights various patterns in respect to the loss of battery power. For example, it can be clearly seen from the results of the bubble sort test that, as the intensity of the processing task is increased, the amount of battery reserves consumed is also increased. In addition, the creation of large number of files has a severe effect on the reduction of the battery capacity, whereas, for the same amount of data, the creation of larger files can significantly reduce the battery consumption. Result such as these, were used to identify the issues involved with ad-hoc routing devices, and were further used to model the devices in terms of their routing ability and resource-consumption rates.

Table 5.18: Average battery consumption during each of continuous tests

Test	Battery used (%)
1D Bubble	3.00
2D Bubble	3.20
3D Bubble	3.40
4D Bubble	5.00
File test 1 - 1KB	0.67
File test 1 - 16KB	2.90
File test 2 - 1KB	27.50
File test 2 - 16KB	34.40
Merge Sort	0.55
Download HTML file	0.79

5.3.7 Outcomes and evaluation of the BASS experimentation

This experimentation phase presented the Benchmarking multi-Agent Software System (BASS), which is light-weighted and can execute on resource-constrained devices, as well as high-end devices. Its purpose is to benchmark the routing fitness of ad-hoc devices by conducting a number of preliminary and continuous tests. Preliminary tests include: the group-level; bubble sort; memory test; CPU merge; Internet connectivity; client-server and proxy throughput; while, continuous tests include: the error packets monitoring; CPU and memory utilisation; Java threads monitoring; and battery reserves monitoring. As previously mentioned, test results from the group-level, the Internet connectivity, and the Java threads monitoring agents do no participate in the metric calculation process due to their lack of relevance with ad-hoc routing. Preliminary tests are required to execute only once, while continuous tests constantly execute in the background. Preliminary tests were shown to require large resources for their execution, and should thus be executed when the device is idle. On the other hand, continuous tests require fewer resources, even when they are executed on handheld devices. Preliminary and continuous test results are used by MARIAN in order to produce a metric representing the device's routing capability, which is then tailored to various routing scenarios.

5.4 Mobile agent migration

This chapter presents results regarding the ability of agents to migrate over the wireless medium. The first phase of the experimentation attempted to benchmark the average time required for an agent migration and provided evidence which suggested that migration time is related to the hardware characteristics of the devices involved. The second phase investigated the ability of agents to migrate to and from handheld devices, and further appraised the general assumption which suggests that mobile agents can reduce network load, under certain circumstances.

The experimentation cycle of mobile agent migration was conducted with purpose to: benchmark the migration requirements of mobile agents; investigate the effect of hardware performance on migration times; and evaluate the assumption that mobile agents can reduce network load when compared to static approaches. In particular, the aims are:

- Benchmark the average time required by two successive hosts to dispatch, and receive a mobile agent which carries no data.
- Investigate the effect of varying hardware capacity on mobile agent migration times.
- Prove the feasibility of agent migration through a series of wireless handhelds.
- Determine the circumstances under which mobile agent technology can reduce network load compared to static approaches.

5.4.1 Mobile agent migration time requirements

As previously mentioned, this experimentation cycle is aiming to benchmark the time requirements for an agent migration, and investigate whether agent migration is related to the hardware characteristics of devices involved. When a Java-based mobile agent requests its migration, the underlying mobile agent system initially performs the necessary preparations, then serialises the agent, and transmits it to the next hop. Table 5.19 presents the average migration times achieved by the runner agent (*see* Appendix A), which travelled through a series of superior devices (*see* Appendix A). Table 5.20 presents similar results, however, in this case the agent travelled through a series of inferior devices (*see* Appendix A). Figure 5.97 illustrates the runner agent's round trip time (RTT) values, and compares the superior and inferior results.

Table 5.19: The average migration times and RTT values for the superior group

Superiors		
Hop count	*Migration time*	*RTT (s)*
1	1.047	1.047
2	1.25	2.297
3	1.031	3.328
4	1.047	4.375
5	0.985	5.360
6	1.015	6.375
7	0.976	7.351
8	1.102	8.453

Table 5.20: The average migration times and RTT values for the inferior group

Inferiors		
Hop count	*Migration time*	*RTT (s)*
1	1.297	1.297
2	1.109	2.406
3	1.187	3.593
4	0.97	4.563
5	1.14	5.703
6	0.469	6.172
7	1.593	7.765
8	1.078	8.843

According to Tables 5.19 - 5.20, it can be seen that the agent's average migration time is approximately 1.057s for the superiors and, approximately, 1.105s for the inferiors. The RTT is approximately 8.453s for the superiors and, approximately, 8.843s for the inferiors. In relation to these figures, it can be clearly seen that the hardware characteristics of a device can influence an agent's migration time, and, in particular, low performance devices can delay the agent's migration compared to high performance devices. Even

though these figures do not provide an absolute distinction, they indicate that hardware performance is a factor. Specifically, the average agent's migration was further delayed by approximately 49ms, while the total RTT of the agent was further delayed by approximately 392ms.

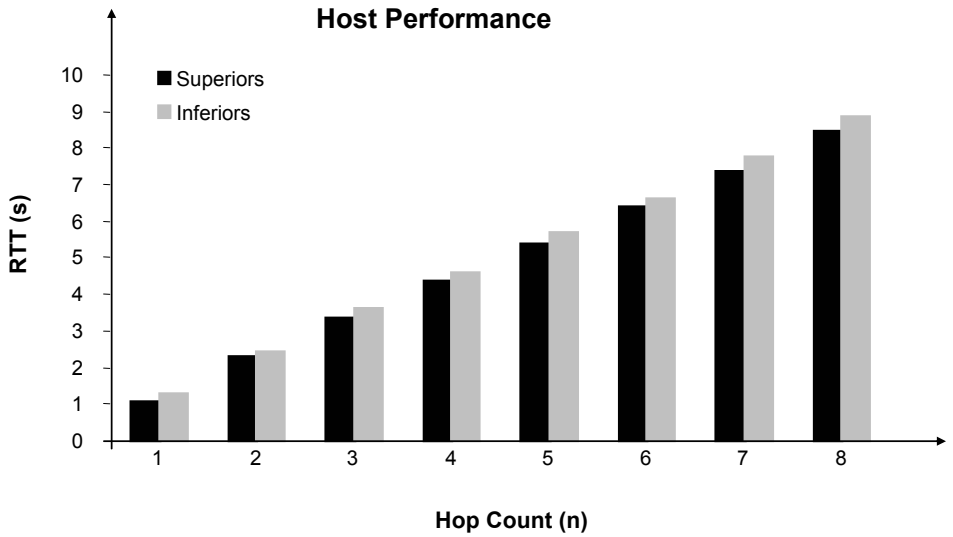

Figure 5.97: The RTT values for the superior and inferior groups

5.4.2 Data gathering based on static and mobile agents

As previously mentioned, a frequently proposed advantage of mobile agent technology is its ability to considerably reduce network load in comparison to standard static approaches. This experimentation phase aims to investigate this and provide definite conclusions on under which circumstances this is possible. For this purpose, a data retrieval application scenario has been designed, as described in section 4.5, and used to compare both static agent and mobile agent approaches.

Figures 5.98 - 5.99 present the time required by the static agent and mobile agent approaches, respectively, in order to complete the data gathering process for data sizes of 100Kbits, 200Kbits, and 300Kbits. The average time required, over 20 iterations, for the static agent approach is approximately 4.82s for 100Kbits, 4.85s for 200Kbits, and 4.98s for 300Kbits, while for the mobile agent approach it is approximately 44.18s for 100Kbits, 51.17s for 200Kbits, and 64.25s for 300Kbits. Therefore, the increase in the size of returned hits increases the time taken to complete, for each approach, as expected. The static agent approach has handled the size increases efficiently as the time incremented by a negligible amount, specifically of the scale of 30ms for additional 100Kbits, and 160ms for additional 200Kbits. In contrast, the mobile agent approach required considerably more time for successive data size increases. Specifically, it required 7s additional for an increase of 100Kbits and 20s additional for an increase of 200Kbits.

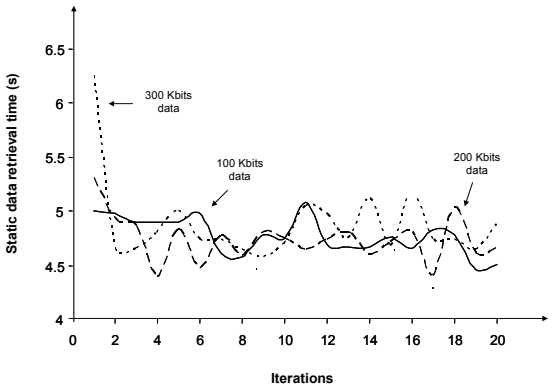

Figure 5.98: Data gathering based on the static agent approach for 100Kbits, 200Kbits, and 300Kbits

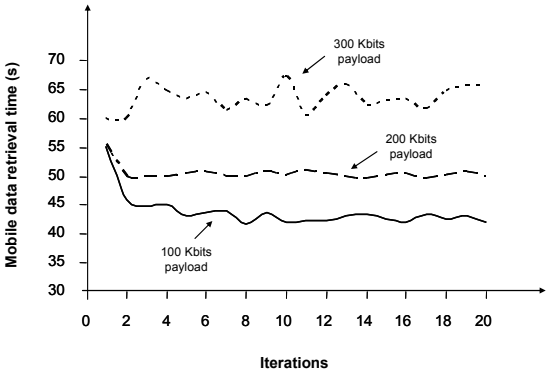

Figure 5.99: Data gathering based on the mobile agent approach for 100Kbits, 200Kbits, and 300Kbits

Figure 5.100 compares the average time required by both the static and mobile agent approaches. Accordingly, the mobile agent approach was 10 times more time consuming than the respective static agent approach. This considerable difference may be attributed to the JVM and the agent platform's migration component, as they are responsible for capturing the agent's state, and serialising the agent, before its actual transmission to the next hop. Therefore, as it can be clearly seen from this particular experiment, the mobile agent data gathering is significantly slower than its counterpart, and can thus not be efficiently used to replace static approaches.

Figure 5.101 illustrates the time required by the mobile agent with filtering approach to retrieve 15Mbits of data, and contrasts the results to the time taken to retrieve the same amount of data by the static agent approach. Results show that the mobile agent with filtering approach significantly improved the time required in relation to the static agent approach. Particularly, the average time required by the mobile agent with filtering approach was approximately 42s, while the average time required by the static agent approach was approximately 62s. Thus, the filtering approach was nearly one third faster than the static agent approach. This was due to the fact that the mobile agent retrieved the data from the database, which was initially of size 15Mbits, and performed data filtering locally based on its user's preferences, which, in this case, reduced the data from 15Mbits to only 56Kbits. Thus, the agent only stored 56Kbits of data in its payload. The static agent approach had no filtering capability, and thus retrieved the full amount of data. Therefore, mobile agent technology can be considered to be more efficient for data gathering than static approaches, under certain circumstances, such as, if mobile agents are capable of intelligent filtering, and that the data to be retrieved is relatively large.

The mobile filtering approach can also be also used in ad-hoc networking applications, such as in MARIAN's proactive network discovery process, where each topology

gathering mobile agent has the capability to filter and discard repeated information. In this way, the agents reduce the amount of information they carry, and thus decrease their migration time, and, consecutively, decrease the time taken by the proactive approach to collect the complete network's topology.

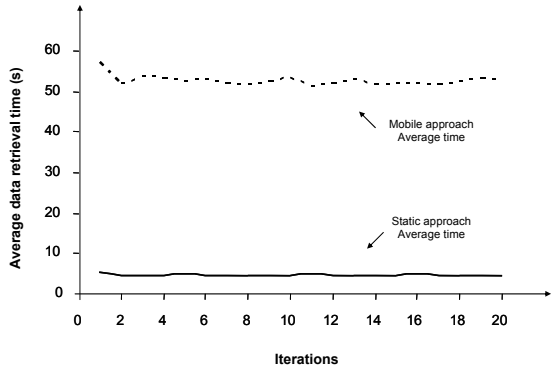

Figure 5.100: Average time taken for the static agent and mobile agent approaches

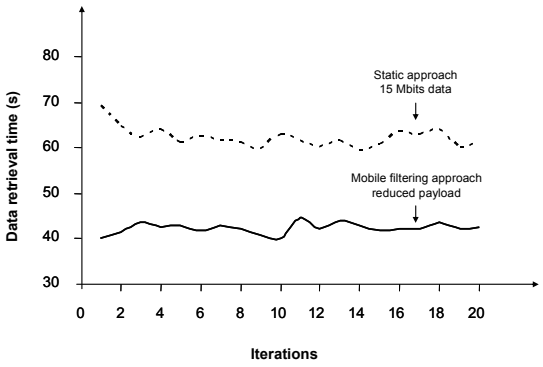

Figure 5.101: Data gathering based on the static agent and mobile agent with filtering approaches for 15Mbits

5.4.3 Outcomes and evaluation of mobile agent migration experimentation

Experimentation results presented throughout this section show that a Grasshopper mobile agent requires approximately 1s for its migration when the underlying hosts (transmitter and receiver) are composed of high performance hardware characteristics, such as the devices defined in Appendix A. However, in the case of handhelds, it can be estimated that this figure significantly increases, reaching up to five times more. This chapter also presented a data gathering application scenario, where the static and mobile agent approaches were compared in terms of the time taken to retrieve varying size of data. The static agent approach was proved to be the most efficient, at all times, apart from the case where the mobile agent was capable of intelligent filtering of data. In this case, the mobile agent approach improved on the static agent approach by nearly one third. The results of this experimentation phase contributed in designing MARIAN's proactive network discovery to allow mobile agent filtering. Therefore, each generation of propagating network discovery mobile agents is designed to filter network topology information in such as way so as to eliminate unnecessary redundant routing information.

5.5 Metrics simulation experimentation

This chapter demonstrates the assignment of routing metrics to both devices and retrieved routes, which are initially expressed in the form of capability/incapability, and,

then, in case of capability, to a meaningful expression, such as poor, good, and so on. As previously discussed in Section 3.13, a single routing metric is assigned for each defined routing scenario, and, thus, it is possible for a device to be determined capable of accomplishing a particular routing scenario, while incapable for another. Then, results obtained from a preliminary metric simulation are presented. Specifically, throughout the simulation, critical factors, such as battery capacity, memory and CPU utilisation, have being varied, and, consequently along with them, the corresponding preliminary metrics, in order to measure the effect that these variations have on the routing capability/incapability and final metric of a device.

The main aim is to demonstrate the accuracy of the metric calculation process, and its dynamics in rapidly responding to critical changes of a device's routing capabilities. In particular to:

- Demonstrate the routing metric calculation process.
- Prove the accuracy of the routing metric assignment process.
- Prove the ability of the design to respond to sudden changes of a device's vital elements.

5.5.1 Routing metric calculation

In order to demonstrate the routing metric calculation process, six distinct device types have been defined, as previously presented in section 4.6. The experimentation results presented in Sections 5.1 - 5.3 are being recapitulated, and further enhanced for the purposes of this section. Thus, in relation to data from Sections 5.1 - 5.3, Table 5.21 presents the test results, which have been achieved by each of the previously defined device types (*see* Section 4.6). Based on the mathematical equations defined in Section 3.13 and on test results presented on Table 5.21, the preliminary metrics for each device type can be calculated, as shown in Table 5.22.

Table 5.21: Test results for each device type

	DT_1	DT_2	DT_3	DT_4	DT_5	DT_6	DT_7	DT_8	DT_9
T_1 (s)	31.83	31.83	31.83	31.83	31.83	31.83	2.61	1.395	0.56
T_2 (s)	76.34	76.34	76.34	76.34	76.34	76.34	7.38	1.69	0.74
T_3 (s)	16.99	16.99	16.99	16.99	16.99	16.99	2.58	0.27	0.24
T_4 (s)	5313.7	5313.7	5313.7	5313.7	5313.7	5313.7	1127	279.3	196
T_5 (%)	0	0	0	0	70	0	0	0	0
T_6 (%)	0	0	0	0	70	0	0	0	0
T_7 (%)	20	20	20	20	85	20	0	0	0
T_8 (s)	52.544	52.544	52.544	52.544	52.544	52.544	44.52	38.104	28.32
T_9 (s)	80.5	80.5	14	299.985	80.5	80.5	5.25	2.625	1.75
T_{10} (%)	3	89	3	3	3	3	2	2	1
T_{11} (%)	56	56	56	56	56	56	50	40	30

T_{12} (%)	98	98	98	98	98	19	98	98	N/A

Once the preliminary metrics have been calculated, the next step involves their translation into an overall metric, specifically tailored for each predefined objective (*see* Section 3.13). Thus, each device calculates a single overall metric for each predefined objective, which is principally distinct and ranges between 0 and 100. Table 5.23 presents the outcomes of applying the overall metric calculation process to the predefined devices' preliminary metrics (*see* Table 5.22).

Table 5.22: Preliminary metrics derived from test results

	DT_1	**DT_2**	**DT_3**	**DT_4**	**DT_5**	**DT_6**	**DT_7**	**DT_8**	**DT_9**
pm_1	63.66	63.66	63.66	63.66	63.66	63.66	5.22	2.79	1.12
pm_2	76.34	76.34	76.34	76.34	76.34	76.34	7.38	1.69	0.74
pm_3	84.95	84.95	84.95	84.95	84.95	84.95	12.9	1.35	1.2
pm_4	75.91	75.91	75.91	75.91	75.91	75.91	16.1	3.99	2.8
pm_5	0	0	0	0	70	0	0	0	0
pm_6	0	0	0	0	70	0	0	0	0
pm_7	20	20	20	20	85	20	0	0	0
pm_8	65.68	65.68	65.68	65.68	65.68	65.68	55.65	47.63	35.4
pm_9	23	23	4	85.71	23	23	1.5	0.75	0.5
pm_{10}	1.9	92.435	1.9	1.9	1.9	1.9	1.54	1.6	1.33
pm_{11}	64	64	64	64	64	64	54.84	39.32	25.23
pm_{12}	0.31	0.31	0.31	0.31	0.31	80.52	0.31	0.31	0

The overall metric calculation is simple and should thus not over-utilise a device's resources irrespectively of its general processing strength, as a device is only required to perform a single mathematical calculation for each objective. The next step involves the determination of the device's capability/incapability of accomplishing certain routing scenarios. Based on the desired ranges of six previously defined objectives (*see* Section 3.13) and on the overall metrics presented in Table 5.23, the capability/incapability determination can be calculated, as shown in Table 5.24.

Table 5.23: Overall metric values for each device type and objectivity combination

	Energy	Synch	Asynch	Critical	Secure	Burst
Av. PDA (DT_1)	23.05	26.22	27.85	16.54	45.63	44.87
High CPU util. PDA (DT_2)	40.20	39.53	36.90	26.07	55.69	60.21
Good throughput PDA (DT_3)	22.33	22.73	27.85	16.14	44.93	44.54
Poor throughput PDA (DT_4)	25.43	37.75	27.85	17.86	47.95	45.93
High network errors PDA (DT_5)	27.71	30.74	48.35	48.90	53.22	48.34
Low battery PDA (DT_6)	53.43	49.81	48.72	33.42	54.54	58.46
Av. laptop (DT_7)	17.01	13.90	15.20	8.79	13.85	17.56
Strong laptop (DT_8)	12.45	9.97	10.98	6.32	8.85	10.12
Powerful workstation (DT_9)	8.15	6.76	7.00	4.14	5.70	6.67

According to Table 5.24, the average PDA is classified as capable of routing energy efficient network traffic, since its memory, CPU, and battery preliminary metrics are fairly low. However, it falls outside the limits of synchronous traffic requirements, as its network throughput was not within the required limits. For asynchronous traffic, an average PDA is sufficient to route data, as this type of traffic does not have any special requirements, apart from the battery metric which is always a key metric in all of the objectives. The average PDA is also capable of routing critical traffic, as its network protocol error rates are significantly low. Finally, it is classified as incapable of routing secure traffic or burst traffic, as it has low buffering capabilities and normally takes considerable time to perform intensive calculations.

Table 5.24:
Capability/incapability determination of 6 predefined device types for 6 predefined routing objectives

	O_1	O_2	O_3	O_4	O_5	O_6
DT_1	**Capable**	Incapable	**Capable**	**Capable**	Incapable	Incapable
DT_2	Incapable	Incapable	**Capable**	Incapable	Incapable	Incapable
DT_3	**Capable**	**Capable**	**Capable**	**Capable**	Incapable	Incapable
DT_4	**Capable**	Incapable	**Capable**	**Capable**	Incapable	Incapable
DT_5	**Capable**	Incapable	**Capable**	Incapable	Incapable	Incapable
DT_6	Incapable	Incapable	Incapable	Incapable	Incapable	Incapable
DT_7	**Capable**	**Capable**	**Capable**	**Capable**	**Capable**	**Capable**
DT_8	**Capable**	**Capable**	**Capable**	**Capable**	**Capable**	**Capable**
DT_9	**Capable**	**Capable**	**Capable**	**Capable**	**Capable**	**Capable**

The final stage, which involves the classification of *capability* to a meaningful QoS measurement, is demonstrated in Section 5.6. The rest of this section is concentrated on the variation of critical elements of the device's resources, and, specifically, on the effect that this may have in the capability/incapability determination.

5.5.2 Simulation results

Simulations were conducted for each device/objectivity combination by varying a number of key preliminary metrics, including: CPU utilisation, memory utilisation, and battery level. For example, the CPU utilisation of a device can easily decrease or increase according to the user's actions, such as the user has started a resource-consuming application, which increased the CPU utilisation by 35%. Along the same line, the amount of free memory available to the system can change for the same reasons. The battery, a metric of vast significance, varies with time and type of usage. For example, if a resource-constrained device is used as an ad-hoc router, it will cause the battery to decrease at a rate of approximately 30% faster that if it was idle (Migas, N., et. al., 2005).

Figure 5.102 presents the variation of the overall metric when the average PDA's CPU preliminary metric ranges from 0 to 100. According to Table 5.25, if the CPU prelimi-

nary metric exceeds the value of 40.43, the PDA becomes incapable of routing energy efficient network traffic. This is due to the fact that increased CPU utilisation can cause the battery to decrease at a much faster rate. In contrast, the CPU increase does not cause the overall metric to exceed the upper limit for asynchronous traffic, and thus the PDA will always be capable of routing this type of traffic, irrespective of the current CPU utilisation. However, if the CPU preliminary metric exceeds the predefined value of 95.25, the system automatically detects this and sets the overall metric to point to infinity (*see* Section 3.13). This is not illustrated in the graphs throughout this section for presentation purposes. Finally, the PDA turns to the incapable state for critical traffic after the CPU preliminary metric exceeds the threshold value of 35.03. A description of the results achieved for the remaining objectives is omitted, as the average PDA was incapable of achieving these objectives in the first place. Table 5.25 summarises the threshold values for each simulated preliminary metric, which can cause an average PDA to inverse its capability state for objectives O_1, O_3 and O_4, while Table 5.34 maps these threshold values to the actual corresponding CPU utilisation, memory usage, and battery level.

Figure 5.103 presents the variation in the overall metric for the PDA with high CPU utilisation device type, where the battery preliminary metric was variable. It can be seen that the battery level can have a strong impact on the overall metric, especially when the PDA's CPU is heavily utilised. Thus, as the CPU being overutilised, the PDA can only route asynchronous traffic (*see* Table 5.26). However, if the battery exceeds the threshold values of 66.65 the PDA moves to the incapable state for this type of traffic. As constant high CPU utilisation causes the battery to discharge at considerably faster rate than low utilisation (by almost 30%) (Migas, N., et. al., 2005), it is important to exclude it from routing when the remaining battery approaches low levels.

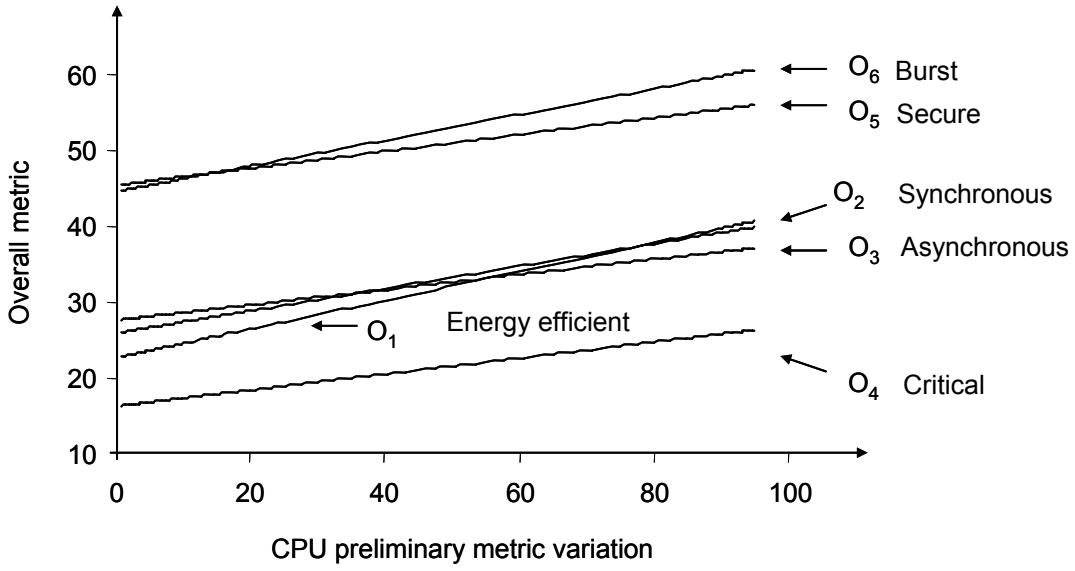

Figure 5.102. The effect of CPU's variation on the overall metric for the average PDA

Table 5.25: Threshold values for each simulated preliminary metric for the average PDA

	CPU	Memory	Battery
O_1	Metric ≥ 40.43	Metric ≥ 91.96	Metric ≥ 21
O_3	Metric ≥ 95.25	Metric ≥ 95.83	Metric ≥ 87.37
O_4	Metric ≥ 35.03	Metric ≥ 91.38	Metric ≥ 17.38

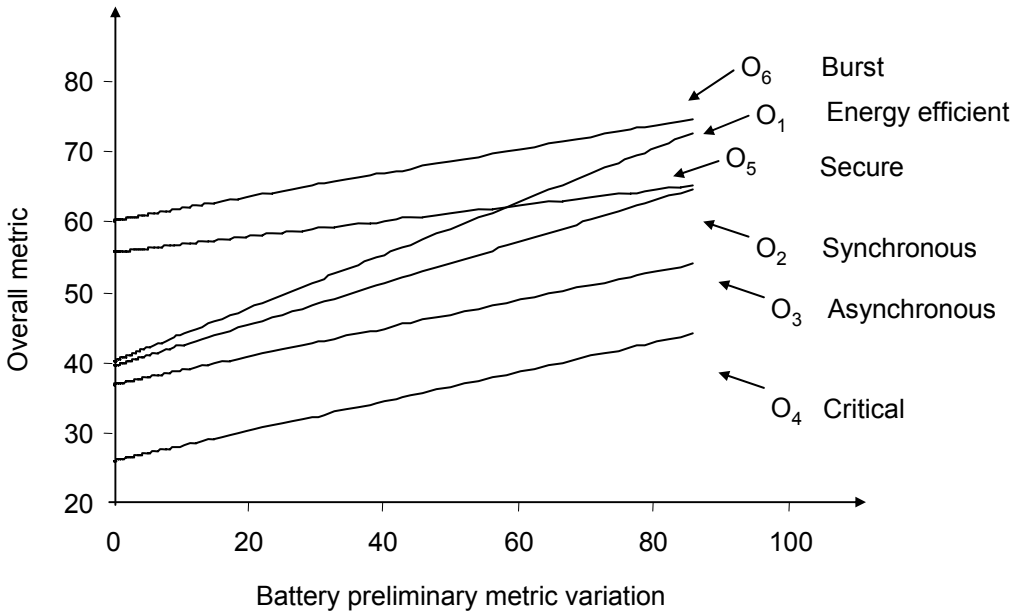

Figure 5.103:
The effect of battery's variation on the overall metric for the PDA with high CPU utilisation

Table 5.26:
Threshold values for each simulated preliminary metric for the PDA with high CPU utilisation

	CPU	Memory	Battery
Asynchronous	Metric ≥ 95.25	Metric ≥ 95.83	Metric ≥ 66.65

Figure 5.104 presents the variation in the overall metric for the PDA with a good throughput device type, where the memory metric was variable. This type of device is capable of routing synchronous network traffic, which is not supported by the average PDA, in addition to energy efficient, asynchronous, and critical. Table 5.27 summarises the preliminary threshold values for the CPU, memory, and battery that can cause the PDA to turn to incapable state. For the energy efficient, asynchronous, and critical traffic types, the threshold values are similar, however, a bit more relaxed, to the ones recorded for the average PDA. This close association is because these three objectives are not usu-

ally strongly linked to high-throughput. Even though this device type is allowed to route synchronous traffic, hard restrictions are imposed in terms of CPU utilisation, memory usage, and battery capacity, in order to protect the PDA from becoming unusable. This is evident in Table 5.28, where the threshold values for CPU, memory, and battery are significantly lower than the corresponding values for the rest of the supported objectives.

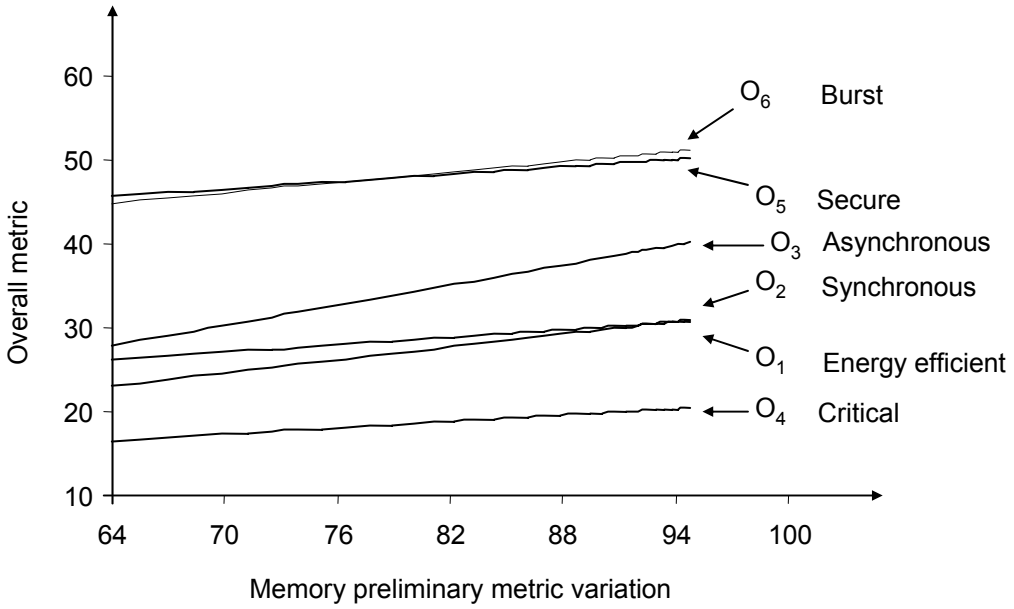

Figure 5.104: The effect of memory's variation on the overall metric for the PDA with good throughput

Table 5.27: Threshold values for each simulated preliminary metric for the PDA with good throughput

	CPU	Memory	Battery
Energy efficient	Metric ≥ 43.23	Metric ≥ 93.82	Metric ≥ 21
Synchronous	Metric ≥ 18.04	Metric ≥ 80.04	Metric ≥ 8.56
Asynchronous	Metric ≥ 95.25	Metric ≥ 95.83	Metric ≥ 87.37
Critical	Metric ≥ 39.05	Metric ≥ 94.8	Metric ≥ 19.74

Figure 5.105 presents the variation in the overall metric for the PDA with a poor throughput device type, where the battery preliminary metrics was variable. This shows that a PDA with poor throughput can still maintain the ability to achieve the same objectives as an average PDA, however, for the energy efficient objective, heavier restrictions are imposed in terms of CPU utilisation, memory usage, and battery capacity. According to Table 5.28, the threshold value of the CPU preliminary metric is reduced by 13 units, and approximately eight units for the memory and battery. The reason for this is that a PDA with low throughput consequently requires more time to route data, and thus more battery is consumed throughout this process, and also more CPU and memory is used.

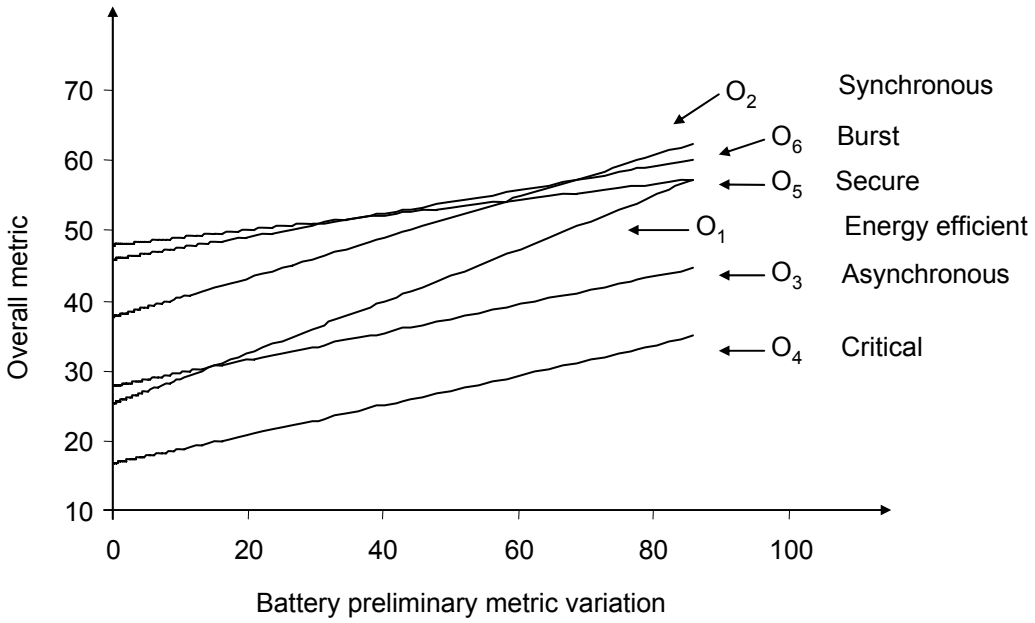

Figure 5.105: The effect of battery's variation on the overall metric for the PDA with poor throughput

Table 5.28: Threshold values for each simulated preliminary metric for the PDA with poor throughput

	CPU	Memory	Battery
Energy efficient	Metric ≥ 27.56	Metric ≥ 83.29	Metric ≥ 13.28
Asynchronous	Metric ≥ 95.25	Metric ≥ 95.83	Metric ≥ 87.37
Critical	Metric ≥ 32.45	Metric ≥ 93.82	Metric ≥ 16.28

Figure 5.106 presents the variation in the overall metric for the PDA with high error network protocol rate device type, where the battery metric was variable. As shown in Table 5.29, this device type can only support two objectives: energy efficient and asynchronous. Due to its high network protocol error rates, critical network traffic cannot be supported in contrast to the average PDA device type. In addition, high network protocol error rates have a significant impact on both energy efficient and asynchronous objectives. In comparison to the average PDA device type (*see* Table 5.25), the threshold values for energy efficient traffic is reduced by 25 units for the CPU, 17 units for the memory, and 14 units for the battery. Similarly, for asynchronous traffic the threshold values were reduced by 75 units for the CPU, 26 units for the memory, and 80 units for the battery. The reason for these significant differences is that high network protocol error rates can cause multiple retransmissions of the same data packets, and, thus, it requires more CPU time, memory usage, and battery.

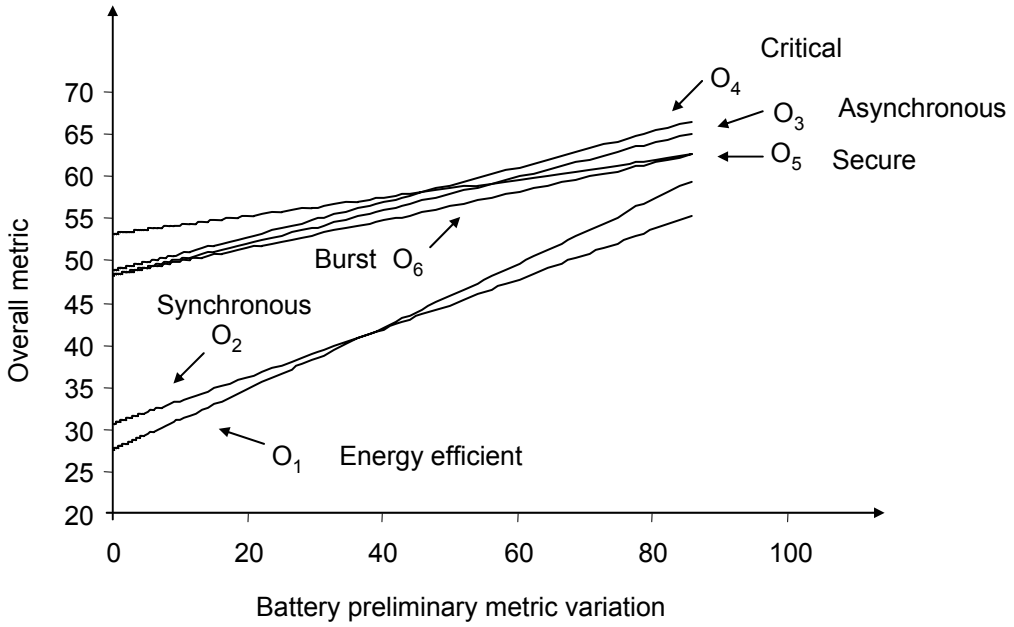

Figure 5.106: The effect of battery's variation on the overall metric for the PDA with high error rate

Table 5.29: Threshold values for each simulated preliminary metric for the PDA with high error rate

	CPU	Memory	Battery
Energy efficient	Metric ≥ 15.34	Metric ≥ 75.19	Metric ≥ 7.32
Asynchronous	Metric ≥ 19.97	Metric ≥ 69.81	Metric ≥ 9.24

Figure 5.107 presents the variation in the overall metric for the PDA with low battery device type, where the CPU preliminary metric was variable. This device type is the least competent, as battery resources are vital for operation. The overall metric is extremely sensitive for low battery readings in a similar way to high CPU utilisation. However, in this case the effect is more severe. In particular, the device can only support asynchronous traffic, if, and, only if, the CPU preliminary metric is lower than 17 and the memory preliminary metric is lower than 71 (*see* Table 5.30). This demonstrates the significance which battery power and utilisation status has on the overall metric calculation. Justifiably, these two factors are significantly important for any device, especially when intended to be used as an ad-hoc router.

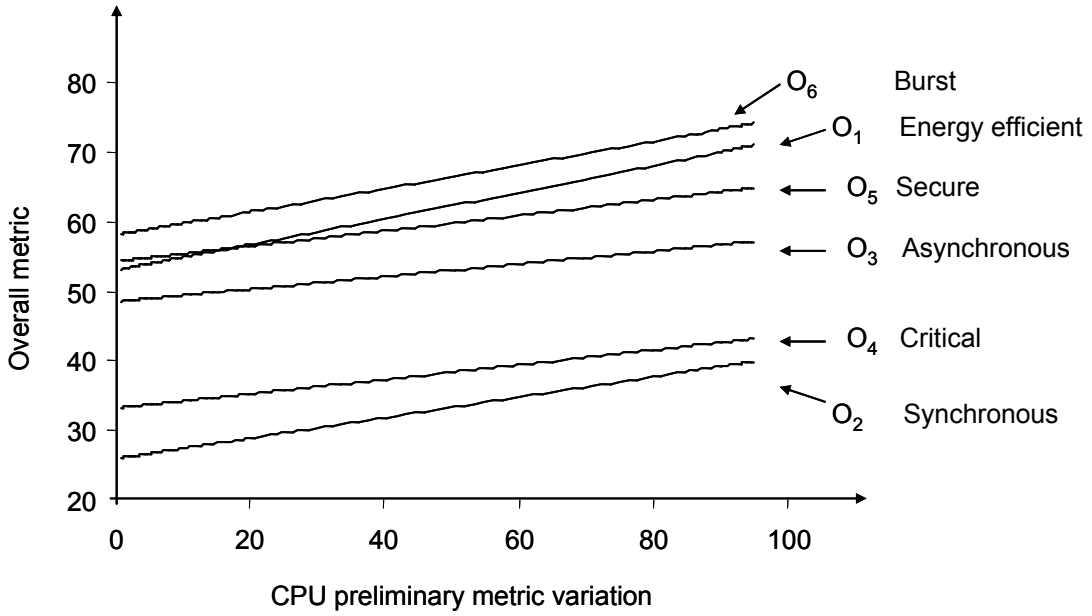

Figure 5.107: The effect of battery's variation on the overall metric for the PDA with low battery

Table 5.30: Threshold values for each simulated preliminary metric for the PDA with low battery

	CPU	Memory	Battery
Asynchronous	Metric ≥ 17.11	Metric ≥ 71.20	Metric ≥ 85.76

Figure 5.108 presents the variation in the overall metric for the average laptop device type, where the battery preliminary metric was variable. As shown in Table 5.31, an average strength laptop is capable of supporting all objectives. This device type is preferable for objectives, such as energy efficient, synchronous, asynchronous, and critical, rather than objectives such as secure and burst. This can be seen by examining the threshold values for each simulated preliminary metric in relation to each objective. Accordingly, this device type can cope better with CPU and memory increases, as well as battery reserve decreases, for the first four objectives than it can for the last two. Comparing these values to the average PDA results, it can be seen that the average laptop device type is given higher priority for all objectives, apart from asynchronous which is set to be equal.

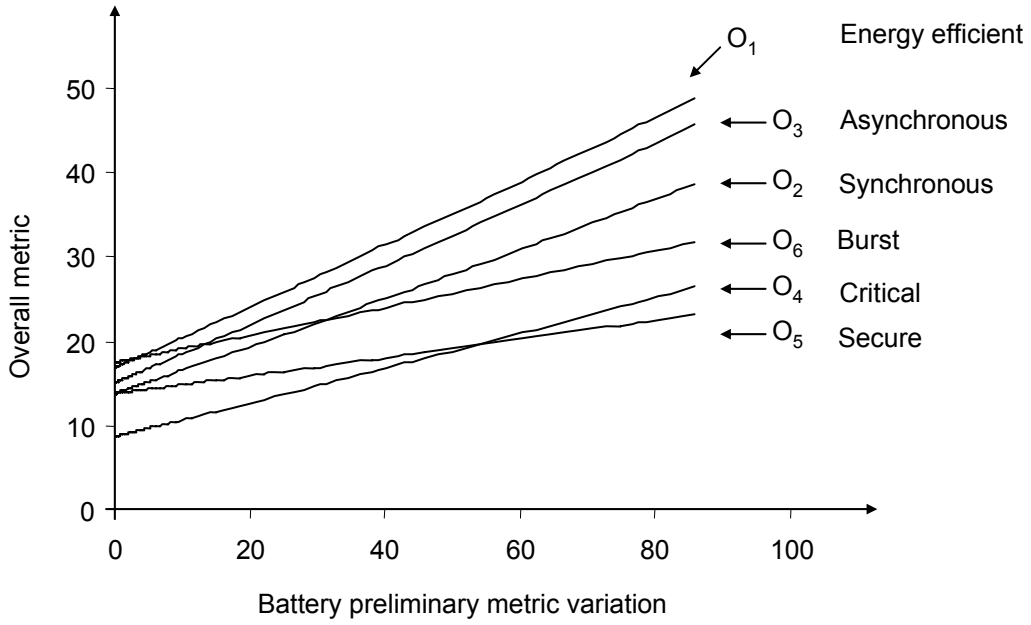

Figure 5.108: The effect of battery's variation on the overall metric for the average laptop

Table 5.31: Threshold values for each simulated preliminary metric for the average laptop

	CPU	Memory	Battery
Energy efficient	Metric ≥ 72.33	Metric ≥ 92.83	Metric ≥ 36.47
Synchronous	Metric ≥ 78.9	Metric ≥ 95.83	Metric ≥ 40.14
Asynchronous	Metric ≥ 95.25	Metric ≥ 95.83	Metric ≥ 87.37
Critical	Metric ≥ 95.25	Metric ≥ 95.83	Metric ≥ 56.18
Secure	Metric ≥ 12.88	Metric ≥ 65.5	Metric ≥ 11.53
Burst	Metric ≥ 17.11	Metric ≥ 68.4	Metric ≥ 16.28

Figure 5.109 presents the variation in the overall metric for the good fitness laptop device type, where the memory metric was variable. Similarly to results obtained for the average laptop, this device type also supports all objectives. In addition, it provides a better solution by further stretching most of the threshold values (*see* Table 5.32). The advantage of this device type is clearly visible for the security and burst objectives, where the threshold values are on average increased by 45 units for CPU, 19 units for memory, and 46 units for battery. Thus, it is safe to conclude that this device type could be more efficiently used by the routing protocol for these two objectives, whereas, all previously stated devices may struggle to cope with this, or may not cope at all.

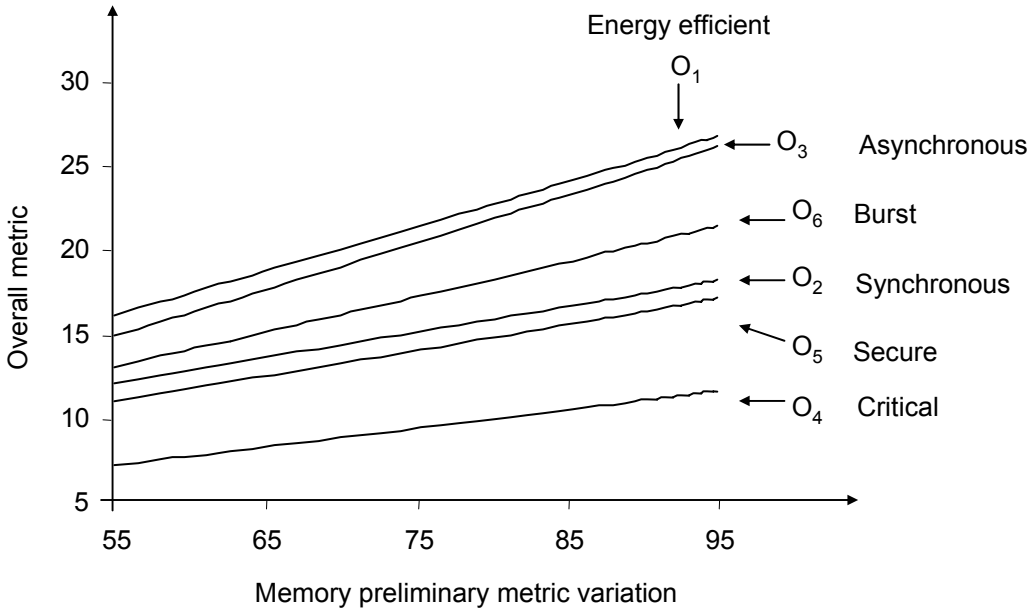

Figure 5.109: The effect of memory's variation on the overall metric for the strong laptop

Table 5.32: Threshold values for each simulated preliminary metric for the strong laptop

	CPU	Memory	Battery
Energy efficient	Metric ≥ 94.92	Metric ≥ 95.83	Metric ≥ 49.97
Synchronous	Metric ≥ 95.25	Metric ≥ 95.83	Metric ≥ 54.09
Asynchronous	Metric ≥ 95.25	Metric ≥ 95.83	Metric ≥ 87.37
Critical	Metric ≥ 95.25	Metric ≥ 95.83	Metric ≥ 70.77
Secure	Metric ≥ 59.36	Metric ≥ 82.25	Metric ≥ 58.27
Burst	Metric ≥ 62.31	Metric ≥ 89.41	Metric ≥ 62.47

Figure 5.110 presents the variation in the overall metric for the workstation device type, where the memory metric was variable. This device type has the advantage over the previous device types, in that it does not rely on battery power to achieve any of the objectives, since workstations are not battery-driven. This certain advantage along with its greater strength in networking and calculations power brings this device type in the position to be the perfect candidate to achieve all 6 objectives in the most efficient manner. This can be verified by examining the data in Table 5.33, where all threshold values are the most stretched, apart from asynchronous objective, which is purposely set to be equal for all device types, in order to allow load balancing between weak and strong devices.

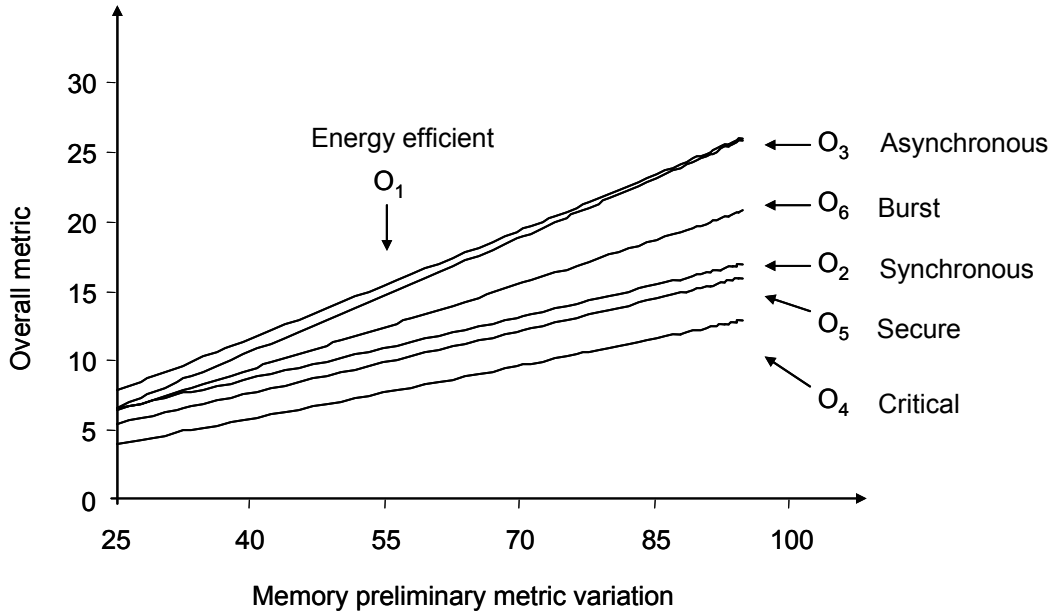

Figure 5.110: The effect of memory's variation on the overall metric for the powerful workstation

Table 5.33: Threshold values for each simulated preliminary metric for the powerful workstation

	CPU	Memory
Energy efficient	Metric ≥ 95.25	Metric ≥ 95.83
Synchronous	Metric ≥ 95.25	Metric ≥ 95.83
Asynchronous	Metric ≥ 95.25	Metric ≥ 95.83
Critical	Metric ≥ 95.25	Metric ≥ 95.83
Secure	Metric ≥ 86.73	Metric ≥ 89.41
Burst	Metric ≥ 81.32	Metric ≥ 91.38

Table 5.34:

The threshold values mapped to the actual CPU utilisation (C), memory usage (M), and battery level (B)

	Energy efficient			**Synchronous**			**Asynchronous**			**Critical**			**Secure**			**Burst**		
	C	M	B	C	M	B	C	M	B	C	M	B	C	M	B	C	M	B
DT_1	48	82	49	-	-	-	95	95	85	44	81	46	-	-	-	-	-	-
DT_2	-	-	-	-	-	-	95	95	74	-	-	-	-	-	-	-	-	-
DT_3	50	86	49	29	68	36	95	95	85	47	89	48	-	-	-	-	-	-
DT_4	38	71	42	-	-	-	95	95	85	42	86	45	-	-	-	-	-	-
DT_5	26	64	34	-	-	-	31	60	37	-	-	-	-	-	-	-	-	-
DT_6	-	-	-	-	-	-	28	61	84	-	-	-	-	-	-	-	-	-
DT_7	70	95	59	75	95	61	95	95	85	95	95	69	23	57	40	28	59	45
DT_8	94	95	66	95	95	68	95	95	85	95	95	76	61	70	70	63	78	72
DT_9	95	95	N/A	95	95	N/A	95	95	N/A	95	95	N/A	82	78	N/A	77	81	N/A

5.5.3 Outcomes and evaluation of the metrics simulation experimentation

This section demonstrated the calculation of the overall routing metric, based on test results obtained for various device types (*see* Section 5.1 - 5.3), which were further enhanced in order to provide support for the each device type that is defined in Section 4.6. For the purposes of this section, six distinct objectives and nine device types were defined, all with different requirements and characteristics, respectively. It has been shown that the metric calculation process is correct, as each device type is assigned a distinct metric for each objective, and determined as capable or incapable according to the desired configuration. In this way, QoS can be guaranteed, as each device will always be assigned to certain routing scenarios according to its capabilities, utilisation, and network status. In addition, this method allows *low-requirements* network traffic to flow through non-optimal routes, and therefore optimal routes may not be overburdened.

Furthermore, a number of simulation cases were presented in order to demonstrate the effect that changes of vital device elements can have on the overall metric. Results show that when key metrics are changed, such as the remaining battery drops, or that the CPU is highly utilised, or that the device is running low on available memory, the device turns to the incapable state of routing *high-requirements* traffic types. The variation of the overall metric is adequately sensitive in all cases, and thus this demonstrates the ability of the proposed scheme to rapidly respond to critical changes. In addition, the threshold values, when a device becomes incapable of achieving a certain objective, are presented and are fully justified.

5.6 A MARIAN-enabled ad-hoc network application scenario

The aim of this section is to provide a demonstration of MARIAN's main aspects. For this purpose, an ad-hoc network has been defined with a total of 44 nodes in order to provide in-depth analysis. Aspects which are covered by this include: the metric-driven clustering formation process; the reactive route discovery process; the proactive network topology gathering process; and the routing decisions taken by a source node in relation to the gathered routing metrics. In addition, it provides network overhead information imposed by both reactive and proactive network discovery processes for this particular ad-hoc network example. Initially, each node's routing fitness, in the example network topology, is classified according to the device type that the node has been defined to belong to. Then an overall routing metric and the capability/incapability determination is calculated for all the distinct routes, which resulted from the previously initiated reactive route discovery. The capable routes are then further classified into a final metric. Three routing

scenarios, which use these routes to transmit different types of traffic, have been defined and presented here. The first scenario assumes that each node remains stationary, and the metrics supplied remain the same. In contrast, in the second scenario, critical elements of intermediate devices are set to be variable in order to demonstrate the MARIAN's adaptability in critical changes. Finally, the third scenario incorporates nodal movements, in order to demonstrate the effect that these have on the routing decisions taken by a source node.

It should thus demonstrate MARIAN's clustering formation process, reactive and proactive route and network discovery processes, and, most importantly, the metric-driven properties which are proposed to effectively provide QoS and route redundancy. In addition, it demonstrates how devices' critical variations force the protocol to change its capability/incapability applied policy, and rapidly respond to new demands. Specifically, the aims are:

- Demonstrate MARIAN's clustering formation process.
- Demonstrate the on-demand route discovery, as well as the proactive network topology acquisition processes.
- Identify the network overhead imposed by an initiated route discovery, where the distance of the source and destination is the distance of the network's diameter.
- Identify the total network discovery mobile agent migrations, and the network overhead imposed.
- Measure the time taken for each approach to complete the network discovery process.
- Demonstrate the determination of the capability/incapability metrics for all the retrieved routes based on the predefined objective.
- Demonstrate the effect that variation of critical devices' elements can cause to the applied capability/incapability metrics for each route involved.
- Demonstrate the effect that mobility can cause to the applied capability/incapability metrics for each route involved.

5.6.1 The ad-hoc network's topology used in the application scenario

Figure 5.111 presents an ad-hoc network consisting of a total of 44 network nodes, where the communication link between two neighbouring devices is represented by a straight line. Accordingly, N_1 and N_2 are neighbouring nodes, as well as nodes N_2 and N_3.

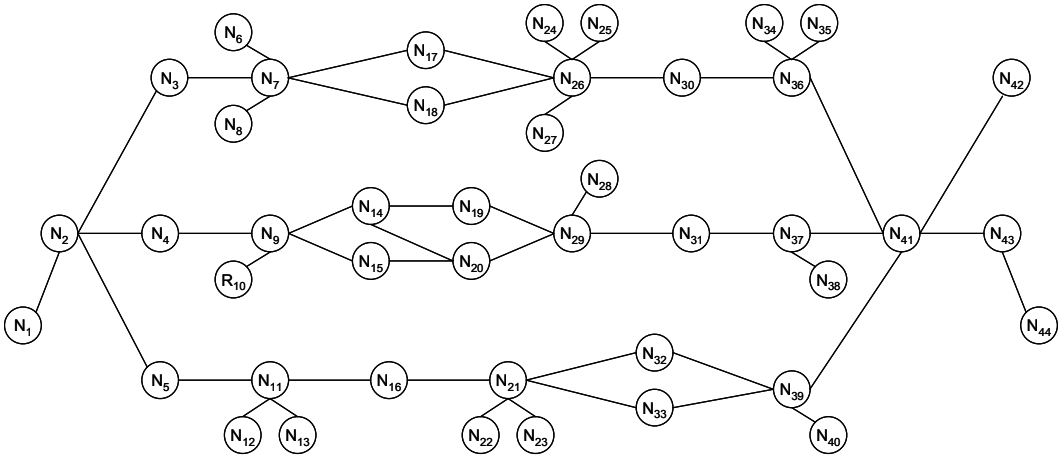

Figure 5.111: An example of an ad-hoc network topology

In order for this topology to be grouped into clusters, the cluster-head metric (node-ID) of each participating node is required. These metrics can be calculated by using the weighting system defined for the cluster-head objective (O_{CH}) (*see* Section 3.13), in relation to the devices' preliminary metrics. As previously mentioned (*see* Section 3.13), an additional preliminary metric (T_m) is used for this calculation, which represents the devices' mobility patterns. However, throughout this routing scenario the devices are assumed to be stationary, thus T_m is set to zero for all cases. Accordingly, the cluster-head metrics (node-IDs) have been determined for each device type and are:

- **Average-strength PDA (P)**. Cluster-head metric of 24.
- **Average-strength laptop (L)**. Cluster-head metric of 11.
- **Strong-fitness laptop (SL)**.Cluster-head metric of 7.
- **Powerful workstation (W)**. Cluster-head metric of 5.

In relation to the above values, a device which belongs to any one of these device types (DTs) inherits the corresponding cluster-head metric. For instance, as shown in Table 5.35, N_1 and N_6 belong to the first category and thus both inherit the cluster-head metric of 24. Although the cluster-head metrics (node IDs) must be represented by a unique value (*see* Appendix B.7-B.8), for the purposes of this example, all nodes were assumed to perfectly match one of the previously presented categories, and, consequently, some nodes ended up having the same cluster-head metric. Nevertheless, this was carefully designed in such a way so as to leave the clustering formation process unaffected.

Table 5.35: Each node categorised into four distinct device types

Node	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9	N_{10}	N_{11}	N_{12}	N_{13}	N_{14}	N_{15}	N_{16}
DT	P	W	L	L	SL	P	W	P	W	P	W	P	P	SL	P	SL

Node	N_{17}	N_{18}	N_{19}	N_{20}	N_{21}	N_{22}	N_{23}	N_{24}	N_{25}	N_{26}	N_{27}	N_{28}	N_{29}	N_{30}
DT	P	P	L	SL	W	P	P	P	P	W	P	P	W	P

Node	N_{31}	N_{32}	N_{33}	N_{34}	N_{35}	N_{36}	N_{37}	N_{38}	N_{39}	N_{40}	N_{41}	N_{42}	N_{43}	N_{44}
DT	L	SL	P	P	P	W	W	P	W	P	SL	W	W	P

According to MARIAN's clustering formation process (*see* Section 3.5.1) and in relation to Table 5.35, the ad-hoc network presented in Figure 5.111 gets transformed to an organised clustered network, as shown in Figure 5.112. The square boxes represent the cluster-head nodes, the circular ones are the member nodes, and the triangular ones are the gateway and distributed gateway nodes. As shown in Figure 5.112, there are a total of 12 cluster-heads, 11 gateways, four distributed gateways, and 14 members. Table 5.36 presents the clusters formed, as well as the cluster-head and members of each cluster.

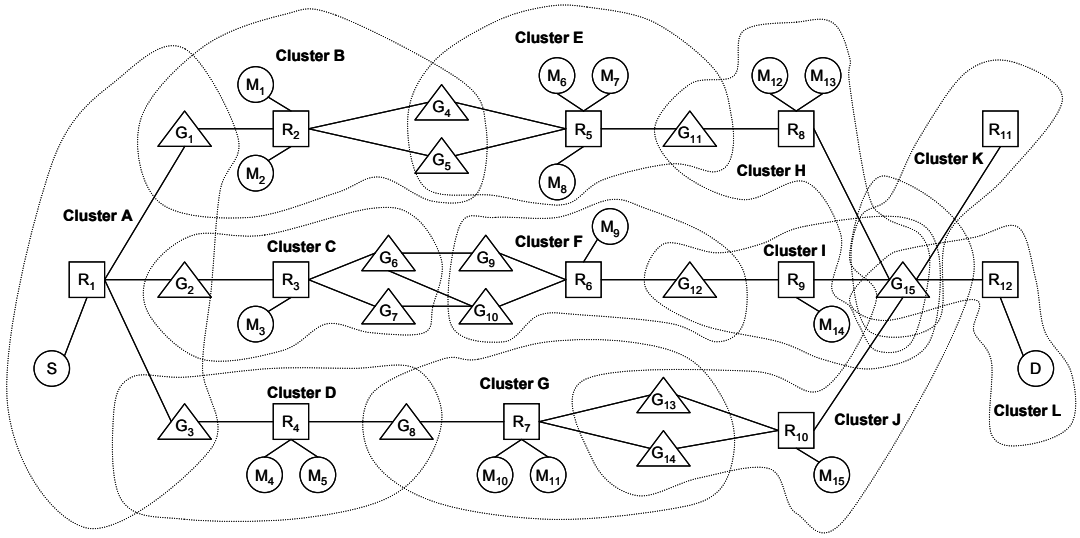

Figure 5.112: The example ad-hoc network organised into clusters

Table 5.36: The clusters formed, along with their respective cluster-heads and non cluster-head nodes.

Cluster	Cluster-head	Members
A	R_1	S, G_1, G_2, G_3
B	R_2	M_1, M_2, G_1, G_4, G_5
C	R_3	M_3, G_2, G_6, G_7
D	R_4	M_4, M_5, G_3, G_8
E	R_5	M_6, M_7, M_8, G_4, G_5, G_{11}
F	R_6	M_9, G_9, G_{10}, G_{12}
G	R_7	M_{10}, M_{11}, G_8, G_{13}, G_{14}
H	R_8	M_{12}, M_{13}, G_{11}, G_{15}
I	R_9	M_{14}, G_{12}, G_{15}
J	R_{10}	M_{15}, G_{13}, G_{14}, G_{15}
K	R_{11}	G_{15}
L	R_{12}	D, G_{15}

5.6.2 Reactive route discovery – network overhead

This section assumes that node *S* requires to communicate with node *D*, but does not have a route for it. In this case, node *S* initially constructs a minimal propagation RREQ packet (see Section 3.6.1), and send it to its cluster-head (R_1). If R_1 has single or multiple routes to *D*, it will reply with single or multiple minimal propagation RREPs, respectively, otherwise it will reply with a route-not-available (RNA) packet (*see* Section 3.6.1). This example, assumes that R_1 has no route to *D*, and thus it replies back to *S* with a RNA packet. In this case, *S* constructs a full-propagation RREQ and sends it to R_1, which will forward it to its neighbouring cluster-head(s), and along this fashion, the RREQ will be flooded throughout the network, traversing **only** through *key* nodes, such as cluster-heads, gateways, and distributed gateways. In other words, member nodes which usually have the lowest performance characteristics (due to metric-driven clustering formation) are typically not used.

Once *D* receives the first RREQ packet, it immediately constructs a RREP packet and sends it to its own cluster-head, which in this case is R_{12}. The RREP is then propagated back to the source, along the same chain of cluster-heads that the corresponding RREQ took. Since MARIAN utilises multiple routes, more RREQs which followed a different chain of cluster-heads will eventually arrive at *D*, and consequently *D* will transmit multiple RREPs. Each RREP packet gathers the node and agency IDs, as well as the routing metrics of each node along its journey. In addition, when a cluster-head identifies more than a single gateway leading to the next clusterhead, it clones the RREP packet and sends one copy to each gateway (or distributed gateway) leading to the next cluster-heads. This process is also performed by distributed gateways. Table 5.37 shows the transmissions of all RREQ and RREP packets that were necessary for *S* to retrieve all possible routes to *D*.

Table 5.37: RREQ and RREP packet transmission for reactive route discovery from S to D

Iterations	Parallel transmission	Parallel transmission	Parallel transmission	Parallel transmission	Parallel transmission
1	$S \rightarrow R_1$				
2	$R_1 \rightarrow G_1$				
3	$G_1 \rightarrow R_2$	$R_1 \rightarrow G_2$			
4	$R_2 \rightarrow G_4$	$G_2 \rightarrow R_3$	$R_1 \rightarrow G_3$		
5	$G_4 \rightarrow R_5$	$R_3 \rightarrow G_6$	$G_3 \rightarrow R_4$		
6	$R_5 \rightarrow G_{11}$	$G_6 \rightarrow G_{10}$	$R_4 \rightarrow G_8$		
7	$G_{11} \rightarrow R_8$	$G_{10} \rightarrow R_6$	$G_8 \rightarrow R_7$		
8	$R_8 \rightarrow G_{15}$	$R_6 \rightarrow G_{12}$	$R_7 \rightarrow G_{13}$		
9	$G_{15} \rightarrow R_9$	$G_{13} \rightarrow R_{10}$			
10	$G_{12} \rightarrow R_9$	$R_{10} \rightarrow G_{15}$			
11	$G_{15} \rightarrow R_8$	$R_9 \rightarrow G_{12}$			
12	$R_8 \rightarrow G_{11}$	$G_{12} \rightarrow R_6$	$R_9 \rightarrow G_{15}$		
13	$G_{11} \rightarrow R_5$	$G_{15} \rightarrow R_8$			

14	$R_8 \rightarrow G_{15}$				
15	$R_8 \rightarrow G_{11}$	$G_{15} \rightarrow R_{10}$			
16	$G_{11} \rightarrow R_5$	$R_{10} \rightarrow G_{15}$			
17	$G_{15} \rightarrow R_9$	$R_{10} \rightarrow G_{13}$			
18	$R_9 \rightarrow G_{15}$	$G_{13} \rightarrow R_7$			
19	$R_9 \rightarrow G_{12}$	$G_{15} \rightarrow R_{10}$			
20	$R_8 \rightarrow G_{15}$	$G_{12} \rightarrow R_6$	$R_{10} \rightarrow G_{13}$		
21	$G_{15} \rightarrow R_{11}$	$G_{13} \rightarrow R_7$			
22	$R_{10} \rightarrow G_{15}$				
23	$G_{15} \rightarrow R_{11}$				
24	$R_9 \rightarrow G_{15}$				
25	$G_{15} \rightarrow R_{11}$				
26	$R_8 \rightarrow G_{15}$				
27	$G_{15} \rightarrow R_{12}$				
28	$R_{12} \rightarrow D$	$R_{10} \rightarrow G_{15}$			
29	$R_9 \rightarrow G_{15}$	$D \rightarrow R_{12}$			
30	$G_{15} \rightarrow R_{12}$				
31	$R_{12} \rightarrow G_{15}$				
32	$G_{15} \rightarrow R_8$	$R_{12} \rightarrow D$			
33	$R_8 \rightarrow G_{11}$	$G_{15} \rightarrow R_{12}$			
34	$G_{11} \rightarrow R_5$	$D \rightarrow R_{12}$			
35	$R_5 \rightarrow G_4$	$R_{12} \rightarrow D$			
36	$G_4 \rightarrow R_2$	$R_5 \rightarrow G_5$	$R_{12} \rightarrow G_{15}$		
37	$R_2 \rightarrow G_1$	$D \rightarrow R_{12}$	$G_{15} \rightarrow R_{10}$		
38	$G_5 \rightarrow R_2$	$G_1 \rightarrow R_1$	$R_{12} \rightarrow G_{15}$	$R_{10} \rightarrow G_{13}$	
39	$R_2 \rightarrow G_1$	$R_1 \rightarrow S$	$G_{15} \rightarrow R_9$	$G_{13} \rightarrow R_7$	$R_{10} \rightarrow G_{14}$
40	$G_1 \rightarrow R_1$	$R_9 \rightarrow G_{12}$	$R_7 \rightarrow G_8$		
41	$R_1 \rightarrow S$	$G_{12} \rightarrow R_6$	$G_{14} \rightarrow R_7$	$G_8 \rightarrow R_4$	
42	$R_6 \rightarrow G_9$	$R_7 \rightarrow G_8$	$R_4 \rightarrow G_3$		
43	$G_9 \rightarrow G_6$	$R_6 \rightarrow G_{10}$	$G_8 \rightarrow R_4$	$G_3 \rightarrow R_1$	
44	$R_1 \rightarrow S$	$G_6 \rightarrow R3$	$G_{10} \rightarrow G_7$	$R_4 \rightarrow G_3$	
45	$G_{10} \rightarrow G_6$	$R_3 \rightarrow G_2$	$G_3 \rightarrow R_1$		
46	$G_7 \rightarrow R_3$	$R_1 \rightarrow S$			
47	$G_6 \rightarrow R_3$	$G_2 \rightarrow R_1$			
48	$R_3 \rightarrow G_2$	$R_1 \rightarrow S$			
49	$G_2 \rightarrow R_1$				
50	$R_3 \rightarrow G_2$	$R_1 \rightarrow S$			
51	$G_2 \rightarrow R_1$				
52	$R_1 \rightarrow S$				

As an example, Figures 5.113 and 5.114 show the packet transmissions that took place at iteration 20 and 39, respectively. Where, three packets are transmitted, in total, one by R_8, G_{12}, and R_{10}, to G_{15}, R_6, and G_{13}, respectively. As shown by the packets' identifiers, they are all RREQs. At iteration 39, five packets are transmitted, in total, one by G_{15}, G_{13}, R_{10}, R_2, and R_1 to R_9, R_7, G_{14}, G_1, and S, respectively. As shown by the packets' identifiers, they are all RREPs.

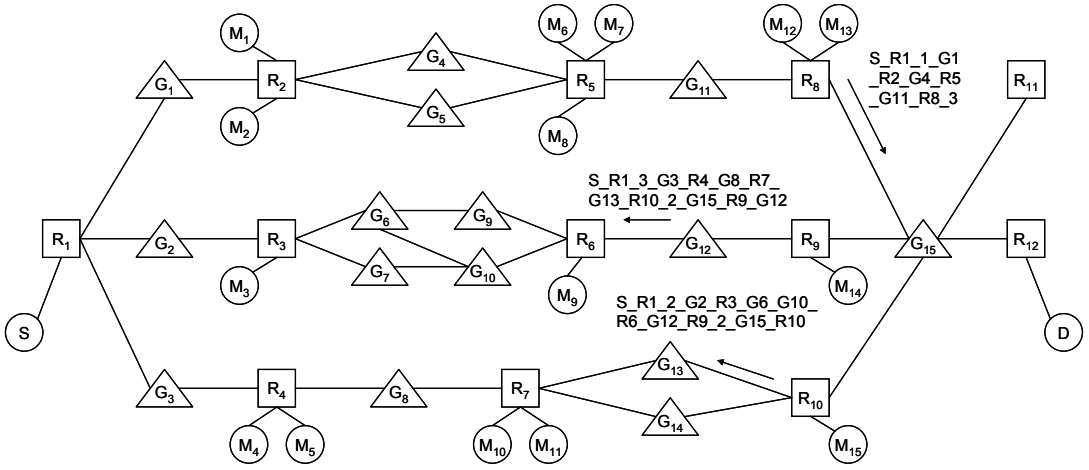

Figure 5.113: A snapshot of the network's RREQ transmissions at iteration 20

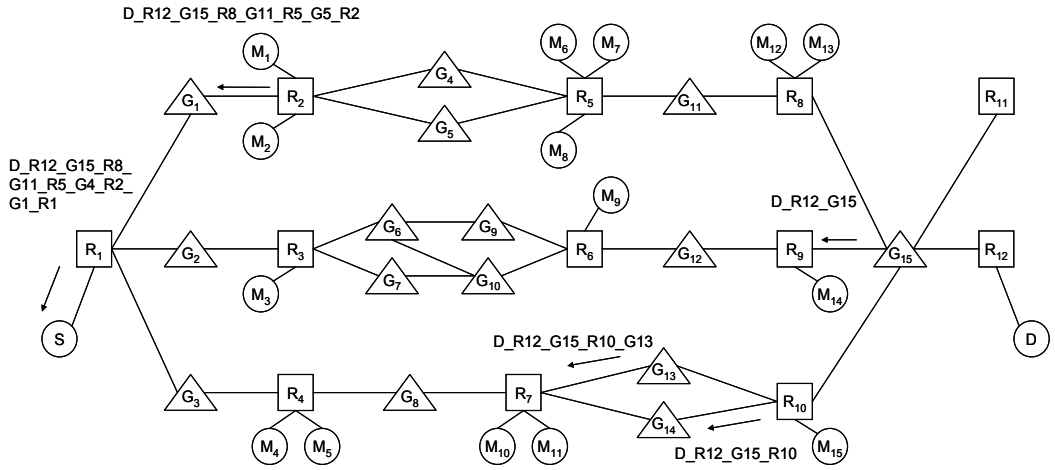

Figure 5.114: A snapshot of the network's RREP transmissions at iteration 39

MARIAN's reactive route discovery complexity in terms of time and communication is similar to CBRP (*see* Section 2.2), and thus imposes a certain network overhead in terms of throughput consumption. This overhead has been approximately calculated for this topology, specifically for *S* requesting a route to *D*. The route discovery process requires 52 iterations in total to complete, and the maximum and minimum packets transmitted, at any given time, is five and one, respectively. This is illustrated in Figure 5.115, while Figure 5.116 presents the total network throughput consumed over the total time required for this process to complete.

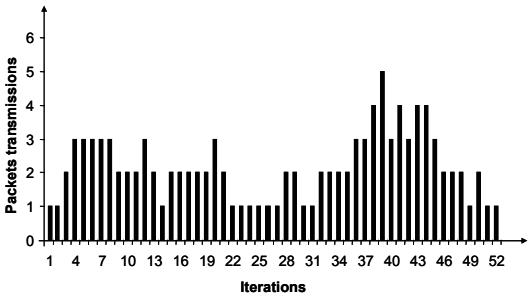

Figure 5.115: The total number of packets transmitted for each iteration

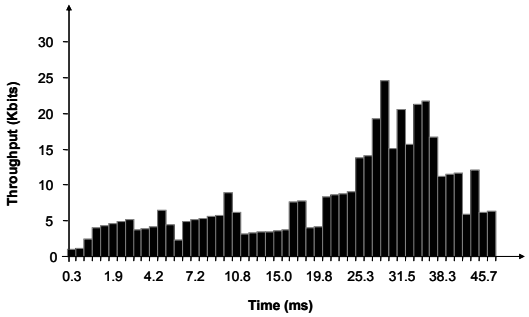

Figure 5.116: Total throughput consumed over the total time taken for route discovery to be completed

According to Figure 5.116, the total time taken by the route discovery to *D* was approximately 47ms. This was calculated by assuming that all communication links were based on the IEEE 802.11b standard and provided an average throughput of 4Mbits/s, which is logical to assume, since the 11Mbits/s upper limit is never reached. In addition, it was assumed that the first packet's size was 1KB and that it increased by 10% for each forthcoming iteration. Thus, the packet's size at the last iteration was approximately 6KB. The overall throughput overhead (oTHo) for this scenario was calculated based on:

$$oTHo = \frac{\sum_{i=1}^{i=52}\left(TH(I_n)\right)}{\sum_{i=1}^{i=52}\left(T(I_n)\right)} \tag{5.1}$$

According to this, the oTHo is given by the summation of the throughput consumed by the total transmission at each iteration, divided by the summation of the total time consumed by each iteration. Thus, the overall network's overhead for route discovery from *S* to *D* was shown to be approximately 8,900Kbits/s. However, the throughput overhead imposed at each routing link (*rl*) is calculated slightly differently and is given by:

$$lTHo = \frac{oTHo}{\sum(rl)} \tag{5.2}$$

The link throughput overhead (lTHo) is defined as the average throughput reduction experienced by a routing link on the network, which is caused by the route discovery process. According to this, the link throughput overhead can be calculated by dividing the overall network's throughput overhead (oTHo) over the total number of routing links in the network. In this routing scenario, 32 routing links exist in total, thus, the throughput required by each link for network discovery is approximately 278.125Kbits/s.

The overhead percentage (LO) that this process imposes at each link is given by:

$$LO\ (\%) = \frac{lTHo}{UpBandLim} \tag{5.3}$$

According to this, the average throughput percentage consumed by each routing link for route discovery is given by the division of the link's throughput overhead over the link's practical upper bandwidth. For example, assuming that the practical maximum bandwidth of an IEEE 802.11b link is 4Mbits/s and the average throughput overhead of a link is 278.125Kbits/s, thus the 6.95 (%) of the links capacity is consumed by the propagation of RREQ and RREP packets.

5.6.3 Routes retrieved by the reactive network discovery

The following summarises the routes that source node *S* has learned by the initiated reactive route discovery targeted for node *D*. Along with the routes, node *S* also gathers the routing metric (the array of PMs) of each individual node, such as:

1. $\mathbf{S \rightarrow R_1 \rightarrow G_1 \rightarrow R_2 \rightarrow G_4 \rightarrow R_5 \rightarrow G_{11} \rightarrow R_8 \rightarrow G_{15} \rightarrow R_{12} \rightarrow D}$
   PM[S], PM[R_1], PM[G_1], PM[R_2], PM[G_4], PM[R_5], PM[G_{11}], PM[R_8], PM[G_{15}], PM[R_{12}], PM[D]
2. $\mathbf{S \rightarrow R_1 \rightarrow G_1 \rightarrow R_2 \rightarrow G_5 \rightarrow R_5 \rightarrow G_{11} \rightarrow R_8 \rightarrow G_{15} \rightarrow R_{12} \rightarrow D}$
   PM[S], PM[R_1], PM[G_1], PM[R_2], PM[G_5], PM[R_5], PM[G_{11}], PM[R_8], PM[G_{15}], PM[R_{12}], PM[D]
3. $\mathbf{S \rightarrow R_1 \rightarrow G_2 \rightarrow R_3 \rightarrow G_6 \rightarrow G_9 \rightarrow R_6 \rightarrow G_{12} \rightarrow R_9 \rightarrow G_{15} \rightarrow R_{12} \rightarrow D}$
   PM[S], PM[R_1], PM[G_2], PM[R_3], PM[G_6], PM[G_9], PM[R_6], PM[G_{12}], PM[R_9], PM[G_{15}], PM[R_{12}], PM[D]
4. $\mathbf{S \rightarrow R_1 \rightarrow G_2 \rightarrow R_3 \rightarrow G_6 \rightarrow G_{10} \rightarrow R_6 \rightarrow G_{12} \rightarrow R_9 \rightarrow G_{15} \rightarrow R_{12} \rightarrow D}$
   PM[S], PM[R_1], PM[G_2], PM[R_3], PM[G_6], PM[G_{10}], PM[R_6], PM[G_{12}], PM[R_9], PM[G_{15}], PM[R_{12}], PM[D]
5. $\mathbf{S \rightarrow R_1 \rightarrow G_2 \rightarrow R_3 \rightarrow G_7 \rightarrow G_{10} \rightarrow R_6 \rightarrow G_{12} \rightarrow R_9 \rightarrow G_{15} \rightarrow R_{12} \rightarrow D}$
   PM[S], PM[R_1], PM[G_2], PM[R_3], PM[G_7], PM[G_{10}], PM[R_6], PM[G_{12}], PM[R_9], PM[G_{15}], PM[R_{12}], PM[D]
6. $\mathbf{S \rightarrow R_1 \rightarrow G_3 \rightarrow R_4 \rightarrow G_8 \rightarrow R_7 \rightarrow G_{13} \rightarrow R_{10} \rightarrow G_{15} \rightarrow R_{12} \rightarrow D}$
   PM[S], PM[R_1], PM[G_3], PM[R_4], PM[G_8], PM[R_7], PM[G_{13}], PM[R_{10}], PM[G_{15}], PM[R_{12}], PM[D]
7. $\mathbf{S \rightarrow R_1 \rightarrow G_3 \rightarrow R_4 \rightarrow G_8 \rightarrow R_7 \rightarrow G_{14} \rightarrow R_{10} \rightarrow G_{15} \rightarrow R_{12} \rightarrow D}$
   PM[S], PM[R_1], PM[G_3], PM[R_4], PM[G_8], PM[R_7], PM[G_{14}], PM[R_{10}], PM[G_{15}], PM[R_{12}], PM[D]

5.6.4 Proactive network topology discovery – network overhead

There are cases in which a node can significantly benefit from considerably low route discovery times. These cases often include applications with low latency requirements and strict timing restrictions. Mobile agents can enhance a routing protocol by proactively gathering routing and metric information in favour of their cluster-heads from which

they originate. By providing this information, cluster-heads can be fairly well informed, at all times, of available routes and the metrics associated. In this way, they can provide a reliable and fast medium that maintains routing information for themselves and their registered members, resulting in reduced latency, often imposed by a non-cluster-head route discovery.

Cluster-heads can be programmed to create a topology discovery agent in a periodic fashion and/or based on triggered events (*see* Section 3.6.2). Assuming that cluster-head R_1 creates a network topology gathering mobile agent based on a triggered event. The mobile agent will initially examine R_1's neighbouring cluster-head information and will then create a number of clones that exactly match the number of R_1's neighbouring cluster-heads. The clones will then be dispatched to the intermediate gateway nodes leading to these cluster-heads. Once the cloned agents reach a point where they cannot progress any further, they will return back home gathering the network topology and the associated routing metrics along their way back. When all agents return back to R_1, they cooperatively build a routing database which can be later used by either R_1 or the members which are registered to R_1. Table 5.38 presents the agent migrations necessary for R_1 to collect the whole network's topology which was presented in Figure 5.112.

Table 5.38: Mobile agent migrations necessary for R_1 to collect the full network topology.

Iterations	Parallel migration	Parallel migration	Parallel migration	Parallel migration
1	$R_1 \rightarrow G_1$			
2	$G_1 \rightarrow R_2$	$R_1 \rightarrow G_2$		
3	$R_2 \rightarrow G_4$	$G_2 \rightarrow R_3$	$R_1 \rightarrow G_3$	
4	$G_4 \rightarrow R_5$	$R_3 \rightarrow G_6$	$G_3 \rightarrow R_4$	
5	$R_5 \rightarrow G_{11}$	$G_6 \rightarrow G_{10}$	$R_4 \rightarrow G_8$	
6	$G_{11} \rightarrow R_8$	$G_{10} \rightarrow R_6$	$G_8 \rightarrow R_7$	
7	$R_8 \rightarrow G_{15}$	$R_6 \rightarrow G_{12}$	$R_7 \rightarrow G_{13}$	
8	$G_{15} \rightarrow R_9$	$G_{13} \rightarrow R_{10}$		
9	$G_{12} \rightarrow R_9$	$R_{10} \rightarrow G_{15}$		
10	$G_{15} \rightarrow R_8$	$R_9 \rightarrow G_{12}$		
11	$R_8 \rightarrow G_{11}$	$G_{12} \rightarrow R_6$	$R_9 \rightarrow G_{15}$	
12	$G_{11} \rightarrow R_5$	$G_{15} \rightarrow R_8$	$R_6 \rightarrow G_{12}$	
13	$R_5 \rightarrow G_{11}$	$R_8 \rightarrow G_{15}$	$G_{12} \rightarrow R_9$	
14	$R_8 \rightarrow G_{11}$	$G_{15} \rightarrow R_{10}$		
15	$G_{11} \rightarrow R_5$	$R_{10} \rightarrow G_{15}$		
16	$R_5 \rightarrow G_{11}$	$G_{15} \rightarrow R_9$	$R_{10} \rightarrow G_{13}$	
17	$G_{11} \rightarrow R_8$	$R_9 \rightarrow G_{15}$	$G_{13} \rightarrow R_7$	
18	$G_{11} \rightarrow R_8$	$R_9 \rightarrow G_{12}$	$G_{15} \rightarrow R_{10}$	$R_7 \rightarrow G_{13}$
19	$R_8 \rightarrow G_{15}$	$G_{12} \rightarrow R_6$	$R_{10} \rightarrow G_{13}$	
20	$G_{15} \rightarrow R_{11}$	$R_6 \rightarrow G_{12}$	$G_{13} \rightarrow R_7$	
21	$G_{12} \rightarrow R_9$	$R_{10} \rightarrow G_{15}$	$R_7 \rightarrow G_{13}$	
22	$G_{15} \rightarrow R_{11}$	$G_{13} \rightarrow R_{10}$		
23	$R_9 \rightarrow G_15$	$G_{13} \rightarrow R_{10}$		
24	$G_{15} \rightarrow R_{11}$			

25	$R_8 \rightarrow G_{15}$			
26	$G_{15} \rightarrow R_{12}$			
27	$R_{10} \rightarrow G_{15}$			
28	$R_9 \rightarrow G_{15}$			
29	$G_{15} \rightarrow R_{12}$			
30	$G_{15} \rightarrow R_{12}$			
31	$R_9 \rightarrow G_{15}$			
32	$R_8 \rightarrow G_{15}$			
33	$R_8 \rightarrow G_{15}$			
34	$R_{11} \rightarrow G_{15}$			
35	$R_9 \rightarrow G_{15}$			
36	$R_{10} \rightarrow G_{15}$			
37	$R_{11} \rightarrow G_{15}$			
38	$R_{10} \rightarrow G_{15}$			
39	$R_{11} \rightarrow G_{15}$			
40	$R_{12} \rightarrow G_{15}$			
41	$R_{12} \rightarrow G_{15}$			
42	$R_{12} \rightarrow G_{15}$			
43	$G_{15} \rightarrow R_8$			
44	$R_8 \rightarrow G_{11}$	$G_{15} \rightarrow R_{10}$		
45	$G_{11} \rightarrow R_5$	$G_{15} \rightarrow R_9$	$R_{10} \rightarrow G_{13}$	
46	$G_{15} \rightarrow R_8$	$R_9 \rightarrow G_{12}$	$G_{13} \rightarrow R_7$	
47	$R_8 \rightarrow G_{11}$	$G_{12} \rightarrow R_6$	$G_{15} \rightarrow R_{10}$	
48	$G_{11} \rightarrow R_5$	$G_{15} \rightarrow R_8$	$R_{10} \rightarrow G_{13}$	
49	$R_8 \rightarrow G_{11}$	$G_{15} \rightarrow R_{10}$	$G_{13} \rightarrow R_7$	
50	$G_{11} \rightarrow R_5$	$G_{15} \rightarrow R_9$	$R_{10} \rightarrow G_{13}$	
51	$G_{15} \rightarrow R_9$	$G_{13} \rightarrow R_7$		
52	$G_{15} \rightarrow R_8$	$R_9 \rightarrow G_{12}$		
53	$R_8 \rightarrow G_{11}$	$G_{12} \rightarrow R_6$	$G_{15} \rightarrow R_{10}$	
54	$G_{11} \rightarrow R_5$	$G_{15} \rightarrow R_9$	$R_{10} \rightarrow G_{13}$	
55	$R_5 \rightarrow G_4$	$R_9 \rightarrow G_{12}$	$G_{13} \rightarrow R_7$	
56	$G_4 \rightarrow R_2$	$G_{12} \rightarrow R_6$	$R_7 \rightarrow G_8$	
57	$R_2 \rightarrow G_1$	$R_9 \rightarrow G_{12}$	$G_8 \rightarrow R_4$	
58	$G_1 \rightarrow R_1$	$G_{12} \rightarrow R_6$	$R_4 \rightarrow G_3$	
59	$R_6 \rightarrow G_{10}$	$G_3 \rightarrow R_1$		
60	$G_{10} \rightarrow G_6$			
61	$G_6 \rightarrow R_3$			
62	$R_3 \rightarrow G2$			
63	$G_2 \rightarrow R_1$			

Figure 5.117 shows the mobile agent migrations which take place at iteration 18. At this stage the mobile agents are propagating throughout the network in order to gather routing information. As shown, four mobile agents were transmitted in this iteration, from G_{11}, R_9, R_7, and G_{15} to R_8, G_{12}, G_{13}, and R_{10}, respectively. Figure 5.118 shows the mobile agent migrations which took place at iteration 56. At this stage the mobile agents are returning to their corresponding home platforms. Accordingly, three mobile agents were transmitted, from G_4, G_{12}, and R_7, to R_2, R_6, and G_8, respectively.

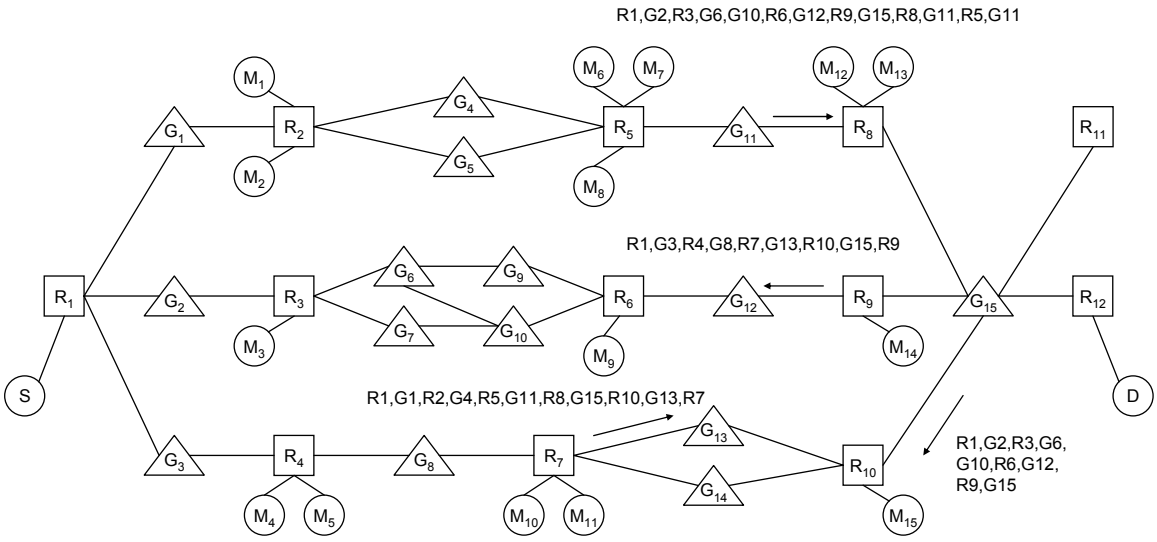

Figure 5.117: Total mobile agent migrations at iteration 18

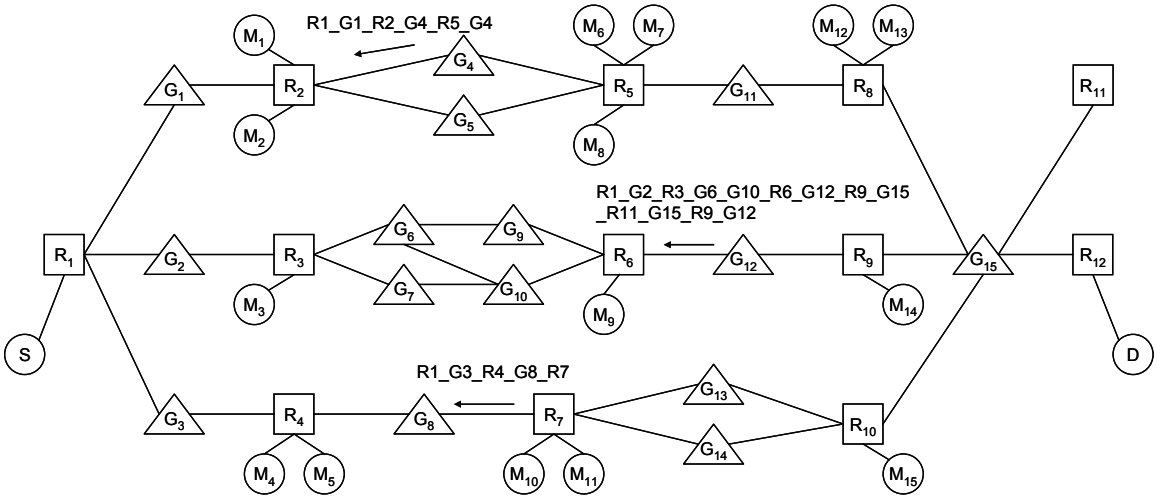

Figure 5.118: Total mobile agent migrations at iteration 56

MARIAN's proactive network topology gathering complexity, in terms of time, and communication, is similar to the respective reactive approach (see Section 5.6.2), however, the migration times involved with mobile agents are considerably higher than packets propagation. Specifically, as shown in Section 5.4.1, the average mobile agent migration is approximately 1s, and thus this approach will always require significantly more time than the reactive. However, as nothing is dependant on the proactive approach, mobile agent delays should not influence the network's performance, as the proactive approach was specifically designed as an additional feature in order to enhance the reactive approach. Figure 5.119 presents the total number of migrations required in order for R_1 to gather the network's topology information. As shown, the total number of migrations required was 128, split over 63 iterations, with a maximum of four and a

minimum of 1 migration. Figure 5.120 presents the total network throughput consumed over the total time required for the network topology gathering process.

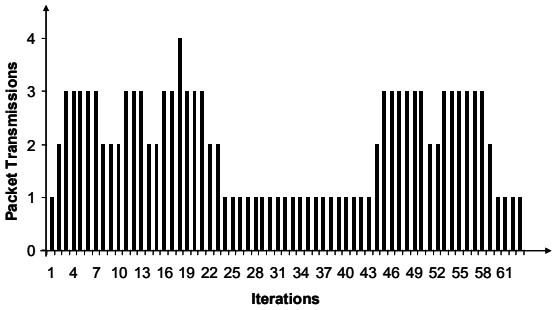

Figure 5.119: The total number of mobile agents migrated in each iteration

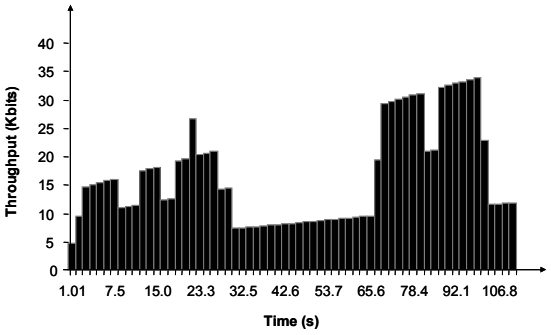

Figure 5.120: Total throughput consumed over the total time taken for the proactive approach to be completed

According to Figure 5.120, the total time taken by the mobile agents to retrieve the network's topology and associated metrics is approximately 112s. This is calculated by assuming that the mobile agent, at the first iteration, required 1s to migrate, while at each, consecutive, iteration it required an additional 2.5% of the original time. This is due to the fact that as mobile agents gather network topology information, their sizes increases. In the iteration 63 the mobile agent required 2.55s to migrate.

The overall throughput overhead for the proactive network discovery scenario was calculated based on the equation 6.1. Thus, the **overall** network overhead imposed by network topology and routing metrics collection agents, which were originated from cluster-head R_1, is approximately 9,295Kbits/s, which is similar to the overhead imposed by the static approach. However, their main difference is the time taken to complete, where the static approach accomplished the task of retrieving all routes from *S* to *D* within 47ms, while the mobile agent approach required 112s to retrieve the complete network topology and metrics. Even though the mobile agents are required to deliver a lot more information than the static agents, the significant difference in time is mainly due to the migration module of Grasshopper, which serialises the mobile agents and transmits it to the next hop, in a slow way. A solution to this problem may be a light-weighted mobile agent system which is only targeted for ad-hoc routing with an optimal time migration module incorporated.

The link throughput overhead, imposed by mobile agent migrations, is given by the mathematical expression presented in equation 5.2. Thus, the throughput consumed over each link is approximately 290.48Kbits/s. The overhead percentage imposed by the mobile agents is given by the mathematical expression in equation 5.3 and is equal to 7.262%. All possible routes to every possible destination, along with the preliminary met-

rics of each node in the graph are now known to R_1. All members of R_1 can quote its services for route discovery of a required route, along with the associated routing metrics.

5.6.5 Routing scenarios

Source node *S* can now estimate the QoS offered by the retrieved routes to *D* (*see* Section 5.6.3) in relation to each predefined objective. This is achieved by passing the preliminary metrics of each device along a desired route to the overall metric calculation process (*see* Section 3.13). Once that capability/incapability determination of each device has been deduced, node *S* can then estimate the capability/incapability of each retrieved route. This is performed in a simple manner, where a node sets capability flags to a route that consists of capable nodes only, and sets an incapability flag to a route that consists of at least one incapable node. Accordingly, Table 5.39 summarises the capability/incapability results for each of the retrieved routes. Node *S* then calculates the average route metric value, the minimum and maximum routing metrics, and the standard deviation, for each retrieved route (*see* Section 5.6.3) and supported objective combinations, as shown in Table 5.40 and Table 5.41.

Table 5.39: Capability/incapability determination results for each of the retrieved routes

	Energy (O_1)	**Synch (O_2)**	**Asynch (O_3)**	**Critial (O_4)**	**Secure (O_5)**	**Burst (O_6)**
Route 1	**Capable**	Incapable	**Capable**	**Capable**	Incapable	Incapable
Route 2	**Capable**	Incapable	**Capable**	**Capable**	Incapable	Incapable
Route 3	**Capable**	**Capable**	**Capable**	**Capable**	**Capable**	**Capable**
Route 4	**Capable**	**Capable**	**Capable**	**Capable**	**Capable**	**Capable**
Route 5	**Capable**	Incapable	**Capable**	**Capable**	Incapable	Incapable
Route 6	**Capable**	**Capable**	**Capable**	**Capable**	**Capable**	**Capable**
Route 7	**Capable**	Incapable	**Capable**	**Capable**	Incapable	Incapable

Table 5.40:
Average metric, minimum and maximum, and standard deviation for each route for objectives O_1 - O_3

	Energy efficient (O_1)				**Synchronous (O_2)**				**Asynchronous (O_3)**			
	AV	MIN	MAX	SD	AV	MIN	MAX	SD	AV	MIN	MAX	SD
R_1	13	23	8	11	12	26	7	13	13	28	7	15
R_2	13	23	8	11	12	26	7	13	13	28	7	15
R_3	12	17	8	6	10	14	7	5	10	15	7	6
R_4	11	17	8	6	9	14	7	5	10	15	7	6
R_5	12	23	8	11	11	26	7	13	12	28	7	15
R_6	10	12	8	3	8	10	7	2	9	11	7	3
R_7	11	23	8	11	10	26	7	13	11	28	7	15

Table 5.41:
Average metric, minimum and maximum, and standard deviation for each route for objectives O_4 - O_6

	Critical (O_4)				Secure (O_5)				Burst (O_6)			
	AV	MIN	MAX	SD	AV	MIN	MAX	SD	AV	MIN	MAX	SD
R_1	8	17	4	9	16	46	6	28	17	45	7	27
R_2	8	17	4	9	16	46	6	28	17	45	7	27
R_3	6	9	4	4	9	14	6	6	11	18	7	8
R_4	6	9	4	4	9	14	6	6	10	18	7	8
R_5	7	17	4	9	12	46	6	28	14	45	7	27
R_6	5	6	4	1	7	9	6	2	8	10	7	2
R_7	6	17	4	9	11	46	6	28	12	45	7	27

According to Tables 5.40 - 5.41, the following can be deduced:

- The most optimal route for all routing scenarios is R_6, at the average metric is the lowest for all objectives and the standard deviation is also the lowest.
- The second most optimal route for all routing scenarios is R_4. This is because the average metric and the standard deviation is the second lowest for all objectives.
- R_1 and R_2 are of identical strength for all objectives as they share the same characteristics, that is, they are composed of devices with identical characteristics. Also, they are the worst routes for all routing scenarios.
- R_1, R_2, R_5, and R_7 seem to be ideal for low requirements traffic, while the remaining routes seem ideal for high, as well as for low requirements traffic. However, in order to provide load balancing, the first set of routes should be used for low requirements traffic, while the second for high.

Based on Tables 5.40 and 5.41 and Table 3.19, node *S* can translate these values into a final metric representing the strength of each route to accomplish each of the predefined objectives, as presented in Table 5.42.

Table 5.42: Final metric for each route/objective combination

	O_1	O_2	O_3	O_4	O_5	O_6
R_1	Good	N/A	V. Good	V. Good	N/A	N/A
R_2	Good	N/A	V. Good	V. Good	N/A	N/A
R_3	V. Good	V. Good	Excellent	V. Good	Good	Good
R_4	V. Good	V. Good	Excellent	V. Good	Good	Good
R_5	Good	N/A	V. Good	V. Good	N/A	N/A
R_6	Excellent	V. Good	Excellent	Excellent	Good	Good
R_7	Good	N/A	V. Good	V. Good	N/A	N/A

By referring to Table 5.42, node *S* can deduce the best route for the network traffic it aims to transmit. Also, node *S* must pay careful attention to the requirements imposed by the type of traffic it aims to transmit, and thus must choose the best route, accordingly. For example, if node *S* needs to transmit critical network traffic with high requirements, it should choose route R_6 which offers excellent QoS. In the case that the network traffic

requirements are greater than the offered QoS, the node can either select a route providing the highest possible maximum QoS, or abandon its transmission until its requirements are fully met. It is assumed that all nodes adhere to the following rules:

- Nodes are honest about their QoS requests, and thus get only what they need provided that such a route exists.
- A node avoids frequently selecting a single route for transmitting network traffic, provided that there exists at least another route which offers the same QoS.
- A node wishing to transmit network traffic with higher requirements than the available QoS offered by any possible route, may either decide to select a route less optimal or abandon transmission until such a route is found.
- A node wishing to transmit network traffic with lower requirements than the available QoS offered by any possible route, may transmit over the route which offers the lowest QoS.

To illustrate this, a simple case scenario where, node *S* wishes to send asynchronous, synchronous, and critical network traffic to *D*. Table 5.43 shows this, along with the sequence and iteration of each transmission, as well as the requirements imposed by each type of traffic. Thus, node *S* wishes to send asynchronous traffic at its first, third, and sixth transmissions, while synchronous, energy efficient, and critical traffic at its second, forth, and fifth transmissions, respectively. The requirements vary with respect to the traffic scenario, that is, for asynchronous, it does not require any special QoS, for synchronous the level of QoS requested is medium-high, and for energy efficient and critical is set to high.

Table 5.43: A simple routing scenario

Traffic scenario	Sequence	Iteration	Requirements
Asynchronous	1, 3, 6	3	Low, Low, Low
Synchronous	2	1	Medium-high
Energy efficient	4	1	High
Critical	5	1	High

For the first transmission, node *S* chooses randomly between routes R_1, R_2, R_5, and R_7, as these are the closest to *S*'s requirements, where, at this instance, it is assumed that node *S* chose route R_1. Decisions of this nature are kept in memory in order to assist the node in future decisions of a similar nature. For the second transmission, node *S* chooses randomly between routes R_3, R_4, and R_6, which is assumed to have chosen route R_4. For the third transmission, node *S* chooses randomly between routes R_2, R_5, and R_7, as route R_1 has been previously used. It is assumed that node *S* chose R_2. For the fourth transmission

the only available choice which matches *S*'s criteria is R_6. For the fifth transmission, node *S* chooses route R_6 since it is the only one that matches *S*'s criteria. Even though route R_6 has been used to route energy efficient network traffic before, there are no alternative routes that provides the same required QoS for critical network traffic, and therefore node *S* is forced to select route R_6 for this type of traffic. Finally, for the last transmission, node *S* can choose between routes R_5 and R_7, as routes R_1 and R_2 have been previously selected. It is assumed that node *S* chose route R_7. Node's *S* decisions are summarised as:

- Network traffic asynchronous, Sequence 1, Iteration 1, Requirements Low, Route R_1.
- Network traffic synchronous, Sequence 2, Iteration 1, Requirements Medium-high, Route R_4.
- Network traffic asynchronous, Sequence 3, Iteration 2, Requirements Low, Route R_2.
- Network traffic energy efficient, Sequence 4, Iteration 1, Requirements High, Route R_6.
- Network traffic critical, Sequence 5, Iteration 1, Requirements High, Route R_6.
- Network traffic asynchronous, Sequence 6, Iteration 3, Requirements Low, Route R_7.

Thus, node *S* receives the required QoS and does not overburden the *best* route, which in this scenario is route R_6, when there are alternative routes that offer equivalent QoS.

5.6.6 Variable preliminary metrics – Routing scenario

As previously mentioned, preliminary metrics are not fixed and they change over-time due to factors, such as decrease in the battery reserves, high utilisation status, and so on. Cluster-heads monitor the preliminary metrics of neighbouring nodes which are gateways or distributed gateways, as well as their own metrics. This is achieved by constantly reading broadcasted NNTs used by the clustering maintenance process (see Section 3.5.1). Cluster-heads compare new values with previously received information, and highlight sudden changes. Cases such as these include, but are not limited, to the following:

- Sudden increase of a routing device's utilisation status, such as CPU utilisation has increased by 60%.
- Sudden drop in battery level, such as battery level has dropped by 45%.
- Increased network protocol errors, such as the packet error rate has increased by 30%.

In the case where a cluster-head identifies a significant change in the preliminary metrics of one of its registered routing devices, including itself, it reports these changes by flooding the network with a metric update. In order to keep these updates at a minimum level, without compromising the network's reliability, these update messages are only transmit-

ted when changes of critical metrics are persistent over a sufficient amount of time.

For the purposes of this illustration, it was assumed that source node *S* is required to transmit a number of different types of network traffic to the destination node *D*. The topology remains as depicted in Figure 5.112, however, some routing devices within the network have undertaken some changes in their critical metrics. Table 5.44 presents these changes, while Table 5.45 presents the metric values before and after these changes took place.

Table 5.44: Changes in key elements for nodes G_5, G_{10}, and G_{13}-G_{14}

Node	Change	Old_Role	New_Role
G_5	High error rate	Gateway	Gateway
G_{10}	High memory & CPU utilisation	Distributed Gateway	Distributed Gateway
G_{13}	Low battery	Gateway	Gateway
G_{14}	Increased throughput	Gateway	Gateway

Table 5.45: Metric changes for nodes G_5, G_{10}, and G_{13}-G_{14}

Objectives	G_5		G_{10}		G_{13}		G_{14}	
	Old_m	**New_m**	Old_m	**New_m**	Old_m	**New_m**	Old_m	**New_m**
Energy eff.	23	28	12	39	12	39	23	22
Synch	26	31	10	28	10	31	26	23
Asynch	28	48	11	30	11	37	28	28
Critical	17	49	6	20	6	21	17	16
Secure	46	53	9	24	9	17	46	45
Burst	45	48	10	33	10	22	45	45
Clustering	24	33	7	21	7	19	24	22

These changes had a crucial effect on the capability/incapability determination on each of the retrieved routes and objectives combination, which were previously presented in Table 5.39. Once node *S* receives the metric updates, it calculates a new capability /incapability determination, as shown in Table 5.46. The routes that are left out are the ones that are not affected by the changes.

Table 5.46:
The new capability/incapability determination information resulted from the metric updates

	O_1	O_2	O_3	O_4	O_5	O_6
Route 2	**Capable**	Incapable	**Capable**	Incapable	Incapable	Incapable
Route 4	Incapable	Incapable	**Capable**	**Capable**	Incapable	Incapable
Route 5	Incapable	Incapable	**Capable**	**Capable**	Incapable	Incapable
Route 6	Incapable	Incapable	**Capable**	Incapable	Incapable	Incapable
Route 7	**Capable**	**Capable**	**Capable**	**Capable**	Incapable	Incapable

Tables 5.47 and 5.48 present the new average, minimum and maximum, and standard deviation metric values for routes R_2, R_4, and R_5 - R_7, after the metric updates have been

received by node *S*.

Table 5.47: New information for each route and objectives O_1 - O_3 due to metric variations

	Energy efficient				Synchronous				Asynchronous			
	AV	MIN	MAX	SD	AV	MIN	MAX	SD	AV	MIN	MAX	SD
R_2	13	28	8	14	13	31	7	17	15	48	7	29
R_4	14	39	8	22	11	28	7	15	12	30	7	16
R_5	15	39	8	22	13	28	7	15	13	30	7	16
R_6	13	39	8	22	11	31	7	17	12	37	7	21
R_7	11	22	8	10	10	23	7	11	11	28	7	15

Table 5.48: New information for each route and objectives O_4 - O_6 due to metric variations

	Critical				Secure				Burst			
	AV	MIN	MAX	SD	AV	MIN	MAX	SD	AV	MIN	MAX	SD
R_2	11	49	4	32	17	53	6	33	17	48	7	29
R_4	7	20	4	11	10	24	6	13	12	33	7	18
R_5	8	20	4	11	14	24	6	13	16	33	7	18
R_6	7	21	4	12	8	17	6	8	10	22	7	11
R_7	6	16	4	8	11	45	6	28	12	45	7	27

Source node *S* recalculates the final metric for each route and objective combination, based on the values in Tables 5.47- 5.48, as shown in Table 5.49.

Table 5.49: Final route metrics for each objective, taking into account the metric changes

	Energy	Synch	Asynch	Critical	Secure	Burst
R_1	Good	N/A	V. Good	V. Good	N/A	N/A
R_2	Good	N/A	Poor	N/A	N/A	N/A
R_3	V. Good	V. Good	Excellent	V. Good	Good	Good
R_4	N/A	N/A	Good	Good	N/A	N/A
R_5	N/A	N/A	Good	Good	N/A	N/A
R_6	N/A	N/A	Poor	N/A	N/A	N/A
R_7	V. Good	Poor	V. Good	V. Good	N/A	N/A

It is assumed that source node *S* wishes to transmit the same types of traffic, at the same sequence and iterations, as shown in Table 5.43. The route selection is, at this stage, totally different than previously stated, and it thus adapts to dynamic changes in order to meet QoS requirements. The new route selections for the same type of data traffic are shown bellow:

- Network traffic asynchronous, Sequence 1, Iteration 1, Requirements Low, Route R_6.
- Network traffic synchronous, Sequence 2, Iteration 1, Requirements Medium-high, Route R_3.
- Network traffic asynchronous, Sequence 3, Iteration 2, Requirements Low, Route R_6.

- Network traffic energy efficient, Sequence 4, Iteration 1, Requirements High, Route R_7.
- Network traffic critical, Sequence 5, Iteration 1, Requirements High, Route R_1.
- Network traffic asynchronous, Sequence 6, Iteration 3, Requirements Low, Route R_6.

This demonstrates the strength of the routing protocol to dynamically adapt to critical changes in the devices' metrics in order to meet QoS requirements. In addition, it avoids overburdening optimal routes by routing lower requirement network traffic through less optimal routes.

5.6.7 Nodal movements – Routing scenario

This routing scenario assumes that nodes G_{10}, G_6, M_{13}, and G_{15} have left from their previous positions and moved towards the direction of the arrows, as depicted in Figure 5.121, resulting in new network topology which is presented in Figure 5.122.

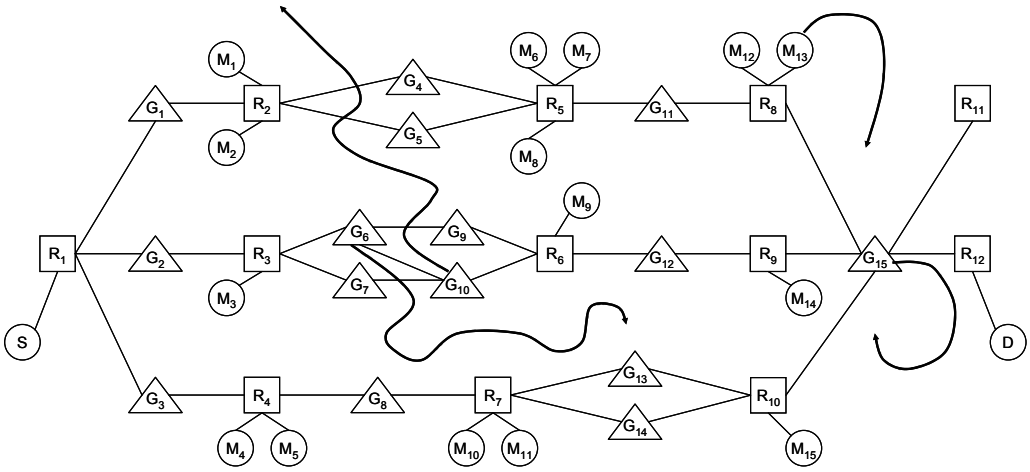

Figure 5.121: The movement direction of nodes: G_{10}, G_6, M_{13}, and G_{15}

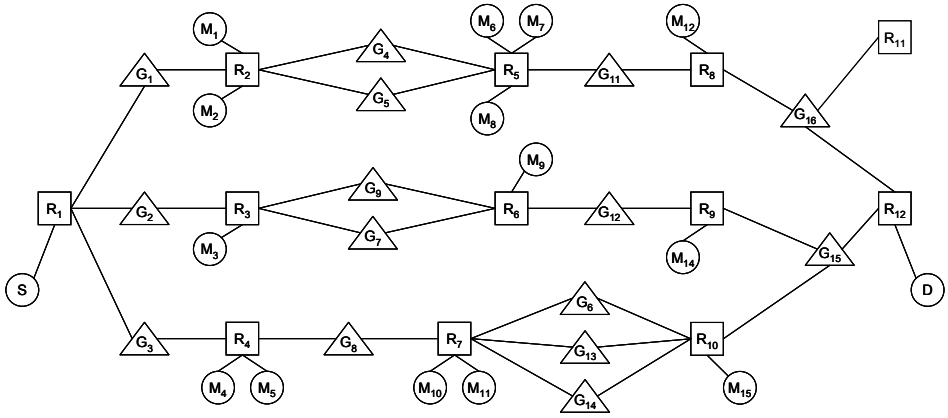

Figure 5.122: The new network topology after the movement of nodes: G_{10}, G_6, M_{13}, and G_{15}

The overall network's structure has now been changed with new routes having been formed and some of the old routes having been released. In more detail, node G_{10} left the network, while node G_6 moved away from the clusters formed by R_3 and R_6 and joined the clusters formed by R_7 and R_{10}. Node G_{15} moved away from R_{11} and R_8, which resulted in breaking the links with these nodes. Node M_{13} replaced G_{15}'s position by moving closer to R_{11} and R_{12}, and thus forming a link with each. In all cases, nodes maintained their previous role in their new positions, with the only exception being that of M_{13} has became a gateway node and has thus renamed to G_{16}.

Assuming that source node *S* wishes to transmit asynchronous, secure, burst, and energy efficient traffic types, at a sequence and iterations, as shown in Table 5.50.

Table 5.50: The sequence, iterations, and traffic type required for transmission by node S

Traffic scenario	Sequence	Iteration	Requirements
Asynchronous	1, 3	2	Low, Medium-high
Secure	2	1	Average
Burst	4	1	Medium-high
Energy efficient	5	1	High

Also, assuming that source node *S* maintains the same routing information as before the nodal movements, and therefore is totally unaware of the changes performed in the network. Then, the source node initiates an asynchronous communication over route R_2 as remembered in the previous scenario, which is:

- **$S \rightarrow R_1 \rightarrow G_1 \rightarrow R_2 \rightarrow G_5 \rightarrow R_5 \rightarrow G_{11} \rightarrow R_8 \rightarrow G_{15} \rightarrow R_{12} \rightarrow D$**
  PM[S], PM[R1], PM[G1], PM[R2], PM[G5], PM[R5], PM[G11], PM[R8], PM[G15], PM[R12], PM[D]

According to *S*'s information, the most optimal routes for asynchronous traffic with low requirements are R_2 and R_6. However, according to *S*'s memory in routes utilisation, R_6 has been previously overused by *S*'s transmissions and therefore *S* decides to transmit over R_2. However, G_{15} along route R_2 is no longer reachable, and will thus create an error when network traffic reaches node R_8, as there is not a direct link between these two nodes. Although R_8 now knows that it can reach R_{12} through G_{16} (through NNT broadcasts) and thus could dynamically alter the route, it will have to discard the data and transmit a route error (RERR) back to the source. The reason of doing so is because R_8 does not know the QoS required by the source, or the QoS offered by G_{15} for this particular routing scenario. Once the source node *S* receives a RERR packet it has two options:

- Immediately resume transmission over an alternative route which offers the same QoS (R_6 as remembered by S).
- Initiate a new route discovery request for destination node D.

As a general rule, any node which knows an alternative route that is offering the same QoS as the previously unavailable route, always resumes its communication through the alternative route, if, and only if, the transmitting node has not received a RERR packet twice for the same destination. In this way, the network transmission overhead from route request packets is minimised. However, if the node has no alternative route in its routing tables that it could use to reach destination D with the required QoS, the node could then initiate a new route discovery request for destination node D.

In this particular scenario, node S retransmits the information over the alternative route R_6, which is remembered by S as:

- $\mathbf{S \rightarrow R_1 \rightarrow G_3 \rightarrow R_4 \rightarrow G_8 \rightarrow R_7 \rightarrow G_{13} \rightarrow R_{10} \rightarrow G_{15} \rightarrow R_{12} \rightarrow D}$
  PM[S], PM[R_1], PM[G_3], PM[R_4], PM[G_8], PM[R_7], PM[G_{13}], PM[R_{10}], PM[G_{15}], PM[R_{12}], PM[D]

This time the transmission is successful and thus the source node S continues with its second transmission which is secure traffic, with medium-high requirements, over the most optimal route remembered by node S, which is R_3, such as:

- $\mathbf{S \rightarrow R_1 \rightarrow G_2 \rightarrow R_3 \rightarrow G_6 \rightarrow G_9 \rightarrow R_6 \rightarrow G_{12} \rightarrow R_9 \rightarrow G_{15} \rightarrow R_{12} \rightarrow D}$
  PM[S], PM[R_1], PM[G_2], PM[R_3], PM[G_6], PM[G_9], PM[R_6], PM[G_{12}], PM[R_9], PM[G_{15}], PM[R_{12}], PM[D]

Yet again, the transmission causes an error at node R_3 due to the fact that R_3 cannot find the next node in the transmission chain (node G_6). Thus, R_3 drops the data and replies back to the source with a RERR packet. Once the RERR packet is received, node S requests a new route from cluster-head R_1. Assuming that R_1 does not have a route, node S is forced to initiate a new route discovery for destination node D. This is possible at this stage, as node S has failed twice to reliably transmit network traffic to D. This process results in the following routes being identified:

1. $\mathbf{S \rightarrow R_1 \rightarrow G_1 \rightarrow R_2 \rightarrow G_4 \rightarrow R_5 \rightarrow G_{11} \rightarrow R_8 \rightarrow G_{16} \rightarrow R_{12} \rightarrow D}$
   PM[S], PM[R_1], PM[G_1], PM[R_2], PM[G_4], PM[R_5], PM[G_{11}], PM[R_8], PM[G_{16}], PM[R_{12}], PM[D]
2. $\mathbf{S \rightarrow R_1 \rightarrow G_1 \rightarrow R_2 \rightarrow G_5 \rightarrow R_5 \rightarrow G_{11} \rightarrow R_8 \rightarrow G_{16} \rightarrow R_{12} \rightarrow D}$
   PM[S], PM[R_1], PM[G_1], PM[R_2], PM[G_5], PM[R_5], PM[G_{11}], PM[R_8], PM[G_{16}], PM[R_{12}], PM[D]
3. $\mathbf{S \rightarrow R_1 \rightarrow G_2 \rightarrow R_3 \rightarrow G_9 \rightarrow R_6 \rightarrow G_{12} \rightarrow R_9 \rightarrow G_{15} \rightarrow R_{12} \rightarrow D}$
   PM[S], PM[R_1], PM[G_2], PM[R_3], PM[G_9], PM[R_6], PM[G_{12}], PM[R_9], PM[G_{15}], PM[R_{12}], PM[D]
4. $\mathbf{S \rightarrow R_1 \rightarrow G_2 \rightarrow R_3 \rightarrow G_7 \rightarrow R_6 \rightarrow G_{12} \rightarrow R_9 \rightarrow G_{15} \rightarrow R_{12} \rightarrow D}$

PM[S], PM[R_1], PM[G_2], PM[R_3], PM[G_7], PM[R_6], PM[G_{12}], PM[R_9], PM[G_{15}], PM[R_{12}], PM[D]

5. **S → R_1 → G_3 → R_4 →G_8 → R_7 → G_6 → R_{10} → G_{15} → R_{12} → D**

   PM[S], PM[R_1], PM[G_3], PM[R_4], PM[G_8], PM[R_7], PM[G_6], PM[R_{10}], PM[G_{15}], PM[R_{12}], PM[D]

6. **S → R_1 → G_3 → R_4 → G_8 → R_7 → G_{13} → R_{10} → G_{15} → R_{12} → D**

   PM[S], PM[R_1], PM[G_3], PM[R_4], PM[G_8], PM[R_7], PM[G_{13}], PM[R_{10}], PM[G_{15}], PM[R_{12}], PM[D]

7. **S → R_1 → G_3 → R_4 → G_8 → R_7 → G_{14} → R_{10} → G_{15} → R_{12} → D**

   PM[S], PM[R_1], PM[G_3], PM[R_4], PM[G_8], PM[R_7], PM[G_{14}], PM[R_{10}], PM[G_{15}], PM[R_{12}], PM[D]

Node *S* can recalculate the capability/incapability determination for each route and objective combination using the information above, as shown in Table 5.51. Then, node *S* can produce the average, minimum and maximum, and standard deviation values for the new routes R_1 - R_7, as shown in Tables 5.52 and 5.53. Finally, node *S* can deduce the final metric for each route and objective combination, as shown in Table 5.54.

Table 5.51:
Capability/Incapability determination for the new routing information resulted from the nodal movements

	Energy	**Synch**	**Asynch**	**Critical**	**Secure**	**Burst**
Route 1	Capable	Incapable	Capable	Capable	Incapable	Incapable
Route 2	Capable	Incapable	Capable	Incapable	Incapable	Incapable
Route 3	Capable	Capable	Capable	Capable	Capable	Capable
Route 4	Capable	Incapable	Capable	Capable	Incapable	Incapable
Route 5	Capable	Capable	Capable	Capable	Capable	Capable
Route 6	Incapable	Incapable	Capable	Incapable	Incapable	Incapable
Route 7	Capable	Capable	Capable	Capable	Incapable	Incapable

Table 5.52: New information for each route and objectives O_1 - O_3 due to mobility

	Energy efficient				**Synchronous**				**Asynchronous**			
	AV	MIN	MAX	SD	AV	MIN	MAX	SD	AV	MIN	MAX	SD
R_1	14	23	8	11	14	26	7	13	15	28	7	15
R_2	15	28	8	14	15	31	7	17	17	48	7	29
R_3	11	17	8	6	10	14	7	5	10	15	7	6
R_4	14	23	8	11	11	26	7	13	12	28	7	15
R_5	10	12	8	3	8	10	7	2	9	11	7	3
R_6	13	39	8	22	11	31	7	17	12	37	7	21
R_7	11	22	8	10	10	23	7	11	11	28	7	15

Table 5.53: New information for each route and objectives O_4 - O_6 due to metric variations

	Critical				**Secure**				**Burst**			
	AV	MIN	MAX	SD	AV	MIN	MAX	SD	AV	MIN	MAX	SD
R_1	9	17	4	9	20	46	6	28	21	45	7	27
R_2	12	49	4	32	21	53	6	33	21	48	7	29
R_3	6	9	4	4	9	14	6	6	11	18	7	8
R_4	7	17	4	9	10	46	6	28	13	45	7	27
R_5	5	6	4	1	7	9	6	2	8	10	7	2

R_6	7	21	4	12	8	17	6	8	10	22	7	11
R_7	6	16	4	8	11	45	6	28	12	45	7	27

Table 5.54: Final route metrics for each objective, taking into account the metric changes

	Energy	Synch	Asynch	Critical	Secure	Burst
R_1	Good	N/A	V. Good	V. Good	N/A	N/A
R_2	Good	N/A	Poor	N/A	N/A	N/A
R_3	V. Good	V. Good	Excellent	V. Good	Good	Good
R_4	Good	N/A	V. Good	V. Good	N/A	N/A
R_5	Excellent	V. Good	Excellent	Excellent	Good	V. Good
R_6	N/A	N/A	Poor	N/A	N/A	N/A
R_7	V. Good	Poor	V. Good	V. Good	N/A	N/A

Node *S* erases from its memory the sequence of previously used routes to destination node *D*. This is because node *S* has built new routing information, and thus it is memorising the sequence of transmissions from the beginning. Thus, since node *S* has not maintained previous knowledge of its transmission sequence, it will transmit secure traffic, as defined in Table 5.50, over either R_3 or R_5 since both exactly match node's requirements. In this scenario the route is randomly selected to be R_3. The following transmission is asynchronous traffic with medium-high requirements, and thus the *best* routes are either R_1 or R_4 or R_7. Route R_1 was randomly selected among these options. For burst traffic with medium-high requirements there is only one route that exactly satisfies node's requirements, which is route number R_5. Similarly, for energy efficient network traffic with high requirements, route R_5 is the only *optimal* route that fully satisfies node's requirements. The route selections performed by node *S* are:

- Network traffic secure, Sequence 2, Iteration 1, Requirements Average, Route R_3.
- Network traffic asynchronous, Sequence 3, Iteration 2, Requirements Medium-high, Route R_1.
- Network traffic burst, Sequence 4, Iteration 1, Requirements Medium-high, Route R_5.
- Network traffic energy efficient, Sequence 5, Iteration 1, Requirements high, Route R_5.

5.6.8 Outcomes and evaluation of the MARIAN-enabled ad-hoc network application scenario

This section demonstrated the most intriguing aspects of MARIAN's routing through the use of a number of routing scenarios. Specifically, this section demonstrated the clustering formation process, the reactive and proactive route and network discovery processes, and highlighted the routing decisions taken by a source node in relation to gathered rout-

ing metrics. The routing scenarios include: a stable network topology; a network topology with devices' varying routing metrics; and a network topology with nodal movements. In all cases, the strength of the routing protocol to adapt with various routing conditions, such as mobility and changes in routing devices' critical elements, is thoroughly demonstrated. In addition, network overhead imposed by reactive route discovery and proactive network discovery processes, has been calculated and presented for the defined network topology.

6 Evaluation

6.1 Research findings discussion

This chapter summarises the research findings and evaluates their significance. Initially, the novelties that this research is claiming to have achieved are presented and backed-up with results derived from the experimentation and modelling phases. Then, the successful culmination of this research is justified by an overall discussion and by referring to resulting publications in major journals, IEEE-level conference proceedings, and symposiums papers. In addition, the work performed for the scope of this PhD is compared to other related work and evaluated. Specifically, strong evidence on how this research work improves on standard methods and on how it may be used to extend these is provided.

6.2 Novelties justification

This section presents the novelties that have been achieved by this research and provides appropriate justification by referring to the proposed model and most importantly the experimentation results. For this purpose, the following sub-sections (6.2.1 - 6.2.7) present the appropriate evidence in an attempt to confirm that these novelties fulfil the proposed aims (*see* Section 1.5, Migas, N., et. al., 2003a.).

6.2.1 Maximised network performance

The proposed routing protocol is able to retrieve multiple redundant routing paths through its reactive route discovery and proactive network topology gathering processes, as specified in Sections 3.6.1 and 3.6.2, and demonstrated in Sections 5.6.2 and 5.6.4. Each retrieved routing path is guaranteed to be distinct due to the careful and precise modelling of the reactive and proactive route and topology discovery algorithms (*see* Algorithms 3.2 - 3.7), and loop-freedom due to the fact that the underlying protocol (CBRP), which MARIAN bases its fundamental functionality on, is also loop-free. Specifically, a node never forwards the same RREQ or RREP packet twice to another node, since the history list of the visited nodes for this particular packet is maintained in the packet itself as well as in the forwarding node. In the same fashion, a network topology mobile agent never visits the same node twice while *Exploring* or *ReturningHome* (*see* Section 3.6.2). This approach also applies to the multiple redundant routing paths discovery, since each cloned packet/agent is handled individually.

This is achieved by utilising an identification number that is unique for each original or cloned packet/agent, and thus assists in forwarding/migrating decisions making, respectively. This way, a node/agent is further protected from undesirable visitations/migrations that might result because of mobility or any other unexpected environmental or parameter changes.

Multiple redundant routing paths maintained at a node's routing cache and at a node's cluster-head routing tables can be especially useful in maximising the network's performance, since a node can utilise multiple routes for its data transmission instead of a single route. Therefore, the load is distributed evenly among each intermediate node along each available routing path, and thus single nodes are not overburdened with frequent routing requests. MARIAN's design allows a source node to transmit different types of traffic along multiple paths, and as a result of that, routing is performed virtually in parallel. This may be correlated to the ability of a multi-processor device to assign each distinct task to each processing unit, and thus perform these in parallel. However, the analogy is not so accurate, since retrieved routing paths are likely to be composed of one or more common nodes, and therefore routing is unlikely to be thus efficient.

MARIAN's strongest element that significantly enforces its ability of multiple redundant routing paths utilisation is its ability to determine the routes' appropriateness in relation to the type of traffic that is to be routed. As shown in Sections 5.1 - 5.3, the routing fitness of devices with different hardware and software characteristics, as well as devices with different current utilisation status, significantly varies (Migas, N., et. al., 2005, Buchanan, W., el. al, 2004a). Therefore, it makes little sense if a source node is allowed to decide on routing paths utilisation at a random basis, since this would probably result in a totally inefficient scenario, as the various data traffic specific requirements as well as the intermediate devices' throughout capabilities are not taken into account.

In order for this issue to be resolved, MARIAN assists a source node by providing routing capability determination criteria for each retrieved routing path on which the source can be based on, in order to efficiently utilise these paths. Specifically, determination criteria incorporate information concerning the available throughput that intermediate devices along a routing path can provide, which is multiplexed with other factors, such as the devices' utilisation status, network error, and so on (*see* Section 3.13). Therefore, a source node can quantify the efficiency of each retrieved routing path, and thus transmit high requirements network traffic through highly capable routes, while low network traffic through less capable routes. This way, the network gets fully utilised in a totally distributed and efficient manner, resulting in an overall maximised network performance. This was demonstrated in the provided routing scenario (*see* Section 5.6.5), where a source node successfully determined the

routing capabilities of each retrieved route, and transmitted its network traffic through the appropriate one, e.g. asynchronous network traffic with low requirements, through route R_1, while synchronous network traffic with medium-high requirements, through route R_4.

Another novelty that has the potential of maximising the network's performance is the agents' capability of intelligent filtering. As specified in Section 3.6.2, network topology gathering mobile agents are required to filter collected data in order to remove unnecessary redundant routing information, and thus help themselves to become lighter in terms of migration requirements. This property, once used efficiently, can significantly minimise the network load when compared to standard static agent approaches. As shown in the data gathering database application scenario (*see* Section 5.4.2), the mobile agent with intelligent filtering approach significantly improves upon the standard static agent approach, when the data volume is large (Migas, N., et. al., 2004a). Similarly, intelligent filtering network topology gathering agents can significantly minimise the information carried with them without compromising it, and therefore compensate for their high migration time requirements, or even improve upon the static agent reactive approach in situations where the ad-hoc network is large, and perhaps mobility is low. This thesis is not concerned in providing detailed information on the specifics of the intelligent filtering methods, as this can significantly vary according to implementation.

In terms of reducing latency, MARIAN provides a concrete solution. The protocol's novelty lies on the utilisation of a combination of the reactive route discovery and proactive network discovery approaches. As defined in Section 3.6.2, network discovery mobile agents update the cluster-heads that created them, in a periodic fashion, with routing information concerning the whole network's topology as well as the routing metrics involved. As a result, cluster-heads are well informed most of the time with fresh routing information for each possible destination, which is held and maintained in local routing tables. A cluster-head's registered nodes can significantly benefit from this functionality as they can on-demand request routing information from their own cluster-heads in the form of a minimal propagation RREQ, and receive it through one or multiple corresponding minimal propagation RREPs (*see* Section 3.6.1). The benefit lies on the rapid request and reply times involved with minimal propagation packets, as in principle, the propagation distance is always one hop. Therefore, registered nodes can rapidly retrieve on-demand routes and hence minimise latency.

6.2.2 Increased scalability

MARIAN organises the network into a hierarchical structure, similarly to CBRP (Jiang, M.,

et. al., 2001). As previously mentioned (*see* Section 3.5.1), a clustering structure is imposed to the network, which results in grouping each device to one or more adjacent or disjoint clusters, where each device has certain responsibilities according to the role that it is given. Nodes with the highest responsibilities are cluster-heads, which maintain intra- and inter-cluster routing information and also provide routing services. In the contrary, member nodes have very limited responsibilities, which are only concentrated on periodically broadcasting information necessary for the clustering formation and maintenance processes that is a requirement for every participating node. Therefore, the clustering structure can be efficiently utilised to minimise the route discovery packets' propagation as well as the network discovery mobile agent migrations, since they are always forwarded and migrated, respectively, along a repeated sequence of alternating cluster-head and gateway node pair(s).

As a consequence, the network becomes more scalable, and can thus accommodate and cope better with increased number of participating devices than standard flat routing protocols. This is evident in Table 2.4, which shows that CBRP's time and communication complexities for route discovery are far less than other routing protocols of the same category. Since MARIAN's clustering formation is based on CBRP's, MARIAN inherits the benefits mentioned above.

As previously stated, MARIAN's clustering formation process is based on a variation of the standard lowest-ID algorithm, which utilises cluster-head metrics (CH_M) for clustering formation and maintenance instead of meaningless IDs. The cluster-head metric is a mixture of various factors, such as the devices' mobility patterns (Basu, P., et. al., 2001), buffering capabilities, throughput, network error percentage, utilisation status, battery level, and so on (*see* Section 3.1). Therefore, nodes that are less mobile and have a lower utilisation status, higher battery capacity, and so on, are more likely to become cluster-heads. In this fashion, re-clustering should occur less frequently due to occasional cluster-heads movements. As a result, the network overhead often involved with re-clustering should be minimised, and therefore the network should become even more scalable. In addition to mobility criteria for cluster-head selection, this research work extends the one presented in (Basu, P., et. al., 2001) by incorporating other equally important factors, which were mentioned above. Therefore, cluster-heads are deliberately chosen to be the fittest devices in terms of processing capacity, battery level, and network ability, in order to allow services to be offered more efficiently and further extend their lifetime, resulting in an overall highly scalable solution.

Furthermore, mobile agents are inherently distributed, and thus they provide a totally distributed solution, which adapts to the increasing and decreasing network population. For instance, a smaller ad-hoc network consisting of only a few cluster-heads will be required to transmit a significantly smaller number of data gathering agents, at any given time, than a

larger ad-hoc network possibly consisting of a couple of hundred cluster-heads. However, the overall network overhead percentage imposed at each link because of agent migrations should be approximately equal for any ad-hoc network size. As shown in Sections 5.6.2 and 5.6.4, approximately 7% was consumed by each link for agent migrations, while approximately the same percentage was consumed by each link for route discovery packets propagation. These figures are expected to be relatively constant for any ad-hoc network size and structure due to agents' inherent characteristics and the employment of the clustering structure. Therefore, this can further enhance network scalability.

6.2.3 Dynamic in nature

The cluster-head and routing metrics are based on standard performance tests executed in advance, as well as on varying parameters, such as mobility and battery level, respectively (*see* Section 3.10). Therefore the metrics are not fixed, and may considerably vary as these parameters change. Specifically, MARIAN defined a point to infinitive value (∞) which is allocated to preliminary metrics that have reached a critical level. For instance in case that the battery level of a device drops below a certain threshold, the battery preliminary metric points to infinitive, which results in the overall cluster-head and routing metrics to point to infinitive as well. Therefore, in case that a critical change occurs in either a cluster-head or routing device that caused the overall cluster-head or routing metric to point to infinitive, the device discards its role and moves to 'undecided' state (*see* Section 3.13). In this fashion, network nodes are protected from running out of vital resources, and thus the routing protocol dynamically adapts to devices' critical changes.

The routing scenarios presented in Sections 5.6.6 - 5.6.7 demonstrate an example of the protocols dynamic adaptation in terms of varying preliminary metrics and mobility, respectively. According to results presented in the first scenario, the source node got informed about critical changes in the intermediate devices' vital resources by means of disseminated metric updates. Specifically, each cluster-head is responsible for identifying sudden preliminary metric changes in its registered devices. As previously mentioned (*see* Section 5.6.6), this is achieved by constantly monitoring broadcasted NNTs and comparing the previously received metrics to new ones. In case a sudden change is identified, it constructs a metric update packet and broadcasts it. Receiving devices update their information and re-broadcast the packet. In this fashion, a source node currently utilising the forwarding services of the node undertaken critical changes, re-calculates the routing metrics of each route in which that node is found and consequently rejects the route. The example provided (*see* Section 5.6.6) demonstrates the ability of the routing protocol to dynamically adapt to such changes

and rapidly respond by rejecting the utilisation of the problematic routes.

Similarly, the mobility scenario demonstrates a dynamic adaptation case in which a source node gets informed of intermediate nodal movements by means of a RERR message. Specifically, a node along the route which realises that the incoming packet's next hop is unavailable, constructs a RERR packet, specifying the unavailable hop, and transmits it back to the source. Upon receiving two consecutive RERR packets for the same destination, the source node has to initiate a new minimal or full propagation RREQ. This demonstrates that the routing protocol is highly capable of rapidly responding to topology charges due to mobility.

Another example of the protocols dynamic adaptation can be found in the network discovery mobile agent process (*see* Algorithm 3.6 - 3.7). Accordingly, mobile agents that are *discovering* or *returning home* can dynamically alter their itinerary in case of the destination being unreachable, after waiting for a predefined amount of time. This problem could occur in situations with high mobility. A mobile agent can dynamically find an alternative path by examining the current visited node's NNT, 2-hop NNT, and NCT and utilise it in order to reach the desired destination. If the agent succeeds, it dynamically alters its previously stale routing information with this new one, otherwise it kills itself. There is also a scope to allow mobile agents to initiate reactive route discovery processes in order to find a viable routing path that leads to the desired destination. However, this was omitted by the current specification because of concerns that this could lead to increased overhead in situations with high mobility.

6.2.4 Quality of Service (QoS)

This is probably the most important novelty that this research has achieved. QoS is non-trivial in highly dynamic environments, such as in multi-hop ad-hoc networks. In order for a network to provide QoS, there must be a way of calculating the quality offered by each available routing path along with estimating the requirements that are imposed by different types of traffics as well as the specific application requirements. For instance, a user application at node *A* might require to transmit synchronous traffic to another application at node *D*, which is 4 hops away. Assuming that the QoS required by the application at node *A* is high and further assuming that there are 3 routes that can reach *D*, in total. MARIAN would initially require from node *A* to calculate a capability/incapability determination for each route and in relation to the type of traffic required for transmission. This calculation would be based on the preliminary metrics (*see* Table 3.14) associated with each node along each route, and in relation to the weighting system and desired range for synchronous traffic routing ob-

jective (*see* Table 3.17 - 3.18). Node *A* would then calculate a final metric (*see* Table 3.19) for each route that was found capable. Since the required QoS for this example was assumed to be high, node *A* would decide on routes that were found to be capable of providing excellent QoS for synchronous network traffic.

By default, MARIAN provides support for six routing objectives, including: energy efficient, synchronous, asynchronous, critical, secure, and burst traffic. Each objective was carefully designed by taking into consideration the different requirements imposed by each type and applying the weighting system (*see* Table 3.17), which has been deduced through experimentation. In particular, energy efficient traffic has high battery capacity requirements, while, in addition to that, synchronous has high buffering and low latency requirements. In contrast, asynchronous network traffic has low buffering requirements and no latency problems. Critical network traffic has low network error percentage requirements, while secure network traffic has high requirements in terms of processing power because of complex encryption and decryption algorithmic executions. Finally, burst network traffic has extremely high buffering requirements.

In addition, each objective was designed in such a way so as to be very sensitive to critical metrics variations. These include: battery and memory capacity, and CPU utilisation. Therefore, if a device is determined to be capable of accomplishing a certain set of routing scenarios now, but at some point in time, a number of critical elements change, e.g. the CPU utilisation gets increased by a high percentage, the capability/incapability determination is being recalculated and consequently the device gets excluded from its routing responsibilities. This is particularly useful, in that, the protocol can predict when participating devices are likely to become unavailable because of low remaining battery, high CPU utilisation, or high memory usage, and thus protects them by withdrawing all routing requests. However, when a device, that has suffered a critical change, e.g. its CPU utilisation has considerably dropped, returns back to normal, the protocol re-considers the device's routing fitness.

As shown in Section 5.5.2, simulation results proved that when key metrics are suddenly changed, e.g. the remaining battery drops, or the CPU is highly utilised, or the device is running low on available memory, the device immediately turns to incapable state of routing high requirements traffic types. In addition, it was shown that the variation of a device's overall metric was high in all cases, and thus this demonstrates the ability of the protocol to rapidly respond to critical changes (Migas, N., and Buchanan, W., 2005).

Furthermore, MARIAN's metric-driven clustering formation process takes into account various factors for electing the most suitable cluster-heads, such as network mobility, utilisation status, battery capacity, throughput error percentage, and so on (*see* Section 3.13). Therefore, it is highly probable, that in a large ad-hoc network with devices having various

hardware characteristics, the network's backbone will be consisting of high performance devices with low or no mobility, and thus the provided QoS can be significantly increased. This is due to the fact, that cluster-heads have vast responsibilities in the routing processes, and thus, selecting the most appropriate devices for this role can be significantly beneficial.

6.2.5 Enabled energy conservation

The most vital resource of a mobile device is its battery. Consequently, since ad-hoc networks are mainly comprised of mobile devices, battery is the most important factor. In order to allow energy conservation, the routing protocol must take into account the battery level of each device involved or is likely to be involved in the routing process. MARIAN calculates a battery preliminary metric (*see* Section 3.13) based on the current battery level of each device in the network and incorporates it to the overall cluster-head and routing metrics. According to the weighting applied to the cluster-head and each routing objective (*see* Table 3.17), the battery capacity always obtains a substantial amount when compared to other preliminary metrics. The only exception is the cluster-head objective in which battery obtains 30 (%) less than mobility. For example, although synchronous network traffic has high throughput requirements, battery obtains twice as much weighting as throughput, due to its vast importance in every routing scenario.

According to results presented in Sections 5.1 - 5.3, the battery discharge rate was shown to be significantly decreased (the battery was reducing at a faster pace) while the device was being used as a router, or, being used to perform complex tasks, and was also shown to vary in relation to the OS and JVM used. Therefore, MARIAN excludes devices with low battery capacity from taking the role of a cluster-head or taking active part in data routing, provided that there is another more charged device in the general vicinity. Therefore, MARIAN allows devices with low battery capacity to join an ad-hoc network and benefit from its services, without exhausting their limited battery capacity.

Furthermore, MARIAN provides an explicitly defined energy efficient routing objective, where battery capacity obtains ten times more weighting than throughput, fifty times more weighting than complex algorithmic calculation ability, and so on (*see* Section 3.13). This scenario was specifically designed to provide energy efficient routing, rather than deliver data within certain time frames. In particular, a source node wishing to send its network traffic along an energy efficient routing path, in principle selects the path that consists of the most devices that do not rely on battery power for their operation, or alternatively consists of devices that have their batteries fully charged. Therefore, an ad-hoc network operating on an energy efficient principle should be able to significantly extent the participating devices' life-

time, and consequently its own lifetime.

6.2.6 Improved reconfigurability

In a pure agent-based implementation of the protocol's specification (*see* Section 3.2), where every entity is deployed as either a stationary or a mobile agent, reconfigurability can be easily achieved by on-demand reconfiguration agents. Specifically, authenticated mobile agents carrying protocol updates in their payloads can be dynamically dispatched in the ad-hoc network in order to automatically update or replace certain protocol components, or to reconfigure the protocol in such a way so as to adapt in various environmental changes.

The main advantage of this strategy is that it eliminates the need of manual update installations. Particularly, in a routing scenario implemented with traditional mechanisms, one would have to gather every mobile device in the network and update it separately, while the devices should have to be cut-off from the network. This approach may not sound particularly difficult when dealing with a small-scale ad-hoc network, possibly consisting of a few dozens of devices, however, in a large-scale network with possibly a couple of hundred or even thousand of devices this is unrealistic. MARIAN provides the fundamental mechanisms to achieve efficient and effective reconfiguration (*see* Section 3.2), without needing to shut down the network or even reboot a single device. This lies in the agent's specific characteristics, such as ease of installation and removal, agent communication, and mobility. A few agent-based reconfiguration examples include but are not limited to the following.

A recently developed mobile agent is dispatched to replace an existing stale clustering agent. The agent is dispatched to the nearest node, and finds its way to the cluster-head. Once the agent arrives, it clones itself and leaves for the next cluster-head. The cloned agent delivers the credentials of its developer and its version number to the guard stationary agent. The guard agent then authenticates the mobile agent and compares its version to the version of the current clustering agent. If the authentication is successfully completed and the version number is the latest, the guard agent kills the previous clustering agent, and installs the newer one. In this fashion, the up-to-date mobile agent could visit every single cluster in the network, and leave its clone. Updates are then disseminated from the cluster-head towards its members.

Another example is an agent carrying a routing metric calculation update. For instance, the update provides a better way of calculating the routing metric of a device required to route asynchronous network traffic. The agent can be dispatched and authenticated at each cluster in the same way as the agent in the previous example. Then, the agent can deliver the update to the metric agent by message exchange.

Finally, an agent could carry an update towards the frequency of network discovery agent initiations performed by each cluster-head. In addition, a new propagation limit may be specified in the update. For instance, assuming that the cluster-heads by default initiate such a process every fifteen seconds, and the span covers the whole network, these measurements are inefficient because of high mobility experienced by the network nodes, at a particular instance. Thus, the mobile agent could instruct the cluster-heads to delay the initiation of this process or cut it off until instructed otherwise.

6.2.7 Improved security

Although MARIAN's specification (*see* Section 3.1) is not specifically tailored to provide strong security measurements, it provides the required infrastructure for enhancing security. By providing only the fundamentals, MARIAN aims to provide a balance between security and performance, and allow specific implementations to allocate the weighting according to the desired principle. For instance, in case of a strictly secure implementation, confidentiality, integrity, and availability of the network's resources need to be guaranteed at all times. This would, however, enforce strong encryption and authentication techniques, as well as the need for guaranteeing that routing information is kept private. In such an extreme scenario, large network overhead would be generated, in addition to high processing tasks imposed on special nodes. This may be impractical on large networks with high mobility patterns, or undesirable in networks that only require lighter forms of security.

The clustering formation process (*see* Section 3.5.1) was designed to be fairly open, as participating devices are required to frequently broadcast their cluster-head metric (CH_M) as well as their test results to other devices in their general vicinity. Although this information could be encrypted and decrypted at the receiving nodes in order to provide confidentiality, this would result in significant clustering maintenance overheads, especially in situations were clusters need to be constantly reformed because of high mobility. Therefore, security at this level was omitted.

The reactive route discovery and proactive topology information gathering approaches were designed in such a way so as to provide appropriate levels of security. A source node requesting a routing path by either utilising the minimal RREQ or the standard RREQ propagation method is limited on the frequency and iterations that these are invoked for a single destination (*see* Section 3.6.1). Specifically, if the node fails to receive a corresponding RREP it enters a backoff algorithm before initiating the same process again. Therefore, cluster-heads are protected from receiving uncontrolled numbers of RREQs, and thus availability is ensured without compromising performance, as the failed reception of a RREP implies

that such a route is unavailable, at this instance. Along the same fashion, a cluster-head cannot initiate a topology discovery agent before the time threshold is reached or a collection of certain triggered events occur. Therefore, special nodes are also protected from frequent mobile agent visitations that generally require their services, and thus availability is ensured without compromising performance, as frequent agent migrations would by themselves compromise performance because of their high requirements in terms of processing power and throughput.

Furthermore, the proactive approach was designed in such a way so as to provide confidentiality at a certain extent (*see* Sections 3.10 - 3.12). Specifically, a network discovery mobile agent is denied direct access to a node's database by a stationary guard agent, whereas the legal form of accessing the node's information is by contacting the librarian agent (*see* Section 3.12) and requesting the desired information. In order to guarantee that information is not passed to malicious external objects or agents, the requesting entity is required to authenticate itself to the database agent. Data gathering agents can then encrypt the information obtained and decrypt it once arrived to their originating platforms. However, this setting has not been included in MARIAN's specification due to the large processing overheads involved with frequent data encryptions and decryptions.

As specified in Section 3.8, in addition to the static routing approach, mobile agents can also be used to route data by appending them to their payload and travelling along the specified route in order to deliver them to the destination. Mobile agents can be used to guarantee communication confidentiality between two or more parties, by initially distributing the public keys of the users who are willing to participate in confidential communications. Then, a routing agent can encrypt its user's message with the other user's public key and append the encrypted message to its payload. The agent can then deliver the message at the destination, where it can get decrypted with the corresponding private key.

6.3 Limitations of MARIAN

MARIAN was shown to have achieved a set of novelties, which are not present in the majority of existing ad-hoc routing protocols, and, thus, provides an enhanced solution for ad-hoc routing and automatic network reconfiguration. However, MARIAN has certain limitations:

- **Temporary routing-loops**. MARIAN may suffer from temporary routing-loops, as it bases the fundamental aspects of its route-discovery process on CBRP, which is known to produce temporary routing-loops. Although, these should be short-lasting, and should thus not significantly influence the performance of the routing protocol.

- **Mobile agents time overhead**. Although MARIAN's proactive network discovery approach can significantly minimise latency within a proactive routing-zone, it may involve long waiting-times for far-reaching zones (in terms of hops). Thus, a proactive routing zone should be defined small enough, to guarantee that mobile agents will retrieve the correct topology information.
- **Moderate mobility support**. As a consequence of the long migration times, which are involved with mobile agents, MARIAN's proactive network discovery process may not adequately cope with high nodal mobility. Thus, in scenarios with high mobility, the proactive network discovery process should be either switched off, or, the routing zones spectrum should be minimised, as such as to allow the discovery of the neighbouring cluster-heads of the cluster-head's neighbouring cluster-heads.
- **Critical nodes**. Although MARIAN uses a metric-driven approach to clustering formation, which ensures that fitter nodes with less mobility will be elected as cluster-heads, it cannot guarantee that this will always be the case, as it is possible for participating devices to have similar routing strength and mobility patterns. In this case, cluster-heads may become bottlenecks, and, thus, unfairly exhaust their battery reserves and consume their resources, where other nodes benefit from the cluster-head's services.
- **Interoperability**. At present, MARIAN does not offer interoperability with other non-agent-based and agent-based routing protocols, as interoperability was left for future development.
- **Security**. Although MARIAN provides the fundamental infrastructure for ad-hoc routing security, at present, it does not fully support security, as it was left for future development.

6.4 Comparison of this work with other related research

As previously mentioned, the main aim of this research work was to develop a concrete solution for routing in multi-hop ad-hoc networks. For this purpose, a novel hybrid routing protocol, named MARIAN, was specifically designed in order to maximise network performance, increase scalability, respond dynamically to changing factors, guarantee QoS, be energy efficient, reconfigurable, and provide the basic infrastructure for security. As previously discussed (*see* Section 6.2.1 - 6.2.7), each aim set by this research was fully met, and justified. The most important outcomes from the modelling and experimentation phases were published in major journals, IEEE-level conference proceedings, and further presented in the recent BCS Symposium on Artificial Intelligence, which were further published in the Expert Update journal. This section compares these outcomes with traditional and agent-based

methods in this area, and provides concrete evidence on the improvements that this research work has achieved.

MARIAN bases its clustering formation process on the lowest-ID algorithm (Gerla, M. and Tsai, J. T.-C., 1995) and further extends it. Initially, the LCC proposed in (Chiang, C.-C., et. al., 1997) has been adopted in order to enforce the least cluster-head changes, and thus minimise overhead involved with frequent re-clustering. In addition, MARIAN employs the mobility scheme presented in (Basu, P., et. al., 2001) for cluster-head elections and enhances it with other equally important parameters. Specifically, the work proposed in (Basu, P., et. al., 2001) proved with the aid of simulations that, when devices' mobility patterns are taken into consideration in the clustering formation process, cluster-head changes can further decrease by 30% when compared to the LCC enhanced, lowest-ID algorithm. However, their work does not take into account the capabilities or utilisation status of the devices involved in the elections. Accordingly, a device with low remaining battery, high CPU utilisation, or low available memory, can be elected as a cluster-head, which is obviously undesirable. Therefore, MARIAN multiplexes mobility criteria with performance and utilisation factors, as described in Section 3.1. In addition, MARIAN senses the devices' critical changes and dynamically responds by allowing cluster-heads which have undertaken critical changes to discard their roles and move to an *Undecided* state. Therefore, this research work employs a sophisticated cluster-head metric, which is highly dynamic and accurate.

As defined in Section 3.1, MARIAN is a hybrid routing protocol that utilises both stationary and mobile agents for route discovery and network topology gathering. The on-demand approach is similar to the one presented in (Jiang, M., et. al., 2001), however, MARIAN enhances it with support for multiple redundant routing paths discovery. Therefore, a source node gets informed of every possible loop-free routing path leading to a specific destination, which generally adds redundancy in such a way so as to allow the source to rapidly switch to an alternative route in case of primary failure. As a result, the source is not required to initiate a new route discovery, which would otherwise generate network overhead.

Route redundancy is also offered by several routing protocols, including DSR (Johnson D. B., et. al., 2004), TORA (Corson, S., 2000), ARA (Bouazizi, I., 2002), SLURP (Woo, S.-C. and Singh, S., 2001), and DST (Radhakrishnan, S., et. al., 1999), a full list is provided in Tables 2.1, 2.3, and 2.5. However, these protocols either provide no mechanisms for deciding on the optimal paths or they are based on the standard shortest path routing mechanism. However, this mechanism oversimplifies such a complex decision by ignoring the fact that participating devices may have considerably unequal performance characteristics and current utilisation status. Accordingly, it is possible for a *best* route to be composed of devices with

high utilisation status or low battery level, which results in an overall unreliable route. Along the same line, a routing path composed of less nodes that another, would be considered more efficient in terms of data delivery time, even though the former may be comprised of PDAs and the latter of powerful workstations. However, as shown in Sections 5.1 - 5.4, this is totally untrue, as PDAs can be up to 100 times less optimal than high-end devices.

MARIAN tackles this issue by providing an intelligent decision making mechanism, which takes into consideration various parameters (*see* Section 3.13) and applies distinct weighting to each one, according to the routing objective the device aims to accomplish. Therefore, this scheme protects routing devices from becoming network bottlenecks by taking into consideration key elements, and intelligently deciding on optimal routes based on accurate and diverse metrics, which are appropriately weighted to conform to the requirements imposed by each predefined routing objective. This is evident in the application scenario presented in Section 5.6, which demonstrated MARIANs full potential in optimal route determination, as well as its dynamic nature to adapt to metric variations (*see* Section 5.5) and nodal movements.

Furthermore, although a few routing protocols, such as FORP (Su, W. and Gerla, M., 1999), SSA (Dube, R., et. al., 1997), ABR (Toh, C., 1996), and DDR (Nikaein, N., et. al., 2000), use metrics for optimal route discovery, these are vastly incomplete, and they do not provide any adaptability to different routing scenarios' requirements.

The proactive network topology information gathering approach has been designed to enhance the on-demand route discovery approach in order to minimise latency, which is often involved with purely reactive protocols. As shown in (Marwaha, S., et. al., 2002), the combination of an on-demand route discovery approach with a proactive distributed topology discovery mechanism can significantly reduce frequent update disseminations, usually required by proactive protocols, and further reduce route discovery latency and end-to-end delays, usually found in reactive protocols. In contrast to the work presented in (Marwaha, S., et. al., 2002), MARIAN delivers full control to the cluster-heads which can responsibly define the propagation horizon of the network topology gathering mobile agents, as well as the frequency of their creation, according to the clusters' needs. Therefore, assuming that a cluster-head receives frequent minimal propagation RREQs from its members for several inter-cluster diverse destinations with low latency requirements, it can dynamically dispatch a network topology discovery agent, as it is likely to assist in reducing latency, since routing information is publicly accessible to its local cluster. Similarly, a cluster-head that has not heard any such activities from its members over a certain period of time, can suspend the agent migrations until a certain triggered event has occurred or a time threshold has been exceeded.

In addition, network discovery mobile agents are equipped with intelligent filtering capabilities (*see* section 3.6.2), which were shown to significantly reduce network overhead (*see* Section 5.4.2), an aspect that was not covered by either (Marwaha, S., et. al., 2002), nor any other agent-based routing protocol (Anderegg, L. and Eidenbenz, S., 2003, Bandyopadhyay, S. and Paul, K., 1999). Furthermore, MARIAN provides security against potentially malicious agents by means of agent authentication. Specifically, a network topology gathering mobile agents is not allowed to directly access the node's database, whereas, the only legal form of data retrieval is by authenticating itself to the database agent, which, in turn, forwards the agent's query to the database and returns the results back to the gathering agent (*see* Section 3.12). MARIAN could be effectively extended to incorporate the agent-based security proposals found in (Ping, Y., et. al., 2004, Peysakhov, M., et. al., 2004). As discussed in Appendix B.10, the former proposed a totally distributed and scalable security solution for ad-hoc networks, which relies on a multi-agent system to provide functions similar to those of the body's immune system, while the latter suggested a build-in network awareness capability to mobile agents that would enable them to reason on whether a host has been compromised or not, and act based on this knowledge.

As previously discussed in Sections 3.7 and 3.8, MARIAN provides two alternatives for data packets routing, a static agent and a mobile agent approach. The former is similar to DSR (Johnson D. B., et. al., 2004), while the latter was specifically defined to add robustness to the protocols source routing mechanism. In both approaches, the source route is a requirement, and thus, in the first case, the complete route must be supplied to the packet's header, while in the second case, the complete itinerary must be supplied to the mobile agent on creation. However, in the mobile agent approach, the agent is allowed to dynamically alter its own itinerary, in case of unreachable next hop, often resulting because of nodal movements, and dynamically seek an alternative path. In this fashion, MARIAN extends DSR with robust source routing agents that can guarantee the successful delivery of information, however, this cannot be used as a primary routing method, because of the high migration timing requirements involved with mobile agents (*see* Section 5.4.2). Nevertheless, robustness can compensate for slow data routing, possibly under extreme circumstances, such as in situations where there is high mobility.

MARIAN's mobile agent source routing is rather similar to the agent-based scheme proposed in Bandyopadhyay, S. and Paul, K., 1999, which propose messenger mobile agents for ad-hoc routing, however, as a primary mechanism. Specifically, agents append communication data in their payload, autonomously navigate through the ad-hoc network, using an efficient routing protocol infrastructure, find the destination, and deliver the message. However, as shown in Section 5.4, mobile agent migration times are far greater than standard

data packets routing, and thus it is unrealistic to dispatch a mobile agent for every message that needs to be routed through an ad-hoc network. MARIAN resolves this issue, by defining the static agent approach always as a primary mechanism for source routing (*see* Section 3.7), whereas it defines the mobile agent approach as a secondary mechanisms, which can be used in situations where robustness of data delivery is necessary and fast delivery is not required.

The agent-based ad-hoc-VCG protocol (Anderegg, L. and Eidenbenz, S., 2003) takes a different approach to ad-hoc routing than MARIAN, in that it utilises selfish agents that accept payments for forwarding data for other agents, and thus uses cost as the primary metric, which represents the real costs of intermediate nodes for forwarding communication data. The protocol is financially cost-efficient, and thus guarantees that data packets are being routed along the most cost-efficient path. As a consequence, the optimal route is chosen to be the one that is composed of the most efficient routing nodes, that is, the nodes with high battery reserves, low utilisation status, high throughput, and so on, as these are likely to have low costs for forwarding data. Thus, ad-hoc-VCG maximises network performance, is dynamic in nature, and enables energy conservation, similarly to MARIAN, however, ad-hoc-VCG requires knowledge of the underlying topology, which inevitably creates a large overhead in the route discovery phase, whereas MARIAN does not requires this information. In addition, ad-hoc-VCG does not take into consideration the diverse requirements of different traffic types, and, thus, irrespectively, each type of traffic is routed along the most cost-efficient path, which reduces load-balancing.

Ant-AODV (Marwaha, S., et. al., 2002) is a hybrid routing protocol rather similar to MARIAN, as it combines an on-demand route discovery approach with a proactive distributed topology discovery approach using ant-like mobile agent. As previously discussed in Section 2.4, the normalised overhead of the Ant-AODV was shown to be slightly greater than AODV, however, it achieved the highest connectivity and fewer end-to-end delays, at a cost of extra processing of the ant messages, and a slightly higher overhead in occupying network capacity. However, their simulation was based on a constant mobile agent migration time, which was set to be very small, that contradicts the experimentation results that were presented in Section 5.4, which showed that mobile agents require approximately 1s to migrate from/to high-end devices, whereas they require 3s-5s to migrate from/to resource-constrained devices, thus, their original assumption was exaggerated. In contrast, MARIAN takes a more realistic approach to its proactive network discovery approach, where cluster-heads are responsible for defining the agent's horizon, that is, the zone that the cluster-head receives routing information proactively. Thus, MARIAN allows mobile nodes to benefit from reduced latency within their own zone, and, at the same time, reduces the long waiting times that are involved with routing zones equal to the complete, large ad-hoc network.

As previously discussed in Section 2.4, RoyChoudhury, R., et. al., 2000, propose a multi-agent based framework for topology discovery in wireless ad-hoc networks, which utilises the notion of *stigmergic communication*, *link stability*, and *information aging* that are used to assist a node with predicting the current network topology, based on the current network information stored at the node. This is achieved with a *recency* token, which reveals the freshness of routing information stored at each node. MARIAN is not using the concept of *recency* tokens, as, unlike the framework proposed in RoyChoudhury, R., et. al., 2000, each network discovery mobile agent, that is, the parent agent and its children, is an independent entity, and, thus, it does not cooperate with other network discovery mobile agents which are initiated at foreign cluster-heads. MARIAN's underlying idea of proactive network discovery is based on the fact that each cluster-head maintains up-to-date routing information for its own routing zone, and not the complete ad-hoc network, which would involve large network overhead and long waiting times.

A cluster-head, agent-based routing protocol, rather similar to MARIAN, was proposed by Denko, M. K., 2003. Specifically they propose the use of mobile agents for clustering formation, clustering maintenance, cluster size adjustment, re-clustering, continuous cluster status monitoring, and routing information collection. The disadvantage of their approach is that clustering and routing information is solely collected by mobile agents, and thus the long migration times involved contribute to an overall slow convergence scheme. In particular, the scheme could only cope with small ad-hoc networks with relatively slow mobility, whereas MARIAN can cope with large-scale ad-hoc networks with relatively moderate mobility, and large-scale ad-hoc networks with relatively high mobility, if the proactive network discovery approach is disabled.

Due to the small number of existing agent-based ad-hoc routing protocols, this section only compares MARIAN to the available ones, which are analysed in Section 2.4.

8 Conclusions

8.1 Chapter Overview - Conclusions

This chapter is the epilogue of this thesis. It summarises the most significant findings, which were derived from this research work, and rationally assess the success of this PhD. In addition it presents the future work.

8.2 Thesis epilogue

The research work presented in this thesis aims to provide a concrete solution to the problems involved with multi-hop ad-hoc routing, which generally include the lack of an infrastructure, the limited capabilities of participating devices, and the disoriented as well as mobile nature of the network. As shown throughout Chapter 2, and Appendix D, current routing schemes are incapable of providing the complete set of essential principles, such as maximum network performance, route redundancy, scalability, dynamic adaptability, QoS, energy conservation, reconfigurability, and security. The main reasons for this are the absence of a logical network structure, the unclassified performance, utilisation, and mobility characteristics of participating devices, and the lack of robust mechanisms in place. This research combines all vital properties into a novel hybrid multi-hop ad-hoc routing protocol, which is metric-driven, highly sensitive, dynamic, scalable, energy efficient, reconfigurable and secure.

In summary, MARIAN provides novel metric-driven methods, which can be efficiently utilised for state of the art clustering formation, reactive route discovery, proactive network topology gathering, and routing. MARIAN pays enough attention to the requirements imposed by various network traffic scenarios and appraises those in terms of the routing fitness of intermediate devices along each routing path. In this fashion, the retrieval of the best routing path is always guaranteed, and the QoS each path is able to offer, at any given instance, is always precisely calculated. In addition, it provides a clear separation between high-end and low end-devices, where fitter devices have increased routing responsibilities than resource-constrained ones. Therefore, routing tasks as well as clustering formation responsibilities are performed more efficiently, and most importantly a balance of the utilisation of devices is achieved. Devices' vital resources are often overlooked by most current routing protocols, whereas MARIAN strongly relies on these in order to take optimal routing decisions. Fur-

thermore, it is extremely sensitive in critical changes, such as low devices' remaining battery life, high utilisation and mobility patterns, and so on, and can thus dynamically adapt its routing strategy based on these criteria. Finally, the protocol's novel agent-based modelling principles resulted in a totally distributed routing scheme that has the ability to automatically reconfigure itself on real-time, and provide the ground for enhanced security and survivability.

Several publications have been achieved throughout this PhD, eight, in total, and one currently under the reviewing process. Two papers were concerned with the modelling phase, while the rest were concerned with the experimentation phases. Therefore, each phase presented in this thesis is published in major journals and IEEE-level conference proceedings, and thus well recognised as novel. This fact, as well as my personal opinion, leads me to conclude that this research work is novel and that it has totally achieved its goals.

8.3 Future work

Based on the outcomes that were derived from this research along with the publications achieved, it is safe to conclude that this research was successful. Currently, MARIAN offers a concrete solution to ad-hoc routing along with novel functionality which is beyond the existing standards (Abolhasan, M., et. al., 2004). MARIAN's hierarchical structure as well as its hybrid nature, coupled with the advanced mobile agent features and its multi-dimensional super fine sensitivity routing metrics makes it a state of the art routing protocol that provides maximised network performance, scalability, dynamic adaptation, QoS, energy conservation, reconfigurability, and fundamental security. Nevertheless, this work can be extended even further in many multiple ways. This section discusses the potential for future work and builds-up a case for a postdoctoral research.

This research has been based on experimental research, and therefore a bottom-up approach has been taken. Accordingly, each experiment was based on real hardware and software, with the only exception being the simulations that were conducted for the metrics sensitivity experiments (*see* Section 5.5). For this reason, MARIAN's specification has not been implemented nor tested in a discrete event simulator, such as *ns2*. Given that MARIAN's specification is comprehensive and includes innovative mechanisms, such as mobile agents and routing metrics, a wide-range experiment based on *ns2* would definitely be an excellent opportunity for a postdoctoral research. However, a challenging one, as ns2 would probably have to get modified in order to provide accurate support for simulating the network discovery mobile agent concept, in addition to defining a wide-range of devices with different performance characteristics and utilisation status, as well as implementing various

routing scenarios based on this information.

In addition to the simulation work, a large-scale real-world application scenario, similar to the one presented in Section 2.3, would provide excellent insight into the protocol's behaviour and performance. For this purpose, MARIAN would had to be implemented fully in various platforms, including PocketPC/Familiar Linux and Windows/Linux targeted for devices, such as PDAs and laptops, respectively. Unlike the outdoor experimentation study (*see* Section 2.3), this particular experiment would produce novel results in terms of the capability and performance of resource-constrained devices, which are considered to be the fundamental building blocks of ad-hoc networking. Although the complete MARIAN implementation in minimalistic platforms, such as PocketPC and Familiar Linux would have been a challenging issue, which would require intensive programming and state of the art software engineering principles, it could definitely be accomplished within a postdoctoral timeframe, since the fundamental aspects of this protocol have already been determined. The proposed experiment could be based on two mobility scenarios, particularly, the first having all regions stationary over a large geographical span, and the second having each node distributed and mobile. The experiments could be repeated with various changing factors, such as partially charged devices, medium or highly utilised, with low or extreme mobility, and so on.

As previously mentioned, the implementation part of this work was based on Grasshopper ME/SE mobile agent system. However, as proven in Section 5.4.2, Grasshopper's migration module is considerably slow, consuming just above 1 (s) for a mobile agent migration, and even worse, approximately three times more when an agent is transmitted from a PDA. In relation to results presented in Section 5.6.4, a network discovery mobile agent, originating from R_1 (*see* Figure 5.112), would approximately require 112s to collect the complete network topology. However, this could be totally unacceptable in scenarios with frequent nodal movements and constant changing factors. Therefore, a light-weighted mobile agent system especially targeted for multi-hop ad-hoc routing, with an optimised migration module, is necessary. Innovative work in this direction was presented in (Braun, P., 2003), where the author developed Tracy, a mobile agent system with an optimised mobile agent migration component, which has the ability to reduce migration times approximately by 60%, when compared to Grasshopper. Future research in this direction would be advantageous for a practical deployment of the network discovery mobile agent process.

Interoperability with fixed networks, as well as with ad-hoc networks operating under the principles of other multi-hop ad-hoc routing protocols, is an issue that perfectly fits into future work. Interoperability is an important issue, however, a challenging one, especially in the context of ad-hoc networking, and requires extensive research into standardisation proc-

esses by which distinct routing protocols could effectively communicate, possibly cooperate, and finally, offer fundamental services to each other. In this fashion, a MARIAN RREQ packet, once arrived at a DSR-enabled territory, could be translated into a corresponding DSR packet, processed, and returned back translated to a MARIAN RREP. Interoperability gets even harder, if one considers the vast number of proposed routing protocols for multi-hop ad-hoc networks. Therefore, this issue could well fit in a post-doctoral research.

Although MARIAN provides the basic infrastructure for enhanced security (*see* Section 6.2.7), an ad-hoc network is inherently open to external security attacks (see Section 2.1). Further research into prevention and detection mechanisms would be beneficial in enhancing the protocol's survivability properties. Novel research in this direction was presented in (Ping, Y., et. al., 2004) and (Peysakhov, M., et. al., 2004), and is described in Appendix B.10. According to both, there is scope for a totally distributed security architecture based on mobile agents, which can effectively protect a totally distributed and dynamic network. Their proposals could easily adapt with MARIAN, since the protocol provides the necessary infrastructure for mobile agent execution, migration, communication, and intelligent reasoning about the nodal behaviour. For example, security enforcement agents could read in test results maintained in nodal databases, and collectively reason about the integrity of each visited node. In this fashion, compromised nodes can be identified and security measurements can be enforced. These actions, for instance, may include: cutting out the node complete of communicating with the rest of the network, or rejecting its role, if applicable.

The research proposals, presented above, are just a few of many possible ways that this research could progress in the future. Other potential future work includes the definition of several more diverse routing objectives, the integration of application specific information that the protocol would automatically adapt by changing its internal structure, and fuzzy logic support for clustering formation. Another interesting approach would be to allow routing agents to negotiate exchanges of the workload delegated to them, in a similar manner as the proposal in (Urquhart, N., et. al., 2003).

References:

Abolhasan, M., et. al., 2004. A review of routing protocols for mobile ad hoc networks. Ad hoc Networks. Vol. 2. pp. 1-22.

Aggelou, G., et. al., 1999. RDMAR: A bandwidth-efficient routing protocol for mobile ad hoc networks. WOWMOM '99: Proceedings of the 2nd ACM international workshop on Wireless mobile multimedia. Seattle, Washington, USA. pp. 26-33.

Alaettinoglu, C., 1994. Design and implementation of MaRS: A routing testbed. Journal of Internetworking: Research and Experience. Vol. 5. No. 1. pp. 17-41.

Anderegg, L. and Eidenbenz, S., 2003. Ad hoc-VCG: a truthful and cost-efficient routing protocol for mobile ad hoc networks with selfish agents. MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking. San Diego, CA, USA. pp. 245-259.

Anderson, R. and Kuhn, M., 1996. Tamper Resistance - a Cautionary Note. Proceedings of the Second Usenix Workshop on Electronic Commerce. Oakland, California. pp. 1-11.

Aron, I. D., and Gupta, S. K. S., 1999. A Witness-aided Routing Protocol for mobile ad-hoc networks with Unidirectional links. Proceedings of the first international conference on Mobile Data Access (MDA). Hong-Kong, China. pp. 24-33.

Aron, I. D., and Gupta, S. K. S., 2000. Analytical comparison of local and end-to-end error recovery in reactive routing protocols for mobile ad hoc networks. MSWIM '00: Proceedings of the 3rd ACM international workshop on Modelling, analysis and simulation of wireless and mobile systems. Boston, Massachusetts, USA. pp. 69-76.

Artz, D., 2003. Network Meta-Reasoning for Information Assurance in Mobile Agent Systems. Eighteenth International Joint Conference on Artificial Intelligence. Acapulco, Mexico. pp. 1455-57.

Baldi, M. and Picco, G. P., 1998. Evaluating the tradeoffs of mobile code design paradigms in network management applications. Proceedings of the 20th International Conference of Software Engineering. Kyoto, Japan. pp. 146-155.

Bandyopadhyay, S. and Paul, K., 1999. Evaluating the performance of mobile agent-based message communication among mobile hosts in large ad hoc wireless network. MSWiM '99: Proceedings of the 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems. Seattle, Washington, USA. pp. 69-73.

Basagni, S., et. al., 1998. A distance routing effect algorithm for mobility (DREAM). MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking. Dallas, Texas, USA. 76-84.

Basu, P., et. al., 2001. A Mobility Based Metric for Clustering in Mobile Ad Hoc Networks. In International Workshop on Wireless Networks and Mobile Computing (WNMC2001). Scottsdale, Arizona, USA. pp. 16-19.

Baumann, J., et. al., 1998. Mole-concepts of a mobile agent system. World Wide Web. Vol. 1, No. 3. pp.123-37.

Bellavista, P., 2000. Protection and interoperability for mobile agents: a secure and open programming environment. IEICE Transactions on Communications. Vol. E83-B, No. 5. pp. 961-72.

Bellavista, P., et. al., 2001. Middleware services for interoperability in open mobile agent systems. Microprocessors and Microsystems. Vol. 25, No. 2. pp. 75-83.

Bellavista, P., et. al., 2003. Java for On-line Distributed Monitoring of Heterogeneous Systems and Services. The Computer Journal. Vol. 45. No. 6. pp. 595-607.

Bertsekas, D. and Gallager, R., 1987. Data Networks. Prentice-Hall, Inc. pp. 297-333.

Binder, W. and Roth, V., 2002. Secure mobile agent systems using Java: where are we heading?. Proceedings of the 2002 ACM symposium on Applied computing. Madrid, Spain. pp. 115-119.

Blackdown, 2004. Java - Linux. Available from <http://www.blackdown.org>. Visited 08/2004.

Bommaiah, E, et. al., 1998. AMRoute: Ad-hoc multicast routing protocol. Internet-Draft, draft-talpade-manet-amroute-00.txt. Work in progress.

Boppana, R. and Konduru, S., 2001. An adaptive distance vector routing algorithm for mobile, ad hoc networks. IEEE INFOCOM 2001 - The Conference on Computer Communications, no. 1. pp. 1753-1762.

Bouazizi, I., 2002. ARA - The Ant-Colony Based Routing Algorithm for MANETs. ICPPW '02: Proceedings of the 2002 International Conference on Parallel Processing Workshops. Washington, DC, USA. pp. 79.

Braun, P., 2003. The Migration Process of Mobile Agents - Implementation, Classification, and Optimization. PhD Dissertation, Friedrich-Schiller-Universität Jena, 315 pages.

Broch, J., et. al., 1998. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98). Dallas, USA. pp. 85-97.

Buchanan, W. J., 2000. Distributed Systems and Networks. Published by McGraw-Hill. ISBN: 0-07-709583-9

Buchanan, W. J., et. al., 2004a. Analysis of an Agent-based Metric-Driven for Ah-hoc, On-Demand Routing. Ad Hoc Networks. In Press, Corrected Proof. Available online 3 July 2004.

Buchanan, W. J., et. al., 2004b. Agent-based Clustering over Ad-hoc Networks. Expert Update. Vol 7, No 3. pp. 4-8.

Buchanan, W. J., et. al., 2005. Agent-based Forensic Investigations with an Integrated Framework, 4th European Conference on Information Warfare and Security (EICW). UK.

Cardi, G., et. al., 2000. Agents for information retrieval: Issues of mobility and coordination. Journal of

mobility and coordination. Vol. 46, No. 15. pp. 1419-1433.

Case, J., et. al., 1990. Simple Network Management Protocol. STD 15, RFC 1157, SNMP, Research, Performance Systems International, MIT Laboratory for Computer Science.

Ceruti, M. G., 2001. Mobile Agents in Network-Centric Warfare. IEICE Transactions on Communications. Vol. E84-B, No.10. pp.2781-5.

Chatzipapadopoulos, F. G., et. al., 1999. Mobile agent standards and available platforms. Computer Networks. Vol. 31. No. 19. pp. 1999–2016.

Chen, S. and Nahrstedt, K., 1999. Distributed Quality-of-Service Routing in Ad Hoc Networks. IEEE Journal on Selected Areas in Communications. Vol. 17, No. 8. pp. 1488-1505.

Chen, T.-W. and Gerla, M., 1998. Global State Routing: A New Routing Scheme for Ad-hoc Wireless Networks. IEEE International Communications Conference. Atlanta, GA, USA. pp. 171-175.

Cheng, C., et. al., 1989. A loop-free extended Bellman-Ford routing protocol without bouncing effect. Vol. 19. No. 4. ACM SIGCOMM Computer Communication Review. pp. 224-236.

Chess, D. M., 1998. Security issues in mobile code systems. Lecture Notes in Computer Science. Vol. 1419. pp. 1-14.

Chess, D., et. al., 1995. Itinerant Agents for Mobile Computing. IEEE Personal Communications. Vol. 2, No. 5. pp. 34-49.

Chiang, C.-C., et. al., 1997. Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel. Proceedings of IEEE Singapore International Conference on Networks (SICON). Singapore. pp. 197-211.

Chpudhury, R. R., et. al., 2000. A distributed mechanism for topology discovery in ad-hoc networks using mobile agents. Proceedings of 1st Annual Workshop on Mobile Ad-Hoc Networking Computing, MobiHOC Mobile Ad-Hoc Networking and Computing. Boston, USA.

Cisco Academy, 2003. CCNA 1: Networking Basics. Available from <http://cisco.netacad.net>. Last visited 21/07/2005.

Corradi, A., et. al., 2001. Security of mobile agents on the Internet. Internet Research: Electronic Networking Applications and Policy. Vol. 11, No. 1. pp.84-95.

Corson, M. S. and Ephremides, A., 1995. A distributed routing algorithm for mobile wireless networks. Wireless Networks. Vol. 1. No. 1. pp. 61-81.

Corson, S., 2000. Temporally-Ordered Routing Algorithm (TORA). IETF MANET Working Group - Internet Draft. draft-ietf-manet-tora-spec-03.txt.

Das, S. R., et. al., 2000. Simulation-based performance evaluation of routing protocols for mobile ad hoc networks. Mobile Network Applications. Vol. 5. No. 3. pp. 179-189.

Dasgupta, D. and Brian, H., 2001. Mobile Security Agents for Network Traffic Analysis. Proceedings DARPA Information Survivability Conference and Exposition II, DISCEX'01. IEEE Computer Society Los Alamitos, USA. pp. 332-40.

De, S., et. al., 2002. Trigger-Based Distributed QoS Routing in Mobile Ad hoc Networks. SIGMOBILE Mobile Computing and Communications Review. Vol. 6, No. 3. pp. 22-35.

Denko, M. K., 2003. The use of mobile agents for clustering in mobile ad hoc networks. SAICSIT '03: Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology. pp. 241-247.

Dube, R., et. al., 1997. Signal Stability-Based Adaptive Routing (SSA) for Ad-Hoc Mobile Networks. IEEE Personal Communications Magazine. Vol. 4. No. 1. pp. 36-45.

Dyer, T. D. and Boppana, R. V., 2001. A comparison of TCP performance over three routing protocols for mobile ad hoc networks. MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing. Long Beach, CA, USA. pp. 56-66.

Elaarag, H., 2002. Improving TCP performance over mobile networks. ACM Computing Surveys (CSUR). Vol. 34, No. 3. pp. 357-374.

Emmerich, W., 1997. An introduction to OMG/CORBA. Proceedings of the 19th International Conference on Software Engineering. Boston, USA. pp. 641-642.

Ephremides, A., et. al., 1987. A design concept for reliable mobile radio networks with frequency hopping signalling. Proceedings of IEEE. Vo. 75. No. 1. pp.56-73.

Fall, K. and Varadhan, K., 2005. The ns Manual. The VINT Project, collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC. Available from <http://www.isi.edu/nsnam/ns/>. Last visited 15-July 2005.

Familiar Project, 2004. The Familiar Project - Familiar v.0.7.2. Available from <http://familiar.handhelds.org>. Visited 08/2004.

Farmer, W. M., et. al., 1996. Security for mobile agents: authentication and state appraisal. Computer Security ESORICS 96, 4th European Symposium on Research in Computer Security Proceedings. Berlin, Germany. pp.118-30.

Fasbender, A., et. al., 1999. Any network, any terminal, anywhere. IEEE Personal Communications. Vol. 6. No. 2. pp. 22 – 30

FIPA, 1997. FIPA: Foundation for Intelligent Physical Agents. Specifications. Available from <http://www.fipa.org>. Last visited 21/07/2005.

Fischmeister, S., et. al., 2001. Evaluating the Security of Three Java-Based Mobile Agent Systems. G.P. Picco (Ed.). Vol. 2240. pp. 31–41.

FleetNet, 2003. Inter-vehicle communications. Available from <http://www.fleetnet.de>. Last visited 21/07/2005.

Franklin, S. and Graesser, A., 1996. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. Proceedings of the 3rd International Workshop on Agent Theories, Architectures, and Languages. Berlin, Germany. pp. 21-35

Franz, W., et. al., 2001. FleetNet - Internet on the Road. Proceedings of the 8th World Congress on Intelligent Transportation Systems. Sydney, Australia. pp. 46-55

Fritzinger, J. S. and Mueller, M., 1996. Java™ Security. White Paper. http://java.sun.com/docs/white/.

Frodigh, M., et. al., 2000. Wireless ad-hoc networking – The art of networking without a network. Ericsson Review. No. 4. http://www.ericsson.com/about/publications/revie w/2000_04/files/2000046.pdf.

Funfrocken, S., 1998. Transparent Migration of Java-Based Mobile Agents. Proceedings of the Second International Workshop on Mobile Agents (MA'98). Stuttgart, Germany. pp. 26-37.

Garcia-Luna-Aceves, J. J., 1989. A unified approach to loop-free routing using distance vectors or link states. SIGCOMM '89: Symposium proceedings on Communications architectures & protocols. Austin, Texas, USA. pp. 212-223.

Garcis-Luna-Aceves, J. J. and Spohn, M., 1999a. Source-Tree Adaptive Routing (STAR) Protocol. IETF MANET Working Group-Internet Draft. draft-ietf-manet-star-00.txt. Work in progress.

Garcis-Luna-Aceves, J. J. and Spohn, M., 1999b. Source-Tree Routing in Wireless Networks. Proceedings of the 7th International Conference on Network Protocols. Toronto, Canada. pp. 273-282.

Garside, R. and Mariani, J., 1998. Java: First Contact, An Introduction to the Java Language and Object-Oriented Programming. Published by Course Technology. ISBN: 185032316X.

Gavalas, D., et. al., 2001. Mobile software agents for decentralised network and systems management. Microprocessors-and-Microsystems. Vol. 25, No. 2. pp. 101-109.

Geier, J. 2003. Overview of Wireless IEEE 802.11 Standards. Available from <http://www.wi-fiplanet.com/tutorials/article.php/1439551>. Last visited 21/07/2005.

Gerla, M. and Tsai, J. T.-C., 1995. Multicluster, mobile, multimedia radio network. Wireless Networks. Vol. 1. No. 3. pp. 255-265.

Gerla, M., et. al., 2001. Fisheye State Routing Protocol (FSR) for Ad Hoc Networks. IETF MANET Working Group-Internet Draft. draft-ietf MANET-fsr-02. txt. Work in progress.

Ghanea-Hercock, R., 2001. Mobile Software Agents. Journal of the Institution of British Telecommunications Engineers. Vol. 2, Pt. 2. pp.54-8.

Gong, L., 1998. Java Security Architecure. Sun Microsystems, Inc. Available from <http://java.sun.com/products/jdk/1.2/docs/guide/security/spec/security-specTOC.fm.html>. Last visited 21/07/2005.

Gosling, J., et. al., 2000. The Java Language Specification, 2nd edition. The Java Series. Published by Addison-Wesley. USA. ISBN: 0201310082.

Gray, R. S., 1995. Agent Tcl: A transportable agent system. In Proceedings of the CIKM Workshop on Intelligent Information Agents, 4th International Conference on Information and Knowledge Management. Baltimore, USA.

Gray, R. S., 1998. D'Agents: Security in a multiple-language, mobile-agent system.

Gray, R. S., et. al., 2004. Outdoor experimental comparison of four ad hoc routing algorithms. MSWiM '04:

Proceedings of the 7th ACM international symposium on Modelling, analysis and simulation of wireless and mobile systems. Venice, Italy. pp. 220-229.

Graziano, A. M. and Raulin, M. L., 1993. Research Methods, A Process of Inquiry. 2nd edition. Published by Allyn & Bacon. ISBN: 0065010906

Green, S., et. al., 1997. Software agents: A review. IAG report. Dublin, Ireland. Broadcom `Ereann Research, Intelligent Agents Group.

Gupta, P. and Kumar, P. R., 1997. A system and traffic dependent adaptive routing algorithm for ad hoc networks. Proceedings of the 36th IEEE Conference on Decision and Control. San Diego. pp. 2375-2380.

Haas, Z. J., et. al., 2002a. The Zone Routing Protocol (ZRP) for Ad Hoc Networks. Internet-Draft. draft-ietf-manet-zone-zrp-04.txt. Work in progress.

Haas, Z. J., et. al., 2002b. Bordercasting Resolution Protocol (BRP). IETF Internet Draft, draft-ietf-manet-brp-02.txt.

Haas, Z. J., et. al., 2002c. Intrazone Routing Protocol (IARP). IETF Internet Draft. draft-ietf-manet-iarp-02.txt.

Hadjiefthymiades, S., et. al., 2002. Supporting the WWW in wireless communications through mobile agents. Mobile Networks & Applications. Vol. 7, No. 4. pp. 305-313

Hagimont, D. and Ismail, L., 1997. A protection scheme for mobile agents on Java. Proceedings of the third annual ACM/IEEE international conference on Mobile computing and networking. Budapest, Hungary. Pp. 215-222.

Harrison, C. G., et. al., 1995. Mobile Agents: Are they a good idea. Technical Report. IBM T.J. Watson Research Centre. New York, USA.

Hassanein, H. and Zhou, A., 2001. Routing with load balancing in wireless Ad-Hoc Networks. Proceedings of the 4th ACM International Workshop on Modelling, Analysis, and Simulation of Wireless and Mobile Systems. Rome, Italy. pp. 89-96.

Hayes-Roth, B., 1995. An Architecture for Adaptive Intelligent Systems. Artificial Intelligence: Special Issue on Agents and Interactivity. Vol. 72. pp. 329-365.

Hemphill, D., 2003. J2ME Gets Personal. JAVAPro. Available from <http://www.fawcette.com/javapro/2002_12/magazine/features/dhemphill/default_pf.asp>. Last visited 21/07/2005.

Hohl, F., 1998. Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts. Lecture Notes in Computer Science. Vol. 1419. pp. 92-113.

Hu, Y. C. and Johnson D. B., 2004. Securing Quality-of-Service Route Discovery in On-Demand Routing for Ad Hoc Networks. SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks. Washington DC, USA. pp. 106-117.

Hurley, S. and Whitaker, R. M., 2002. An agent based approach to site selection for wireless networks. Proceedings of the 2002 ACM symposium on Applied computing. Madrid, Spain. pp. 574-577.

Hwang, K. and Gangadharan, M., 2001. Micro-Firewalls for Dynamic Network Security with Distributed Intrusion Detection. Proceedings IEEE International Symposium on Network Computing and Applications. Los Alamitos, USA. pp.68-79.

IBM, 2004. WebSphere Everyplace Micro Environment. Available from <http://www-306.ibm.com/software/wireless/weme>. Visited 08/2004.

IBM, Inc., 1997. IBM Aglets software development kit. Technical Report home page. Available from <http://www.trl.ibm.co.jp/aglets>. Last visited 21/07/2005.

IEEE Standards Association, 2003. Overview of IEEE Wireless Standards. 802.11™ Working Group for Wireless Local Area Networks. Available from <http://standards.ieee.org/wireless/overview.html#802.11>. Last visited 21/07/2005.

IEEE Standards, 802.11, 1999. IEEE Computer Society LAN MAN Standards Committee. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std 802.11-1997. The Institute of Electrical and Electronics Engineers, New York, USA. http://grouper.ieee.org/groups/802/11/index.html.

IKV++, Inc., 2003. Grasshopper mobile agent system. Grasshopper Documentation. Available from <http://www.grasshopper.de>. Last visited 21/07/2005.

Ince, D. and Freeman, A., 1997. Programming the Internet with Java. Published by Addison-Wesley. ISBN: 0201175495.

Insignia, 2004. Jeode - Java Virtual Machine for resource-constrained devices. Available from <http://www.insignia.com/content/products/jvmProducts.shtml>. Visited: 08/2004.

Jacquet, P., et. al., 2000. Optimized Link State Routing (OLSR) Protocol, Internet Draft, draft-ietf-manet-olsr-01.txt. Work in progress.

Jacquet, P., et. al., 2001. Optimized Link State Routing Protocol for Ad Hoc Networks. IEEE INMIC Conference. Lahore, Pakistan.

Jaffe, J. M. and Moss, F. H., 1982. A Responsive Distributed Routing Algorithm for Computer Networks. IEEE Transactions on Communications. Vol. 30. No. 7. pp. 1758-1762.

Jansen, W., 2000. Countermeasures for Mobile Agent Security. Computer Communications: Special Issue on Advanced Security Techniques for Network Protection. Elsevier Science BV.

Jennings, N. and Woolridge, M., 1998. Agent Technology: Foundations, Applications and Markets. Published by Springer.

Jiang, M., et. al., 2001. Cluster Based Routing Protocol (CBRP). Internet-Draft. draft-ietf-manet-cbrp-spec-01. Work in progress.

Jiang, M.-H., et. al., 2002. An efficient multiple-path routing protocol for ad hoc networks. Computer Communications. Vol. 25. No. 5. pp. 478-484.

Joa-Ng, M. and Lu, I.-T., 1999. A Peer-to-Peer zone-based two-level link state routing for mobile Ad Hoc

Networks. IEEE Journal on Selected Areas in Communications, Special Issue on Ad-Hoc Networks. Vol. 17. No. 8. pp.1415-1425.

Johnson D. B., et. al., 2004. The Dynamic Source Routing protocol for Mobile Ad Hoc Networks (DSR). Internet Engineering Task Force (IETF). Mobile Ad Hoc Networking working group (MANET) Official Internet Draft. draft-ietf-manet-dsr-10.txt. Work in progress.

Johnson, D. B. and Maltz, D. A., 1996. Dynamic source routing in ad-hoc wireless networks. In Mobile Computing. Chapter 5. Kluwer Academic Publishers. pp. 153-181.

Johnson, D. B., et. al., 2004. The Dynamic Source Routing protocol for Mobile Ad Hoc Networks (DSR). Internet Engineering Task Force (IETF). Mobile Ad Hoc Networking working group (MANET) Official Internet Draft. draft-ietf-manet-dsr-10.txt. Work in progress.

Jonsson, U. and Alriksson, F., 2000. MIPMANET: mobile IP for mobile ad hoc networks. Proceedings of the first ACM international symposium on Mobile and ad hoc networking & computing. Boston, USA. pp. 75-85.

Jubin, J. and Tornow, J. D., 1987. The DARPA Packet Radio Network Protocols. Proceedings of the IEEE. Vol. 75. No. 1. pp. 21-32.

Jul, E., et. al., 1988. Fine-grained mobility in the Emerald System. ACM Transactions on Computer Systems (TOCS). Vol. 6, No. 1. pp. 109-133.

Kaplan, E. D., 1996. Understanding GPS: principles and applications. Artech House. Boston, MA.

Kapoor, R., et. al., 2001. Multimedia support over Bluetooth Piconets. Proceedings of the first workshop on Wireless mobile Internet. Rome, Italy. pp. 50-55.

Karjoth, G. and Posegga, J., 2000. Mobile agents and telcos' nightmares. Annales des Telecommunications. vol. 55, no.7-8. pp. 388-400.

Karnik, N. M. and Tripathi, A. R., 2001. Security in the Ajanta mobile agent system. Software Practice and Experience. Vol. 31, No. 4. pp.301-29.

Karp, B. and Kung, H. T., 1998. Dynamic neighbor discovery and loopfree, multi-hop routing for wireless, mobile networks. Hardvard University, Draft. Available at <http://www.eecs.harvard.edu/~karp/aprl.ps>. Last visited, 16-07-2005.

Keeble, S., 1995. Experimental Research 1, An introduction to Experimental Research. Published by the Open Learning Foundation. ISBN: 0443052700

Ko, Y.-B. and Vaidya, N. H., 1998. Location-aided routing (LAR) in mobile ad hoc networks. MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking. Dallas, Texas, USA. pp. 66-75.

Kotz, D., et. al., 1997. Agent TCL: Targeting the needs of Mobile Computers. IEEE Internet Computing. Vol. 1, No. 4. pp. 58-67.

Kotzanikolaou, P., et. al., 2000. Secure Transactions with Mobile Agents in Hostile Environments. Fifth Austalasian Conference on Information Security and Privacy. Brisbane, Australia. pp. 289-297.

Krugel, C., et. al., 2002. SPARTA-a mobile agent based intrusion detection system. Advances in Network and Distributed Systems Security, First Annual Working Conference on Network Security. Norwell, USA. pp. 361–370.

Lang, G. and Heiss, G. D., 1984. A Practical Guide to Research Methods. Published by University Press of America. ISBN: 081913726X

Lange, D. B. and Oshima, M. 1999. Seven good reasons for mobile agents. Communications of the ACM. Vol. 42, No. 3. pp 88-89.

Lee, S. J., et. al., 2002. On-demand multicast routing protocol in multihop wireless mobile networks. Mobile Network Applications. Vol. 7. No. 6. pp. 441-453.

Lee, S.-J., et. al., 2001. Wireless ad hoc multicast routing with mobility prediction. Mobile Network Applications. Vol. 6. No. 4. pp. 351-360.

Lee, T. O., et. al., 2001. An agent-based micropayment system for E-commerce. E-commerce agents, Marketplace solutions, security issues, and supply and demand. Berlin, Germany. pp.247-63.

Lindholm, T. and Yellin, F., 1999. The Java Virtual Machine Specification, 2nd edition. The Java Series. Published by Addison-Wesley. USA. ISBN: 020163452X

Liu, J., et. al., 2002. A Unified Framework for Resource Discovery and QoS-Aware Provider Selection in Ad Hoc Networks. SIGMOBILE Mobile Computing and Communications Review. Vol. 6, No. 1. pp. 13-21.

Maes, P., 1995. Artificial Life Meets Entertainment: Life like Autonomous Agents. Communications of the ACM. Vol. 38, No. 11. pp. 108-114.

Marques, P., et. al., 2001. Providing applications with mobile agent technology. IEEE Open Architectures and Network Programming Proceedings. Piscataway, USA. pp.129-36.

Marrow, P. and Ghanea-Hercock, R., 2000. Mobile software agents-insect-inspired computing. BT Technology Journal. Vol. 18, No. 4. pp. 129-139.

Marwaha, S., et. al., 2002. Mobile Agents based Routing Protocol for Mobile Ad Hoc Networks. Symposium of Ad-hoc networks. Proceedings of the IEEE Globecom.

McDonald, A. B. and Znati, T., 2000. Predicting node proximity in ad-hoc networks: a least overhead adaptive model for selecting stable routes. MobiHoc '00: Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing. Boston, Massachusetts, USA. pp. 29-33.

McQuillan, J. M., et. al., 1980. The New Routing Algorithm for the ARPANET. IEEE Transactions on Communications. Vol. 28. No. 5. pp. 711-719.

Microsoft, Corporation, 2004. Windows Mobile-based Pocket.PCs. Available from <http://www.microsoft.com/windowsmobile/pocketpc/ppc/default.mspx>. Visited: 08/2004.

Migas, N. and Buchanan, W. J., 2005. Ad-hoc Routing Metrics and Applied Weighting for QoS support. Ad-hoc networks. To be published.

Migas, N., et. al., 2003a. Mobile Agents for Routing, Topology Discovery, and Automatic Network Reconfiguration in Ad-Hoc Networks. 10th IEEE International Conference and Workshop on the Engineering of Computer Based Systems. Huntsville, USA, pp. 200-206.

Migas, N., et. al., 2003b. MARIAN: A Framework using Mobile Agents for Routing in Ad-hoc Networks. IADIS International Conference on WWW/Internet, Algarve, Portugal. pp. 1129-1134.

Migas, N., et. al., 2004a. Migration of Mobile Agents in Ad-hoc, Wireless Networks. 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04). Brno, Czech Republic. pp. 530-535.

Migas, N., et. al., 2004b. Benchmarking Bandwidth and Resource Consumptions of Java-based Proxy PDAs in Ad-hoc Networks. Expert Update. Vol. 7, No. 3. pp. 9-17.

Migas, N., et. al., 2005. Metric Evaluation of Embedded Java-Based Proxies on Handheld Devices in Cluster-Based Ad Hoc Routing. 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05). Washington D.C., USA. pp. 147-154.

Mills, D.L., 1991. Internet time synchronization: the Network Time Protocol. IEEE Transactions on Communications Vol. 39. No. 10. pp. 1482-1493.

Minar, N., et. al., 1999. Cooperating Mobile Agents for Dynamic Network Routing. Published by Springer-Verlag. ISBN: 3540655786

Mitsubishi, Electric, 1997. Concordia. Available from <http://www.merl.com/projects/concordia2>. Last visited 21/07/2005.

Mohapatra, P. K., 2000. Public key cryptography. Crossroads. Vol. 7, No. 1. pp. 14-22.

Murthy, S. and Garcia-Luna-Aceves, J. J., 1995. A routing protocol for packet radio networks. Proceedings of the 1st annual international conference on Mobile computing and networking. Berkeley, California, USA. pp. 86-95.

Necula, G. and Lee, P., 1996. Safe Extensions without Run-Time Checking. Proceeding of the 2nd Symposium on Operating System Design and Implementation (OSDI '96). Seattle, USA. pp. 229-243.

Nikaein, N., et. al., 2000. DDR: distributed dynamic routing algorithm for mobile ad hoc networks. International Symposium on Mobile Ad Hoc Networking & Computing. Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing. Boston, Massachusetts, USA. pp. 19-27.

Nikaein, N., et. al., 2001. HARP: Hybrid Ad hoc Routing Protocol. Proceedings of IST: International Symposium on Telecommunications. Teheran, Iran.

Nishiyama, H. and Mizoguchi, F., 2001. Design of security system based on immune system. Proceedings of the 10th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. Los Alamitos, USA. pp. 138-43.

NSICom, 2004. CrEme™ - The Java™ Enabler for Windows® CE. Available from <http://www.nsicom.com/Default.aspx?tabid=138>. Visited: 08/2004.

ObjectSpace, Inc, 1997. ObjectSpace Voyager Core Technology. Technical report. http://www.cis.upenn.edu/~bcpierce/courses/629/papers/unfiled/AgentPlatformsW97.PDF.

Ogier, R., et. al., 2003. Topology Dissemination Based on Reverse-Path Forwarding (TBRPF).Internet draft. draft-ietf-manet-tbrpf-10.txt. Work in progress.

OMG, MASIF, 1997. Mobile Agent Facility Specification. Object Management Group (OMG). Available from <http://www.omg.org/docs/orbos/97-10-05.pdf>. Last visited 21/07/2005.

Papadimitratos, P., et. al., 2002. Path Set Selection in Mobile Ad Hoc Net-works. MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing. Lausanne, Switzerland. pp. 1-11.

Parekh, A. K., 1994. Selecting routers in ad-hoc wireless networks. Proceedings of the SBT/IEEE International Telecommunications Symposium.

Pei, G., et. al., 1999. A Wireless Hierarchical Routing Protocol with Group Mobility. Proceedings of IEEE Wireless Communications and Networking Conference (WCNC). New Orleans, LA, USA. pp. 1538-1542.

Perkins, C. and Bhagwat, P., 1994. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. Proceedings of the ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications. London, UK. pp. 234-244.

Perkins, C. E. and Royer, E. M., 1999. Ad-hoc On-Demand Distance Vector Routing. Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications. New Orleans, USA. pp. 90-100.

Perkins, C. E., 2001. Ad-hoc networking: an introduction. Ad-hoc networking. Published by Addison-Wesley. ISBN: 0201309769

Perkins, C. E., et. al., 2000. Quality of Service for Ad Hoc On-Demand Distance Vector Routing. IETF Internet draft. draft-ietf-manet-aodvqos-00.txt. Work in progress.

Perkins, C. E., et. al., 2003. Ad hoc On-Demand Distance Vector (AODV) Routing. Internet Draft. draft-ietf-manet-aodv-13.txt. Work in progress.

Peysakhov, M., et. al., 2004. Network Awareness for Mobile Agents on Ad Hoc Networks. Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04). Vol. 1. No. 1. pp. 368-376.

Pfleeger, C. P., 1997. Security in Computing, 2nd edition. Published by Prentice Hall. ISBN: 0-13-185794-0.

Pham, V. A. and Karmouch, A., 1998. Mobile software agents: an overview. IEEE Communications. Vol. 36, No. 7. pp. 26-37.

Phan, T., et al., 2002. Challenge: integrating mobile wireless devices into the computational grid. Proceedings of the eighth annual international conference on Mobile computing and networking. Atlanta, USA, pp 271-278.

Ping, Y., et. al., 2004. Securing ad hoc networks through mobile agent. InfoSecu '04: Proceedings of the 3rd international conference on Information security. Shanghai, China. pp. 125-129.

Powell, M. L. and Miller, B. P., 1983. Process migration in DEMOS/MP. Proceedings of the ninth ACM symposium on Operating systems principles. Bretton Woods, USA. pp. 110-119.

Puliafito, A., et. al., 2000. MAP: Design and implementation of a mobile agents' platform. Journal of Systems Architecture. Vol. 46, No. 2. pp.145-62.

Qi, H. and Wang, F., 2001. Optimal itinerary analysis for mobile agents in ad hoc wireless sensor networks. Proceedings International Conference on Wireless Communications. Calgary, Canada. pp. 147-153.

Radhakrishnan, S., et. al., 1999. DST - a routing protocol for ad hoc networks using distributed spanning trees. IEEE Wireless Communications and Networking Conference. New Orleans, USA. pp. 100-104.

Rajaraman, R., 2002. Topology control and routing in ad hoc networks: a survey. ACM SIGACT News. Vol. 33, No. 2. pp. 60-73.

Raju, J. and Garcia-Luna-Aceves, J. J., 1999. A New Approach to On-demand Loop-Free Multipath Routing. Proceedings of the 8th Annual IEEE International Conference on Computer Communications and Networks (ICCCN). Boston, Massachusetts. pp. 522-527.

Ramanathan, R. and Steenstrup, M., 1998. Hierarchically-organized, multihop mobile wireless networks for quality-of-service support. Mobile Network Applications. Vol. 3. No. 1. pp. 101-119.

Ramarathinam, V. and Labrador, M. A., 2002. Performance Analysis of TCP over static Ad-Hoc Wireless Networks. In Proceedings of the ISCA 15th International Conference on Parallel and Distributed Computing Systems (PDCS). Louisville, USA. pp. 410-415.

Reilly, D. and Reilly, M., 2002. Java Network Programming and Distributed Computing. Published by Addison-Wesley. ISBN: 0201710374.

Riordan, J. and Schneider, B., 1998. Environmental Key Generation Towards Clueless Agents. Lecture Notes in Computer Science. Vol. 1419. pp. 15-24.

Roberts, L. G., 1967. Multiple Computer Networks and Intercomputer Communication. Proceedings of the ACM Symposium on Operating System Principles. 1967, pp. 3.1-3.6.

Roth, V., 1998. Secure Recording of Itineraries through Cooperating Agents. In Proceedings of the 4th ECOOP Workshop on Mobile Object Systems: Secure Internet Mobile Computations. Brussels, Belgium. pp. 147-154.

RoyChoudhury, R., et. al., 2000. A distributed mechanism for topology discovery in ad hoc wireless networks using mobile agents. MobiHoc '00: Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing. Boston, Massachusetts, USA. pp. 145-146.

Royer, E. M. and Toh C. K., 1999. A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks. IEEE Personal Communications Magazine. Vol. 6, No. 2, pp. 46-55.

Samaras, G. and Panayiotou, C., 2002. Personalized Portals for the Wireless User based on Mobile Agents. International Conference on Mobile Computing and Networking, Proceedings of the second International

Workshop on Mobile Commerce. Atlanta, USA. pp. 70-74.

Sander, T. and Tchudin, C. F., 1998. Mobile Agents against Malicious Hosts. Lecture Notes in Computer Science. Pp. 44-60.

Schneider, F. B., 1997. Towards Fault-Tolerant and Secure Agentry. Proceedings of 11th International Workshop. Saarbr cken, Germany. pp. 1-14.

Schwartz, M. and Stern, T. E, 1980. Routing Techniques Used in Computer Communication Networks. IEEE Transactions on Communications. Vol. 28. No. 4. pp. 539-552.

Sedgewick, R., 1983. Weighted Graphs. Chapter 31. Addision-Wesley.

Shankar, A. U., et. al., 1992a. Transient and steady-state performance of routing protocols: Distance vector versus link-state. Journal of Internetworking: Research and Experience. Vol. 6. pp. 59-87.

Shankar, A. U., et. al., 1992b. Performance comparison of routing protocols using MaRS: distance-vector versus link-state. SIGMETRICS Performance Evaluation Review. Vol. 20. No. 1. pp. 181-192.

Silva, A. R., et. al., 2001. Towards a reference model for surveying mobile agent systems. Autonomous Agents and Multi Agent Systems. Vol. 4, No. 3. pp.187-231.

Smith, D. C., et. al., 1994. KidSim: Programming Agents Without a Programming Language. In Communications of the ACM. Vol. 37, No. 7. pp. 54-67.

Spafford, E. H. and Zamboni, D., 2000. Intrusion detection using autonomous agents. Computer Networks. Vol. 34, No. 4. pp.547-70.

Stajano, F. and Anderson, R., 2000. The resurrecting duckling: Security issues for ad-hoc wireless networks. In the Proceedings of the 7th International Workshop on Security Protocols. Cambridge, UK. pp. 172-182.

Stamos, J. W. and Gifford, D. K., 1990. Remote Evaluation. ACM Transactions on Programming Languages and Systems (TOPLAS). Vol. 12, No. 4. pp. 537-564.

Su, W. and Gerla, M., 1999. IPv6 flow handoff in ad hoc wireless networks using mobility prediction. GLOBECOM: IEEE Global Telecommunications Conference. Rio de Janeiro, Brazil. pp. 271-275.

Su, W., et. al., 2001. Mobility prediction and routing in ad hoc wireless networks. International Journal of Networks Management. Vol. 11. No. 1. pp. 3-30.

Sun, Microsystems, 2003a. Java™ 2 Platform, Micro Edition. Available from <http://java.sun.com/j2me/j2me-ds.pdf>. Last visited 21/07/2005.

Sun, Microsystems, 2003b. The Java™ 2 Platform, Standard Edition. Available from <http://java.sun.com/j2se>. Last visited 21/07/2005.

Sun, Microsystems, 2003c. Java™ 2 Micro Edition (J2ME™). Sun Community Source Licensing (SCSL). Available from <http://wwws.sun.com/software/communitysource/j2me/index.html>. Last visited 21/07/2005.

Tanenbaum, A. S., 1996. Computer Networks 3rd Edition. Published by Prentice Hall, Inc. ISBN: 013394248-1.

Tay, B. H. and Ananda, A. L., 1990. A survey of remote procedure calls. ACM SIGOPS Operating System Review. Vol. 24, No. 3. pp. 68-79.

Tcl, Developer, 2003. Tcl language. Available from <http://www.tcl.tk>. Last visited 21/07/2005.

Tianfield, H., 2001. Enterprise Federation and Its Multi-agent Modelization. E-Commerce Agents, Marketplace Solutions, Security Issues, and Supply and Demand. pp. 295--322.

Tianfield, H., 2003. Multi-Agent Autonomic Architecture and Its Application in E- Medicine. IEEE/WIC International Conference on Intelligent Agent Technology (IAT'03). pp. 601.

Toh, C., 1996. A novel distributed routing protocol to support ad-hoc mobile computing. IEEE 15th Annual International Phoenix Conference. pp. 480 - 486.

Tripathi, A. R., 1998. Design Issues in Mobile Agent Systems. PhD Thesis. Computer Science & Engineering, University of Minnesota, USA. pp. 1-136.

Tripathi, A. R., et. al., 2000. Experiences and Future Challenges in Mobile Agent Programming. Microprocessors and Microsystems. Vol. 25, No. 2. pp. 121-129.

Urquhart, N., et. al., 2003. Routing Using Evolutionary Agents and Proactive Transactions. EvoWorkshops. pp. 696-705.

Vigna, G., 1997. Protecting Mobile Agents through Tracing. Proceedings of the Third Workshop on Mobile Object Systems. Jyvalskyla, Finland.

Vigna, G., 1999. Mobile Agents and Security. Published by Springer-Verlag. ISBN: 3540647929.

Vinaja, R., 2001. Mobile agents, mobile computing and mobile users in global e-commerce. Managing Information Technology in a Global Environment, Information Resources Management Association International Conference. PA, USA. pp.173-6.

Wahbe, R., 1994. Efficient software-based fault isolation. Proceedings of the fourteenth ACM symposium on Operating system principles. Asheville, USA. pp. 203-216.

Wang, H., 2000. On mobile agent-based scheme for e-commerce on the Internet. 16th World Computer Congress 2000, Proceedings of Conference on Intelligent Information Processing. Beijing, China. pp. 386-9.

Wang, X., et. al., 2001. A Multicast Routing Algorithm Based on Mobile Multicast Agents in Ad-Hoc Networks. Special Issue on Internet Technology, IEICE TRANS. COMMUN. Vol. E84-B, No. 8. pp. 2087-2094.

Wang, Y. and Pang, X., 2003. Security and robustness enhanced route structures for mobile agents. Mobile Network Applications. Vol. 8. No. 4. pp. 413-423.

White, J., 1996. Mobile Agents White Paper. Technical report. General Magic.

White, J., 1997. Telescript technology: An introduction to the language. General Magic Inc. White paper.

Wolthusen, S. D., 2002. Access and use control using externally controlled reference monitors. ACM SIGOPS Operating Systems Review. Vol. 36, No. 1. pp. 58-69.

Wong, D., et. al., 1997. Concordia: An Infrastructure for Collaboration Mobile Agents. In Proceedings of the

first International Workshop on Mobile Agents (MA97). Berlin, Germany. pp. 86-97.

Woo, S.-C. and Singh, S., 2001. Scalable Routing Protocol for Ad Hoc Networks. Wireless Networks. Vol. 7. No. 5. pp. 513-529.

Wooldridge, M. J. and Jennings, N. R., 1995. Agent Theories, Architectures, and Languages: a Survey. Intelligent Agents; Workshop on Agent Theories, Architectures, and Languages. Berlin, Germany. pp. 1-39.

Wu, H. K. and Chuang, P. H., 2001. Dynamic QoS Allocation for Multimedia Ad Hoc Wireless Networks. Mobile Networks and Applications. Vol. 6, No. 4. pp. 377-384.

Yi, Y., et. al., 2002. The selective intermediate nodes scheme for Ad Hoc on-demand routing protocols. ICC 2002 - IEEE International Conference on Communications. Vol. 25. No. 1. pp. 3191 - 3196.

Zhang, M., et. al., 2001. Towards a Secure Agent Platform Based on FIPA. MATA 2001. Montreal, Canada. pp. 277-289.

Zhang, R., et. al., 2001. Multi-agent Based Intrusion Detection Architecture. Proceedings of 2001 International Conference on Computer Networks and Mobile Computing, IEEE Computer Society. Los Alamitos, USA. pp. 494-501.

Zorzi, M., 1998. Mobile and Wireless Telecommunication Networks. Centre for Wireless Communications. University of California San Diego, UCSD. February 18, 2000.

A Appendix - Additional information

A.1 Java Micro Edition (J2ME)

The Java 2 Micro Edition (J2ME) is the developers' platform for consumer and embedded devices, such as mobile phones, PDAs, TV set-top boxes, and in-vehicle telematics systems (Sun, Microsystems, 2003a, Sun, Microsystems, 2003c). It is highly suitable for the high-end PDA market. It contains full support for graphical user interfaces (GUI), including support for applets, as well as a complete toolkit. The initial implementation of J2ME specification was PersonalJava, which was created a few years ago to support Java on Pocket PCs (Windows OP) and other handheld devices. A new implementation is Personal Profile, which allows moving code between more limited devices (such as PalmOS devices) and more powerful devices (such as Pocket PC) with little or no modification. Since J2ME is designed for limited devices, it contains a subset of classes when compared to Java 2 Standard Edition (J2SE) (Sun Microsystems, 2003b), and also includes classes specific to J2ME that are not present in the J2SE libraries. Figure A.1 illustrates the relationship of J2ME (Sun Microsystems, 2003a, Sun Microsystems, 2003c) with J2SE.

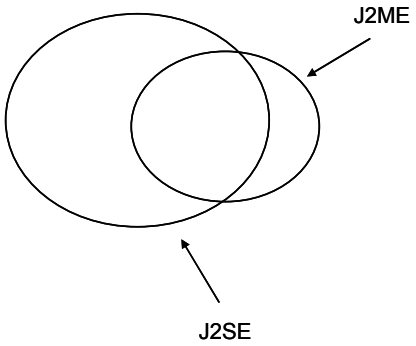

Figure A.1: Relationship between J2ME and J2SE

J2ME has powerful capabilities since it is based on JDK 1.3.1 (Sun, Microsystems, 2003b), although it does not fully support JDK 1.3.1 because of the requirement of a small footprint, since it aims at limited devices such as PDAs. Some of the most important features supported by J2ME are (Hemphill, D., 2003):

- **Collections Framework**. The full set of collections framework is supported in J2ME such

as *Vector*, *Hashtable*, *Arrays*, *Linked lists*, and so on.

- **Interface Capabilities**. J2ME provides full *AWT* support. However, *Swing* is not supported.
- **Networking**. J2ME provides fundamental networking support such as *HTTP*, *File*, *TCP/IP* Socket, and Datagram connections, while other connection types are optional (such as *HTTPS*, *SMTP*, or *FTP*).
- **Data Storage**. A variety of support is provided for data storage, such as *File*, *RandomAccesFile*, *FileInputStream*, *FileOutputStream*, and so on. JDBC support (the java.sql packages) is not included in J2ME, however, implementers may choose to provide support as part of their offering.
- **Alpha Blending**. J2ME provides support for alpha blending, which offers the ability to mix source and destination pixels together to provide transparency and blended image effects.
- **Remote Method Invocation** (RMI). J2ME supports a subset of RMI, namely the Remote and Registry interfaces. The role of Remote and Registry interfaces is somewhat different than RMI. It exists to provide a way to manage intra-JVM class communications, or more specifically, it provides an interface to manage interactions between classes that are loaded into the same JVM, but by different class loaders.
- **Java Native Interface** (JNI). This is optional, and thus it is up to the user to decide whether there is adequate storage space in her/his devices for this package. On the other hand, portability issues may arise due to the fact that JNI support is not present by default.

A.2 Suitability of Java for the Mobile Agent Paradigm

As a language, Java is ideally suited to the development of software agents (Reilly, D. and Reilly, M., 2002). Also, Java is the predominant language for mobile agent systems, both for implementing mobile agent execution environments and for writing mobile agent applications (Binder, W. and Roth, V., 2002). Java has several features that are not found in any other language, which may directly facilitate efficient implementation of mobile agents (Funfrocken, S., 1998). Therefore, mobile agent and mobile agent system design and development may directly benefit from key features of Java:

- **Portability of mobile code**. Code portability is achieved by the extra layer of code interpretation (Hagimont, D. and Ismail, L., 1997). Java Runtime Environments (JREs) are available for most hardware platforms and operating systems, such as Linux, Windows,

and so on. Thus, mobile agent systems written in Java can in theory run flawlessly on any heterogeneous machine that has a Java runtime environment installed.

- **Java object serialisation**. Java's serialisation support allows the conversion of an agent and its state into a form suitable for network transmission, and the reconstruction of the agent on the other end. This process is almost transparent to the programmer.
- **Dynamic class loading**. This technique allows the dynamic loading of classes included in an application either locally or through the network by means of a hierarchy of class loaders.
- **Multi-threading**. A mobile agent system may execute multiple agents and service components concurrently in a time-sharing fashion. Multi-threading can effectively and efficiently cope with this demand.
- **Java's security model**. It generally imposes security restrictions for code which may be considered untrusted. Code loaded by Java's class loaders is subject to security restrictions, and thus protects the agents and the host from unauthorised access. Also, classes that are downloaded from the Internet are placed within the sandbox and executed with certain limitations.
- **Separate class naming space**. Each class loader constitutes a separate name space that can be used to isolate classes of the agent system and of different agents from each other, and thus provides security against agent-to-agent attacks.
- **Type-Safe language.** It does not allow direct access to the address space of the program and it is strongly typed, implementing rigorous compile and runtime checks. Therefore, when a block of code is executed, Java makes sure that types are not misunderstood and data is not mistaken.

Other features include: bytecode verification, strong typing, automatic memory management, dynamic bound checks, and exception handlers (Fritzinger, J. S. and Mueller, M., 1996).

A.3 Wireless Standards: IEEE 802.11

The IEEE 802.11 specifications are wireless standards that detail an "over-the-air" interface between a wireless client and a base station or access point, as well as among wireless clients (IEEE Standards Association, 2003). The specifications address both the Physical and Media Access Control (MAC) layers, which are adapted to resolve compatibility issues between manufacturers of Wireless Local Area Network (WLAN) equipment. The 802.11 wireless standards include (IEEE Standards, 802.11, 1999, Geier, J. 2003):

- **IEEE 802.11a**. This is a Physical Layer (PHY) standard that specifies operating in the 5GHz UNII (Unlicensed National Information Infrastructure) band using Orthogonal Frequency Division Multiplexing (OFDM). This standard supports data rates ranging from 6 to 54Mbps. Because of operation in the 5 (GHz) bands, 802.11a offers much less potential for radio frequency (RF) interference than other PHYs (e.g., 802.11b and 802.11g) that utilize 2.4 (GHz) frequencies.
- **IEEE 802.11b**. Also known as Wi-Fi is the most commonly used standard in WLAN communications. It operates in the 2.4 (GHz) band and can transmit up to 11 (Mbps). The maximum bandwidth can be achieved if wireless devices implementing this standard are in a distance within 100 feet. Bandwidth gradually decreases as the source and the destination are moving further apart. Most wireless devices such as mobile phones, PDAs, laptops, and so on, utilise this standard to create ad-hoc networks.
- **IEEE 802.11c**. It defines wireless bridge operations. This standard is mainly utilised by developers of access points.
- **IEEE 802.11d**. It defines standards for companies developing wireless products in different countries. This is especially important for operation in the 5 (GHz) bands because the use of these frequencies differ widely from one country to another.
- **IEEE 802.11e**. It defines enhancements to the 802.11 MAC for Quality of Service (QoS) support, in order to optimise the transmission of multimedia traffic such as voice and video. Upgrading an existing 802.11 access point to comply with 802.11e standard, may be achieved through relatively simple firmware changes.
- **IEEE 802.11f**. It specifies an inter access point protocol that provides the necessary information that access points need to exchange, in order to provide support for 802.11 distribution system functions (e.g., roaming).
- **IEEE 802.11g**. It aims on the development of a higher speed extension to the 802.11b PHY, while operating in the same 2.4 (GHz) band. Available bandwidth may be up to 54 (Mbps). 802.11g is a strong candidate for the wireless format of the near future. Thus, allowing users to create ad-hoc networks at higher speeds.
- **IEEE 802.11h**. It provides dynamic channel selection (DCS) and transmit power control (TPC) for devices operating in the 5 (GHz) band (802.11a), aiming to avoid interference with satellite communications and any other communications operating in the 5 (GHz) band.
- **IEEE 802.11i**. It defines enhancements to the MAC Layer in order to provide Wired Equivalent Privacy (WEP). This standard aims to provide stronger encryption techniques such as Advanced Encryption Standards (AES), and thus enhance security in wireless

communications.

A.4 MARIAN Terminology

This section summarises the terminology that is used in Chapter 3.

- **Node-ID**. A unique, real number between the range of 0 and 100, which results from the device's mobility patterns, multiplexed with the device's performance characteristics. Its uniqueness may be guaranteed in a distributed environment by translating the node's address into an integer number and appending the number into the decimal part of the original cluster-head metric.
- **Routing-ID**. An array of preliminary metrics, deduced by the output of standard performance tests and monitoring status. The performance tests are executed on each device, in advance, including tests, such as complex calculation, buffering capability, network throughput, error packet percentage, and so on. The monitoring status includes varying parameters, such as the CPU utilisation, memory usage, and battery level, if applicable.
- **NodeAddress**. It is a *string* which uniquely identifies a mobile node within an ad-hoc network. The node address is typically the IP address of the node's wireless interface and is used for the purpose of routing and interoperability with fixed networks. Similarly to CBRP, the node address is represented by the IP address in this specification. In case of a node having two or more IP addresses, the default address is selected for its node address.
- **AgencyAddress**. Similarly to node address, the agency address is essentially a *string*, consisting of a node's IP address along with the protocol and port number used for agent communications, and the given name of the mobile agent system. The agency address can be customized by each node, however, its uniqueness among a distributed environment is guaranteed, as long as there are no IP address duplicates. The agency address is required for mobile agent migration and agent communication services.
- **RegionAddress**. It is very similar to the agency address, with the only difference being that the region registry implements a registration service for agencies, agents, and services, rather than an execution environment for agents. In particular, only cluster-heads are required to implement a region registry, with purpose of maintaining information about all other agencies, stationary and mobile agents, and services, which may exist within their own clusters. Therefore, a node can effectively and efficiently retrieve information about other nodes (within its cluster) that are two hops away by quoting the cluster-head's region registry.

- **Cluster**. A cluster, in respect to ad-hoc networks, is defined as a collection of mobile nodes sharing some common characteristics, such as nearby geographical position, in which devices are at a maximum of *N* hops away, where *N* is defined by the cluster-horizon. MARIAN defines the horizon being at most two hops away, while cluster-horizon values greater than two hops will be investigated in further work. The cluster-head, elected by the clustering process, is always directly linked to every node in its cluster, whereas the remaining nodes may be up to two hops away. Thus, the size of the cluster is defined by the wireless transmission range of the cluster-head. A node is elected as a cluster-head, if and only if, its cluster-head metric is the strongest amongst its neighbouring nodes or the node has no neighbours.
- **Host cluster**. A mobile node can belong to one or more clusters, providing that the node is in direct communication range to the cluster's (s') cluster-head(s). Therefore, a node *X* regards cluster *Y* as its host cluster, if and only if, it has a communication link to the cluster-head of cluster *Y*. Along this principle, a node can have two or more host clusters.
- **Cluster-head**. Elections take place in the beginning of the clustering formation process. The fitter device is elected as a cluster-head and a cluster is formed. The cluster-head is always in direct communication range to every device in the same cluster. In addition, two or more cluster-heads must not be directly linked. If this happens, the fitter device is chosen to remain as a cluster-head and the remaining devices change their roles accordingly. Criteria that make one device fitter than another include: low mobility, high processing power, high memory and battery capacity, and low utilisation factors, and so on. A cluster-head's main responsibilities include: routing within its own cluster, maintaining a region registry, and maintaining routing tables for inter-cluster routing.
- **Cluster address**. The cluster address of a cluster *Y* with cluster-head *X*, is *X*'s node address, or, a combination of *X*'s region registry and agency addresses.
- **Undecided**. A node that has not decided its role yet, and has thus not completed its clustering formation process as yet.
- **Member**. A node *N* which belongs to one, and only one, cluster *Y*. Member nodes have no routing responsibilities.
- **Gateway**. A node which belongs to two or more clusters and is thus responsible for inter-cluster routing.
- **Distributed Gateway**. A pair of nodes which are directly linked, however, each node belongs to different clusters. Thus these clusters, which these nodes belong to, are linked with this distributed gateway pair. This pair can be used for inter-cluster routing, in a similar way to gateway nodes.

A.5 Benchmarking the routing capabilities of a proxy-based ad-hoc routing device

As previously mentioned, the proxy throughput agent requires the assistance of other agents, such as the TCP transmitter agent, the TCP receiver agent, and the throughput calculation agent (*see* Figure A.2). In addition to these, a number of other agents are linked to this test, such as the battery monitoring agent, the temperature variation monitoring agent, the CPU utilisation monitoring agent, and the heap memory usage monitoring agent. This is due to the fact that CPU, memory, battery, and temperature measurements may provide a better insight in the performance of a routing device than throughput alone.

This test requires three networked nodes, a transmitting, a proxy, and a receiving, where raw data are passed from the transmitting to the proxy and from the proxy to the receiving device. The throughput calculation is derived by measuring the time taken for certain amounts of data to arrive at the destination through the intermediate device. This test can also be used to measure the speed of the local network protocol stack by running the transmitter, proxy, and receiver agents on the same machine.

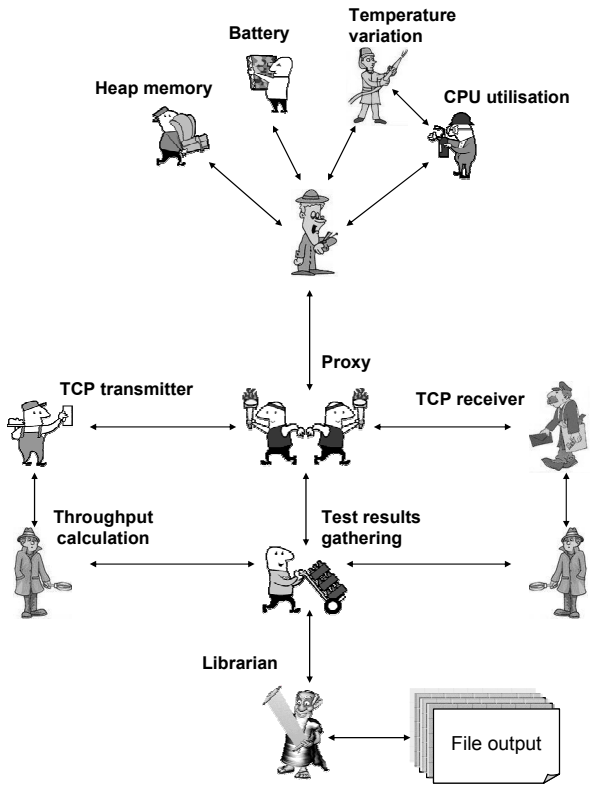

Figure A.2: Proxy throughput and resource consumptions multi-agent model

The TCP transmitter agent resides in the transmitter device, while agent the TCP receiver agent resides in the receiver device, while agent the throughput calculation agent resides in both. The remaining agents reside in the PDA, apart from the data gathering agent which is mobile, and its purpose is to collect the throughput results, which are generated by the throughput calculation agents. In particular, the TCP transmitter agent is responsible for transmitting heavy network traffic to the proxy agent, which then forwards it to the TCP receiver agent. The throughput calculation agent is responsible for calculating the throughput achieved by the proxy device, for each successful iteration, from the perspective of the transmitting device, as well as the receiving device. The agent measures the time taken for the data to arrive at the destination and calculates the throughput by performing simple arithmetic calculations. This test is likely to yield important results concerning the performance of different device types, when used as routing elements. Similarly to all other tests, the results are then passed to the librarian agent for storage, and maintenance.

Hardware used for the preliminary experimentation cycle

Three different device types were selected, including a workstation, a laptop, and a PDA. The hardware characteristics are summarized below:

Workstation

Processor: 1000 MHz Intel Pentium III.
Memory: 512-MB SDRAM.
WiFi: IEEE 802.11b enabled.
Support: USB and Serial.

Laptop

Processor: 750 MHz Intel Pentium II.
Memory: 256-MB SDRAM.
WiFi: IEEE 802.11b enabled.
Support: USB and Serial support.

Handheld (PDA)

Processor: 400 MHz Intel PXA250.
Memory: 64-MB SDRAM; 48-MB Flash ROM Memory.
Support: USB and Serial cable.
Bluetooth: enabled.
WiFi: IEEE 802.11b enabled.
Display: 64K colour TFT LCD.
Power Supply: 1250 mAh Lithium-Ion Polymer removable / rechargeable battery.

Software used for the preliminary experimentation cycle

The Operating System, Java Runtime environment, agent-platform, and agent-based software, used throughout this experimentation cycle, are presented bellow:

Workstation

OS: Microsoft Windows XP Professional.
Java: Sun Microsystems, J2SE v. 1.4.2 Java Runtime Environment (JRE).
Agent platform: Grasshopper v2.2.4 (SE)
Agent-based software: BASS (reference 4.2.1)

Laptop

OS: Microsoft Windows XP Professional.
Java: Sun Microsystems, J2SE v. 1.4.2 Java Runtime Environment (JRE).
Agent platform: Grasshopper v2.2.4 (SE)
Agent-based software: BASS

Handheld (PDA)

OS: Microsoft Pocket PC 2002 (Microsoft, Corporation, 2004).
Java: Insignia Jeode (Insignia, 2004).
Agent platform: Grasshopper v2.2.4 (ME)
Agent-based software: BASS

The agents used include the following:

Proxy throughput agent. As previously mentioned, its task is to forward incoming network traffic from a source node to a destination node. The proxy can be dynamically configured to listen for incoming network traffic to a certain port and transmit it to another, however, the destination IP address must also be specified. It is based on Java's multithreading and can thus accept multiple connections at a single time.

TCP transmitter agent. As previously mentioned, its purpose is to transmit network traffic to a receiving machine, which can be dynamically defined by its IP address and port number. In addition, the buffer size and the number of buffers required to transmit can also be defined.

TCP receiver agent. As previously mentioned, its purpose is to receive incoming network traffic from other transmitting nodes. This agent can be dynamically configured to listen for incoming network traffic to a certain port, using a certain buffer size and number, usually set to be the same as the TCP transmitter agent's.

Throughput calculation agent. As previously mentioned, this agent resides on both the transmitting and receiving devices, and its purpose is to calculate the available throughput provided by the proxy device, from both the perspectives of the transmitter and receiver node.

Gathering agent. As previously mentioned, its task is to hop from device to device and gather the results when they are available.

Librarian agent. As previously mentioned, its purpose is to store data from test results into a local database or file output, and also provide an interface to external objects or agents that require this information.

Memory and battery monitoring agent. As previously mentioned, its task is to constantly measure the battery discharge rate and the memory utilisation. This agent was developed in the Microsoft Visual Studio environment, which provides support for developing code for handheld platforms in Visual Basic. In order to provide interoperability with Java-based agents, this agent stores retrieved information to a number of text files, which can be later used by other agents, and include the following:

BSSCurrentState.txt: This file gets overwritten every 10 seconds with information including the current battery strength, memory utilisation, and the corresponding timestamp. Java-based agents can read this file to obtain the current state information.
BSSMemoryCE.txt: In this file the current memory utilisation and corresponding time-stamp are constantly appended every 10 seconds.
BSSMonCEBattery.txt: In this file the current battery strength and corresponding time-stamp are constantly appended every 10 seconds.

Hardware used for the proxy experimentation cycle

Two different device types were selected, including a workstation and a PDA. The hardware characteristics of these devices are the same as presented in the *hardware used for the preliminary experimentation* section.

Software used for the proxy experimentation cycle

The software used for this cycle of experimentation is shown below. The PDA was equipped

with three different OS, four JVMs, and two agent platforms. Due to the lack of space and the lack of dual OS support in the PDA, each defined OS, JVM, and agent platform was installed in turn for the respective experiment. The two workstations were kept unchanged in terms of software for this cycle of experiments.

Workstation

OS: Microsoft Windows XP Professional.
Java: Sun Microsystems, J2SE v. 1.4.2 Java Runtime Environment (JRE).
Agent platform: Grasshopper v2.2.4 (SE).
Agent-based software: BASS (reference 4.2.1).

Handheld (PDA)

OS: Microsoft Pocket PC 2002/2003 (Microsoft, Corporation, 2004).
Familiar Linux v0.7.2 with GPE (Familiar Project, 2004).
Java: Insignia Jeode (Insignia, 2004).
IBM, J9 (IBM, 2004).
NSIcom, CrEme (NSICom, 2004).
Blackdown, JRE 1.3 (Blackdown, 2004).
Agent platform: Grasshopper v2.2.4 (ME)/(SE).
Agent-based software: BASS.

Temperature variation monitoring agent. As previously described, this agent is responsible for measuring the temperature variation each time the battery drops by 1 (%). For example if the temperature of the device is 25 (C°) when the remaining battery is 75 (%), and the temperature becomes 26 (C°) when the remaining battery becomes 74 (%), the agents calculates the temperature variation being 1 (C°) for that particular measurement.

Heap memory usage monitoring agent. As previously described, this agent constantly monitors the amount of heap memory used by Java objects. It differs from the CPU, memory, and overall utilisation agent in that it does not monitor the JVM's memory requirements, but instead only the memory allocated to running agents.

CPU and memory utilisation agent. This agent calls native code via the JNI in order to obtain current CPU and memory usage per process running in the system as well as total utilisation. This agent can be used to determine whether the device has available resources.

Hardware used for the BASS experimentation cycle

The hardware equipment used for this experimentation cycle include low, medium, medium-high, and high performance devices, and, in particular, a PDA, a server, a laptop, and a

workstation. The PDA's hardware characteristics are the same as in *the hardware used for the preliminary experimentation* section, while the rest are shown in the following:

Workstation
Processor: 1999 MHz Intel Pentium IV.
Memory: 512-MB SDRAM.
Support: USB and Serial.

Laptop
Processor: 1100 MHz, Intel Pentium III.
Memory: 512-MB SDRAM.
WiFi: IEEE 802.11b enabled.
Support: USB and Serial support.

Server
Processor: 450 MHz, Intel Pentium III.
Memory: 256-MB SDRAM.
WiFi: IEEE 802.11b enabled.
Connected to a wireless base station.
Also connected to a broadband router providing Internet access.
Support: USB and Serial support.

Software used for the BASS experimentation cycle

The workstation and PDA were equipped with exactly the same OS, JVM, and agent platform as in the Section *software used for the preliminary experimentation*. The only difference was the server and laptop which were equipped with a different OSs, Microsoft Windows 2000 Professional and Microsoft Windows 2000 Server respectively.

Laptop
OS: Microsoft Windows 2000 Professional.
Java: Sun Microsystems, J2SE v. 1.4.2 Java Runtime Environment (JRE).
Agent platform: Grasshopper v2.2.4 (SE)
Agent-based software: BASS (reference 4.2.1)

Server
OS: Microsoft Windows 2000 Server.
Java: Sun Microsystems, J2SE v. 1.4.2 Java Runtime Environment (JRE).
Agent platform: Grasshopper v2.2.4 (SE)
Agent-based software: BASS (reference 4.2.1)

Group-level agent. This agent is required to run only once and gather system information, at

runtime. System-level information includes the OS architecture, name, and version, the Java Runtime name and JVM version, the main IP address and hostname, and so on. It achieves this by calling the method *getProperty* of the Java *System* class.

Bubble sort agent. The purpose of this agent is to test the processing capabilities of a device's CPU by implementing a complex sorting algorithm. The computational complexity of this algorithm grows exponentially for each added number, at a rate of $O^{(2n-1)}$. In addition, the agent offers multiple complexity dimensions, i.e. 1-Dimensional sorting array, 2D, 3D, and 4D. Accordingly, if a value of 13 is entered for the 4D bubble sort test, this will equate to 28,561 (13×13×13×13) values being sorted, which results to roughly (2,8561×2,8561×2,8561×2,8561)-1 comparisons.

CPU merge agent. This agent is very similar to the bubble sort, however, it implements a less complex sorting algorithm. Its purpose is to test the processing capabilities of the device's CPU by spending, on average, less time. As with the bubble sort agent, this agent can be dynamically instructed on the amount of numbers required for sorting.

Memory and hard-drive test agent. This agent is particularly useful for handhelds, which usually use their memory as their permanent storage medium, and thus this agent exercises their buffering capabilities. It achieves this by performing two similar tests: creation of varying number of files with constant file-sizes and creation of a constant number of files with varying file-sizes. In case of a non-handheld device, this test may be particularly useful in situations where the device is using Virtual memory to accomplish routing tasks.

Internet-connectivity agent. This test attempts to download a HTML page. If successful, it calculates the time taken to connect and download and ultimately shows that there is a connection, otherwise it shows that there is no Internet connection. The HTML page to download is passed to the agent, on creation, by its supervisor agent.

Error packets monitoring agent. The purpose of this agent is to monitor the TCP, UDP, IP protocol data rates and dynamically calculate the error percentage in each. It achieves this by calling native code via Java JNI. Currently, this agent's implementation version does not support handheld devices.

Java threads monitoring agent. This agent is responsible for monitoring the number of Java threads running and the CPU percentage each one of them is utilising. This test may be par-

ticularly useful in situations where local security needs to be enforced. The agent performs calls to native code via Java JNI. This code has been supplied by a cooperative researcher in a similar research field, specifically in agent-based on-line distributed monitoring systems at different levels of abstraction (MAPI) (Bellavista, P., et. al., 2003).

Hardware used for the experimentation of mobile agent migration

As previously mentioned, the superior group consisted of identical workstations far greater in performance than the workstations in the inferior group:

Superior group:

Processor: 2800 MHz Intel Pentium IV.
Memory: 512-MB SDRAM.
WiFi: IEEE 802.11b enabled.
Support: USB and Serial.

Inferior group:

Processor: 1000 MHz Intel Pentium III.
Memory: 128-MB SDRAM.
WiFi: IEEE 802.11b enabled.
Support: USB and Serial.

The laptops used to maintain the database and the region registry, as well as the PDAs used as a client and proxy, had identical hardware characteristics as the respective ones presented in Section *hardware used for the preliminary experimentation*, and are thus not presented here.

Software used for the experimentation of mobile agent migration

The devices were equipped with the same OS, JVM, and agent-platform, as in Section *software used for the preliminary experimentation*. The agents that were implemented for this experimentation cycle are presented and described bellow:

Runner mobile agent. This agent was used in the first phase of this experimentation cycle. Its main task was to migrate along a route, as defined in its itinerary, and measure the time taken to reach the next hop and the total RTT. Its itinerary was fixed and was composed of the agent-platform addresses of each device in the group. Analytically, the agent measured the current time, upon its creation, and stored this information locally. Then the agent requested its migration to the next hop along its itinerary. Once the agent was re-instantiated in the new platform, it measured the current time and compared it with the previous stored value, and thus calculated the time taken to migrate to this new node. In this fashion, at the

end of its itinerary, the agent calculated the total RTT. Each node along the agent's itinerary had its clock synchronised.

Database agent. This agent, as well as the following, was used for the second phase of this experimentation cycle. It was designed to remain stationary on the database laptop, and its main task was to maintain a public database of research articles from journals, conferences, workshops, and tutorials. In addition, it provided a simple search facility that once a query was passed, a number of hits was returned, that included the article's unique identification number, title, and so on.

Client agent. This agent was designed to remain stationary on the client PDA, and its main task was to request a number of articles from the database, based on a default user's query. Since the client PDA was set to be not in direct communication range with the database laptop, the client agent had to pass its query to the appropriate intermediate proxy agent (described below) and receive the resulting matches through it. In order to retrieve the proxy's location, the client initially requested this information from the region registry which was maintained in the nearby laptop.

Client mobile agent. This agent's task was identical to the previously described agent, with the only difference being its mobility feature. In particular, the agent retrieved the proxy's location the same way as its stationary counterpart, but instead of passing its query, it migrated to the actual gateway node. Once there, it contacted the region registry once more in order to retrieve the location information of the database laptop. Then, it migrated to the database and communicated with the database agent locally. Once the results were available, it inverted its itinerary and returned to its home agent platform.

Proxy agent. This agent acted as a gateway between the client agent and the database agent, in the static agent approach. In particular, once it received a query from the client agent, it searched the region registry for the database agent's location. Then, it passed the query to the database agent and gathered the results. Finally, it allowed the client agent to gather the returned hits by calling its *getResults()* method. This agent was bypassed in the mobile agent approach.

Hardware used for metrics simulation experimentation cycle

A single workstation was required for this experimentation cycle. The hardware specification is shown bellow:

Workstation

Processor: 1000 MHz Intel Pentium III.
Memory: 512-MB SDRAM.
WiFi: IEEE 802.11b enabled.
Support: USB and Serial.

Software used for metrics simulation experimentation cycle

The simulation was implemented in Microsoft Excel and several scripts were developed and used for this purpose. The implementation details of each script, along with its purpose, are presented bellow:

Metric variation for O_1. The purpose of this script was to individually vary the CPU, memory, and battery preliminary metrics from 0 to 100, for DT_1 - DT_9 in relation to O_1, and calculate the overall metric values for each variation. The script was designed in such a way so as to gather preliminary information for each type of device and produce overall metric results for each one in turn, in a totally automatic fashion. The implementation was based in Visual Basic (VB).

Metric variation for O_2. The purpose of this script was the same as the previous one, however, it was designed to produce results for O_2. Similarly, the implementation was based in VB.

Metric variation for O_3. Similarly to metric variation for O1 script, however, this one was simulating O3. Similarly, the script was implemented in VB and was designed to conduct the simulation automatically.

Metric variation for O_4. The purpose of this script was to vary the CPU, memory, and battery preliminary metrics from 0 to 100, for DT_1 - DT_9 in relation to O_4, and calculate the overall metric values for each variation.

Metric variation for O_5. The purpose of this script was to vary the CPU, memory, and battery preliminary metrics from 0 to 100, for DT_1 - DT_9 in relation to O_5, and calculate the overall metric values for each variation.

Metric variation for O_6. The purpose of this script was to vary the CPU, memory, and battery preliminary metrics from 0 to 100, for DT_1 - DT_9 in relation to O_6, and calculate the

overall metric values for each variation.

Error packets monitoring test results

Table A.1 presents a snapshot of the laptop's network state, at a random instance. In this particular case, the laptop's network at that point seemed to have suffered a UPD packet loss, however the single packet loss may constitute an insignificant glitch, and is thus difficult to judge from such a small sample. Nevertheless, increased number of packet losses may contribute to an overall unreliable link, and consequently to high demand for bandwidth because of frequent packet retransmissions.

Table A.1: Sample data from error packets monitor test

Network traffic type	
udp_in	0
udp_out	2
udp_err	1
tcp_in	4
tcp_out	8
tcp_con	0
ip_in	4
ip_out	8
ip_pack_err	0
lastUpdated	14:09:46

CPU, memory, and overall utilisation test

This test has the ability to monitor the CPU and memory utilisation at pre-defined time intervals. As an example, a snapshot of this test results, obtained by its execution on the workstation, is presented in Table A.2.

Table A.2: Sample results of the CPU, memory, and overall utilisation test

CPU and Memory utilisation	
totalMemory	523,676
physicalMemoryUsed	214,960
VirtualMemoryUsed	2,317,516
availableMemory	308,716
cpuPercentUsed	91
numOfProcesses	41
numOfThreads	341
percentFreeMemory	58
lastUpdated	14:09:48

Java threads monitoring test

This test is ideal in Java-based agent environments, since it has the ability to monitor each agent's resource-consumptions in terms of CPU, and thus keep a history of possible denial of service activities. In addition, it can be used to identify the roots of high CPU utilisation, observed by the CPU, memory, and overall utilisation agent. Table A.3 presents a snapshot of the individual threads utilisation, while Table A.4 presents their overall utilisation. This test has been executed on the laptop, at a random time.

Table A.3: Sample data captured from the Java threads test, individual Java thread utilisation

threadID	cpuPercentUse	cpuTimeUsed	lastUpdated
1212	0	520	14:09:48
840	0	0	14:09:48
1132	0	0	14:09:48
620	0	0	14:09:48
976	0	0	14:09:48
1572	0	0	14:09:48
1632	0	0	14:09:48
1288	2	550	14:09:48
1628	0	430	14:09:48
1612	0	290	14:09:48
1664	0	0	14:09:48
1668	0	0	14:09:48
1672	0	0	14:09:48
1684	0	0	14:09:48

Table A.4: History of Java thread totals

NumOfThreads	cpuPercentUse	cpuTimeUsed	lastUpdated
11	1	720	13:20:58
15	3	1750	13:21:05
11	4	3010	13:21:11
15	6	4540	13:21:17
11	7	6260	13:21:23
12	8	8210	13:21:28
11	10	10320	13:21:34
9	0	600	13:47:42
9	0	1310	13:47:48
18	2	820	14:09:27
14	4	1960	14:09:34
14	5	3370	14:09:41
14	9	5160	14:09:48

B Appendix - Definition of concepts

B.1 Wireless networks

The most important concepts, which are required for understanding this project's area of research are defined in this Appendix, including: computer networks; wireless networks; ad-hoc networks; ad-hoc routing; software agents; and mobile agents. Then similar novel research in this field is presented.

A computer network allows many computers to communicate with each other, to interchange, and execute programs or data (Roberts, L. G., 1967). Accordingly, the motivation for computer networks is: load sharing; messaging; data sharing; program sharing; remote services; and scientific computation. Similarly, Tanenbaum, A. S., 1996, defines a computer network as an *interconnected* collection of *autonomous* computers. Overall, two computers are said to be *interconnected* if they are able to exchange information, while the term *autonomous* excludes from the definition systems in which there is a clear master/slave relation.

A wireless network is defined as a network where computers can communicate with each other by the use of radio signals (Zorzi, M., 1998). In wireless networks, communication is facilitated by a collection of transmitters, each of which is configured to provide services over a local region (Hurley, S. and Whitaker, R. M., 2002). The main benefit of wireless networks is the ability of users to communicate with each other, or with machines, such as databases and e-mail servers, without being constrained to a fixed location. Wireless networks emerged in 1970s and become popular in the computing industry. This is especially true within the last decade, where wireless networks have been adapted to enable mobility, and wireless interfaces were produced cost effectively, thus giving the opportunity for everyone to create a wireless network. Nowadays, people carry numerous portable devices, such as laptops, mobile phones, Personal Digital Assistants (PDAs), and MP3 players, for their professional and private lives (Frodigh, M., et. al., 2000). Mobility is an important feature that makes wireless networking so essential. In general, wireless networks can be grouped into two categories:

- **Infrastructure networks**. These have fixed and wired gateways, which are typically known as access points. A mobile unit within these networks connects to, and communicates with (by the use of radio signals), the nearest access point that is within its communication radius.

- **Infrastructure-less networks**. These networks have no fixed or central infrastructure, and can thus be formed on a temporary basis. Mobile devices within these networks are free to move about in an arbitrary fashion, and act as routers for other nodes, so that multi-path communications can take place (Royer, E. M. and Toh C. K., 1999).

B.2 Ad-Hoc Networks

There are many definitions in the literature for infrastructure-less networks, commonly known as ad-hoc networks. However, each definition is specifically tailored to suite the context in which it is defined. Accordingly, Jonsson, U. and Alriksson, F., 2000, consider an ad-hoc network, as an alternative solution to a fixed network, which can be formed on a temporary basis, and provide the following benefits: easy to setup; and can operate without any fixed infrastructure. The temporary aspect of ad-hoc networks is also highlighted in (Frodigh, M., et. al., 2000), who suggest that there is no need for a central administration. Wang, X., et. al., 2001, further suggest that routing and location management is entirely left to the participating mobile nodes, which have to use their wireless interfaces to route data packets in a multi-hop manner. An important issue that has not covered by these definitions is suggested in Rajaraman, R., 2002, that is, ad-hoc networks have limited capabilities, as participating mobile devices often rely on battery power for operation. Ramarathinam, V. and Labrador, M. A., 2002, further suggest that because of the lack of an infrastructure, mobile nodes must, at least, take one of the following roles: end system; a server; a router; a gateway; or all of them at the same time. In the context of this thesis, an ad-hoc network is defined as:

> **… a computer network that is typically formed in case of an emergency, which does not have a fixed infrastructure that nodes could rely on for location management and routing, and thus nodes have to create their own structure in a dynamic way. Accordingly, nodes are required to use their wireless interfaces for forwarding data packets to the next hop, along a multi-hop fashion, and participate equally in the tasks delegated to them by the ad-hoc routing protocol.**

Figure B.1 illustrates an abstract representation of a basic ad-hoc network. Each device defines a wireless domain, where its radius is equal to the transmitting emission distance of its wireless interface. The middle device is situated in the edges of wireless domain *A*, defined by the device on the left, and wireless domain *B*, defined by the device on the right. Thus, the middle device merges wireless domain *A* and *B* by acting as a routing device.

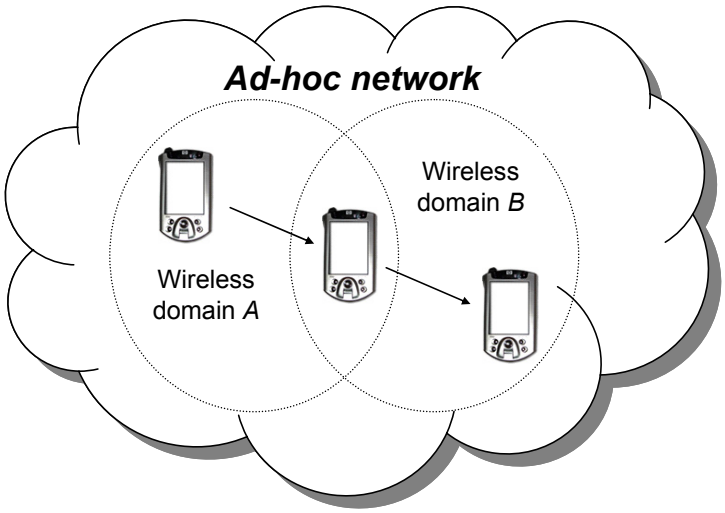

Figure B.1: A typical ad-hoc network

B.3 Ad-hoc Routing

Traditional ad-hoc routing methods, initially designed for fixed networks, are typically based on the shortest path calculation (Schwartz, M. and Stern, T. E, 1980), where each node applies a shortest path algorithm from itself to all other destination nodes, and transmits its data packets through the shortest route. However, routing protocols for fixed networks have not been designed specifically to provide the kink of dynamic, self-starting behaviour required for ad-hoc networks (Perkins, C. and Bhagwat, P., 1994). Two of the most fundamental algorithms, are:

- **Link-state** (McQuillan, J. M., et. al., 1980). Each node in the network maintains a database describing the complete network topology with a cost associated for each link. Database maintenance is achieved by each node periodically broadcasting the link-state costs of its neighbouring nodes to all other nodes by the means of a flooding strategy. Nodes use the information from periodic broadcasts to update their current view of the network topology, and their link-state information by applying a shortest-path algorithm to choose the next hop node for each destination.
- **Distance vector**. This approach is based on the fundamental principles of the Distributed Bellman-Ford (DBF) algorithm (Bertsekas, D. and Gallager, R., 1987). Each node *i* maintains a set of distances for every destination *x*, where *j* ranges over the neighbours of node *i*. Node *i* selects a neighbour, say k, to be the next hop for *x* if:

$$D_{ik}^{x} = min_{j}\left(D_{ij}^{x}\right) \quad \text{(B.1)}$$

Nodes periodically disseminate their current estimates of the shortest distance to every node in the network. When a node receives a routing vector from a neighbouring node, it updates its distance to all other destinations via this neighbour.

Link-state methods often suffer from routing loops, however, they are temporary. This behaviour is usually linked to inconsistent views of the link costs maintained on the nodes' databases, caused by long propagation delays, partitioned network, and so on. Distance vector methods are computationally more efficient, requiring less storage space, and are relatively easier to implement when compared to link-state methods. However, the underlying algorithm (DBF) of distance vector is known to cause the formation of long-lived routing loops, as well as temporary loops, which respond slowly to link failures, and provide no guarantees for successful termination (Cheng, C., et. al., 1989). The primary cause for formation of routing loops is that nodes choose their next hop in a totally distributed manner, based on information which may be stale, and thus incorrect. The slow response to link failures could be a direct result of these routing loops formation. Proposed modifications (Jaffe, J. M. and Moss, F. H., 1982, Garcia-Luna-Aceves, J. J., 1989) to the basic DBF algorithm were shown to eliminate the formation of routing loops by providing some form of inter-nodal coordination protocol where each node is required to participate.

Link-state and distance-vector algorithms, unfortunately, impose scalability problems in large ad-hoc networks (Shankar, A. U., et. al., 1992a), where frequent periodic updates may cause large amounts of network overhead, and thus consume a valuable amount of the available bandwidth. Shared workload by each participating node, even by the weakest performance nodes, such as PDAs, can rapidly increase their utilisation status and decrease their power capacity. Several routing methods have been proposed in the literature, which either aim to enhance these algorithms, or totally replace them (*see* Appendix D).

B.4 Software Agents

In general, agents are software programs which aim to automate user tasks and have significant applicability in mobile and distributed applications (Jennings, N. and Woolridge, M., 1998). White, J., 1997, distinguished software agents from normal programs due to the agents' abilities to execute in distributed computing environments and their ability to supply domain specific knowledge in automating user-tasks. Hayes-Roth, B., 1995, suggest that intelligent agents typically perform three functions: sense the environment they live in; reason

about the environment; and perform actions that affects the environment. A similar viewpoint is suggested in Maes, P., 1995, that is, intelligent agents realise the purpose of their existence by sensing and acting autonomously in the environment they live in. Smith, D. C., et. al., 1994, suggest that two properties, which distinguishes intelligent agents from subroutines, are: persistency; and goal-orientation, that is, agents autonomously decide on their own actions and are designed to accomplish small tasks. Wooldridge, M. J. and Jennings, N. R., 1995, propose that intelligent agents typically have characteristics, such as: autonomy, that is, they can operate without any direct intervention from humans or other software; social ability, that is, they communicate with other agents; reactivity, that is, they sense their environment and perform actions based on this knowledge; and pro-activeness, that is, they are goal-oriented. In the context of this thesis, an intelligent agent is defined as:

> ***… a software entity that is typically light-weighted, and precise about its goal, which is often required to monitor its environments and act upon critical changes, and possibly cooperates with other agents to serve an overall goal.***

The fundamental properties of an intelligent software agent are (Franklin, S. and Graesser, A., 1996):

- **Reactive** (sensing and acting). Monitor the environment and act upon critical changes.
- **Autonomous**. Controls its own actions.
- **Goal-oriented** (proactive purposeful). Plans its actions in relation to its goal.
- **Temporally continuous**. It is a continuously running process.
- **Communicative** (socially able). Communicates with other agents including people.
- **Learning** (Adaptive). Changes its behaviour based on its previous experience.
- **Flexible**. Its actions are not scripted.
- **Character**. Believable *personality* and *emotional state*.

B.5 Mobile Agents

In general, mobile agents are software agents that have one extra feature, which is mobility of code, state, and, possibly, execution state. Thus they inherit all, or some, of the intelligent software agent properties. Tripathi, A. R., et. al., 2000, propose that users should be kept fully responsible for their mobile agents' actions, as the autonomy of agent migration is an issue that may raise security concerns, whereas, Pham, V. A. and Karmouch, A., 1998, suggest that mobile agents act on behalf of the user, as well as other entities that need their

services. Karjoth and Posegga, 2000, propose that a mobile agent usually has knowledge on its user's ideas and problem-solving techniques in a specific context, which then efficiently implements in an automated manner. Harrison, C. G., et. al., 1995, as well as White, J., 1997, identified mobile agent essential models, which include: a life-cycle model; a computational model; a security model; a communication model; and, finally, a navigation model.

In the context of this thesis, an intelligent agent is defined as:

> ***… a software agent that has some, or, all characteristics of an intelligent agent, and, in addition, it has the ability to initiate its own migration, that is, it data state, code, and possibly, its execution state, and thus transfer itself to another agent-enabled host and resume execution on the new host.***

The mobile agent paradigm greatly differs from the traditional client-server communications model (Figure B.2).

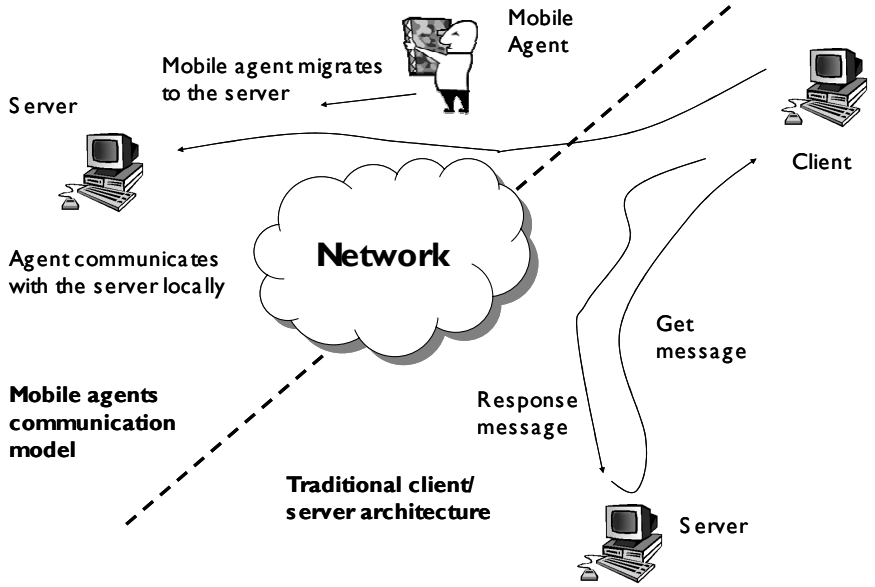

Figure B.2: Mobile agents versus client-server model

As shown in Figure B.2, in the client-server case, the client requests a service from the server, and the server replies back with the results, while, in the mobile agent case, the actual computation is transferred to the server computer, and all interactions are performed locally.

B.6 Mobility prediction

In an attempt to improve performance of traditional routing protocols, Lee, S.-J., et. al., 2001, proposed the incorporation of mobility prediction metrics with standard on-demand routing protocols, in order to reduce the control overhead generated through route discovery, and the route maintenance process. The underlying idea of their proposal is based on the assertion that by exploiting non-random behaviours of the mobility patterns that mobile users exhibit, the future state of the network topology can be predicted, and route reconstruction can be performed proactively. Mobility prediction can be achieved using a number of different methods, such as by utilising the location and mobility information provided by GPS, or by periodically measuring the transmission power samples from received data packets from a mobile node's neighbours. With GPS, if the mobility speed, direction of movement, and the propagation range of two mobile nodes is known (from GPS), the remaining connection time of these two nodes can be approximately predicted. In contrast, according to the second method a mobile node can compute the rate of change in the transmission power of its neighbours, and can thus forecast the time expected for the transmission power to drop below an acceptable level.

Accordingly, mobility prediction information can be used to estimate the link expiration time (LET) of all possible links between any two adjacent mobile nodes, and the route expiration time (RET) of each route, maintained in a node's route cache. Based on this prediction information, routes can then be reconstructed before they actually expire. Thus, there is scope for providing a seamless connection service by reacting before the connectivity breaks. The underlying assumptions are: a free propagation space, where the received signal strength solely depends on its distance to the transmitter; and nodal clock synchronisation by either using the network time protocol (NTP) (Mills, D.L., 1991) or the GPS clock itself.

Rerouting before link disconnections may assist in minimising packet losses and further increase the overall performance of the routing protocol. A simple example is illustrated in Figure B.3, where, assuming that the LETs are known, the RETs can be calculated, and thus assist the routing protocol in finding the potentially more robust routes. For example, route $A \rightarrow B \rightarrow C \rightarrow E \rightarrow F$, provides a total RET of $5 + 4 + 3 + 7$, which is *19*. However route $A \rightarrow B \rightarrow D \rightarrow E \rightarrow F$, provides a RET of $5 + 6 + 5 + 7$, which is *23*, and, therefore, this route is chosen over the former, as the route is expected to be available for a longer time based on the gathered information.

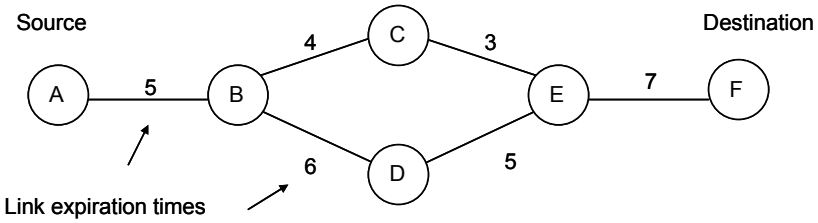

Figure B.3: Routing using prediction based on LET and RET

In their later work, Su, W., et. al., 2001, applied their mobility prediction mechanism to three representative ad-hoc routing protocols, including: on-demand multicast routing protocol (ODMRP) (Lee, S. J., et. al., 2002); highly dynamic-sequenced distance-vector DSDV (Perkins, C. and Bhagwat, P., 1994); and multicast routing protocol (AMRoute) (Bommaiah, E, et. al., 1998). Simulation results showed that with prediction enhancements in place, more data packets were delivered to their destination, while the routing overhead was considerably reduced in low-mobility simulations and in high-mobility control packets were utilised more efficiently. In addition, the routing protocols with mobility enhancements were shown to effectively choose more stable routes that did not become invalid due to node movements.

An ad-hoc, node proximity model, proposed by McDonald, A. B. and Znati, T., 2000, enhanced the performance of routing algorithms by selecting stable routes, and further facilitated mobility-adaptive dynamic clustering. The proximity model was designed to provide a quantitative metric in order to reflect to the future stability of any given link in an ad-hoc network, with minimum information gathering requirements. Accordingly, the initial baseline *link availability* is calculated by assuming random-independent mobility similarly to (Fasbender, A., et. al., 1999). The model then adapts future computations depending on the expected time-to-failure of the link based on the independence assumption and a parameter that reflects the environment.

B.7 Clustering

The clustering process involves the organisation of mobile nodes with similar geographical position in an ad-hoc network, which are grouped into adjacent or disjoint clusters. A cluster-head is elected in order to provide coordination of data transmissions within its own cluster. The wireless range of a cluster-head defines a cluster, and thus every node within its transmission range belongs to this cluster. Therefore, every node within a cluster can communicate with the cluster-head, and, possibly with each other. Nodes which are situated in

the edges of two or more clusters are called gateways and are responsible for inter-cluster routing. A cluster-head ad-hoc routing protocol can take advantage of the clustering formation, and thus efficiently minimise the flooding traffic during route discovery by allowing packets to be routed only through cluster-heads and gateways. In addition, such a structure may also provide a convenient framework for the development of important features such as wireless channel separation (among clusters), routing, and bandwidth allocation (Chiang, C.-C., et. al., 1997).

Cluster-formation

The main objective of the cluster-formation process is to create a feasible interconnected set of clusters covering the entire node population. A good clustering algorithm should not change the clustering configuration too drastically when nodes are moving slowly, and should maintain cluster-heads, as much as possible. Two fundamental clustering algorithms have been initially proposed in the literature: lowest-ID (Ephremides, A., et. al., 1987); and highest-connectivity (Parekh, A. K., 1994), and they have both been later revised in (Gerla, M. and Tsai, J. T.-C., 1995).

Lowest-ID algorithm

Each node is assigned a distinct ID which it periodically broadcasts, alongside with the list of nodes it can hear. The node with the lowest ID in a neighbourhood becomes a cluster-head. The detailed algorithm is:

- Each node is assigned a distinct ID.
- Each node periodically broadcasts its ID, and the list of nodes that it can hear (including, itself).
- A node which only hears nodes with IDs higher than itself is a cluster-head (CH).
- The lowest-ID node that a node hears is its clusterhead, unless the lowest-ID specifically gives up its role as a clusterhead (deferring to a yet lower ID node).
- A node which can hear two or more clusterheads is a gateway.
- Otherwise, a node is an ordinary node.

Figure B.4 illustrates the clustering process using the lowest-ID algorithm to an example topology. For instance, in the lower cluster, node *4* has been elected a cluster-head as it has the lowest-ID from its neighbouring nodes (*7* and *9*).

Highest-connectivity algorithm

Each node periodically broadcasts the list of nodes it can hear, including itself. A node is

elected as a clusterhead if it is the most highly-connected. The detailed algorithm is:

- A node is elected a cluster-head, if it is the most highly-connected node of all *uncovered* neighbour nodes (in case of a tie, the lowest ID wins).
- A node which has not elected its cluster-head, yet, is an *uncovered* node, otherwise it is a *covered* node.

Figure B.5 illustrates the clustering process using the highest-connectivity algorithm to an example topology. For instance, in the middle cluster, node *8* has been elected a cluster-head as it is the most highly connected node, among its neighbours (*9*, *3*, *1*, *6*, *10*, and *2*).

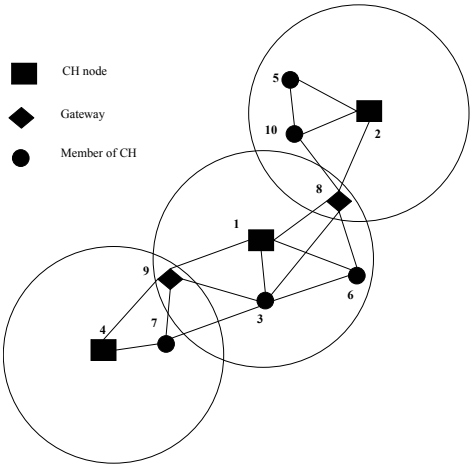

Figure B.4 (Gerla, M. and Tsai, J. T.-C., 1995): Application of the highest-connectivity algorithm to a random topology

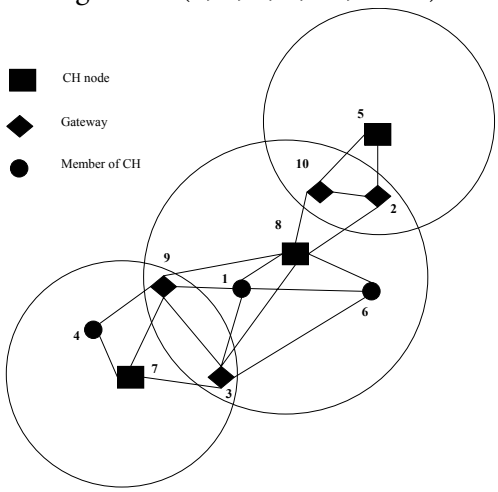

Figure B.5 (Gerla, M. and Tsai, J. T.-C., 1995): Application of the highest-connectivity algorithm to a random topology

Clustering properties

The following properties apply to both lowest-ID and highest-connectivity clustering algorithms:

- No cluster heads are directly linked.
- In a cluster, any two nodes are, at most, two hops away, as the clusterhead is directly linked to every other node in the cluster.

Thus, each node can either be a cluster-head, or directly linked to one or more clusterheads. In addition, only one cluster-head is allowed per cluster. The clustering algorithm must be

performed as rapidly as possible, so that each clusterhead can take and maintain control of its members efficiently. Chiang, C.-C., et. al., 1997, has shown that, in most situations, the lowest-ID algorithm performs better than the highest-connectivity in terms of cluster stability. Thus, lowest-ID is the most stable algorithm with the least cluster-head changes. In addition, the authors proposed a number of modifications to the lowest-ID, and highest-connectivity algorithm, in order to improve its performance. The modified version is named least cluster-head change (LCC) and its operation is:

- A clustering algorithm, such as lowest-ID or highest-connectivity, may be initially used to form the clusters.
- A non-cluster-head never challenges the status of an existing cluster-head.
- Only when two cluster-heads move next to each other, one of them looses the cluster-head role, such as the one with the highest ID, or with the least neighbours.
- When a non-cluster-head node moves out of its cluster, and does not enter into any existing cluster, it forms a new cluster and becomes the cluster-head of this cluster.
- Member nodes which leave their cluster(s), will have to re-execute the clustering algorithm.

This has the potential to significantly reducing the number of cluster-head changes, which often occur due to re-clustering. The convergence time of the proposed modified algorithm is O(d), where d is the diameter of the whole ad-hoc network, in terms of hops, and thus has good scalability.

B.8 Cluster-head metrics

The lowest-ID algorithm uses the node's ID metric, which may simply be the node's IP address, whereas the highest-connectivity algorithm uses the node's connectivity metric, which is simply the number of mutual bi-directional links among the node, and its neighbours. However, since mobility can significantly influence the stability of clusters, it is logical to assume that mobility should be a key factor in clustering formation process. In other words, nodes with high mobility patterns should not be chosen to become cluster-heads, as their rapid movements may result in frequent cluster rearrangements. A study which proposed the usage of mobility metrics for cluster formation and selection can be found in (Basu, P., et. al., 2001). The authors proposed a novel mobility metric, called MOBIC, which is based on the ratio of power levels due to successive receptions at each node. The calculation of the MOBIC metric does not involve a GPS, and generally assumes that any signal strength

measured by a receiving node, *PxPr*, directly indicates the distance between the transmitter and receiver node pair. Although this method cannot be used to accurately measure distance in a real-life application, successive power measurements of two or more consecutive transmissions from a neighbouring node may allow the calculation of the relative mobility between the nodes. The relative mobility metric, at a node *Y* with respect to *X*, is defined as follows:

$$M_Y^{rel}(X) = 10log_{10}\frac{RxPr_{X\to Y}^{new}}{RxPr_{X\to Y}^{old}} \tag{B.2}$$

$$\text{If } RxRr_{X\to Y}^{new} < RxRr_{X\to Y}^{old}, \text{ then } M_Y^{rel}(X) < 0 \tag{B.3}$$

A negative value of the relative mobility between two nodes may thus indicate that the nodes are moving away from each other. In case of a positive value, it may thus indicate that these nodes are moving closer to each other. A node *Y* having a number of neighbours *n*, can calculate *m* such values for M_Y^{rel}. It can then calculate the *aggregated* local mobility (M_Y) by calculating the variance (with respect to zero) of the entire set of mobility samples $M_Y^{rel}(X_i)$, where X_i is *Y*'s neighbour:

$$M_Y = var_0\left\{M_Y^{rel}(X_j)\right\}_{j=1}^{m} = E\left[\left(M_Y^{rel}\right)^2\right] \tag{B.4}$$

Accordingly, every node *Y* can measure the power levels of successive transmissions from all its neighbours, and also calculate the *aggregated* local mobility (M_Y). The underlying principle of the *aggregated* local mobility is that node *Y* may infer its mobility pattern in respect to its neighbouring nodes. In particular, a low M_Y value may indicate that *Y* is relatively less mobile, with respect to its neighbours, while a high value of M_Y may indicate that *Y* is highly mobile, with respect to its neighbouring nodes. Thus, a mobile node *Y* with lower *aggregated* relative mobility than its neighbours should be favoured in becoming a cluster-head. The authors suggest that the calculation of this mobility metric is a simple task, and can be achieved by any mobile device in a distributed way, whereas, the incoming signal power measurements can be easily achieved with existing hardware.

A number of simulation experiments were conducted using *ns2* simulator, and proved that MOBIC can reduce the rate of cluster-head changes by 33% when compared to the standard lowest-ID algorithm. Thus, the mobility criteria for cluster-head selection may provide a better metric than the node ID, which could allow the formation of a more stable

structure, with minimal cluster-head changes.

B.9 QoS for Ad-hoc networks

Almost every routing protocol discussed so far, utilises a standard shortest-path mechanism in order to decide on *optimal* routes. This typically requires every route to be represented by a metric which is essentially the number of intermediate nodes that need to be traversed to reach the destination. For example, if node A maintains two routes for a destination node B, where the first route defines a path of five hops, while the second route defines an alternative path of four hops, the shortest path in terms of hops will be selected, which, in this case, is the second one. Also, there is normally minimal, or no, support for multiple redundant paths. The most common case, involves the usage of multiple redundant paths in the case of primary shortest-path failures. In this way, the source node can immediately resume transmission over an alternative path, and thus save additional latency from an on-demand route discovery process.

Although shortest-path routing, and single route maintenance, are normally two fundamental mechanisms found in most traditional ad-hoc routing protocols, they considerably limit the support a routing protocol can offer. The goal of QoS is thus to provide certain guarantees on the ability of a network route to support the transfer within a certain time limit, and offer the required throughput, latency and error rate. The shortest-path algorithm assumes that participating devices are of equal strength, such that they can offer the same throughput, perform equally reliably, have the same utilisation status; and battery capacity, at any given time (Migas, N. and Buchanan, W. J., 2005). In addition, single routes are normally used for routing instead of multiple routes, and, thus, the expense of route rediscovery, in the case of primary failure, can be high. Furthermore, even if multiple redundant paths are available, they are not utilised to their full potential.

Recently, a number of innovative methods have been proposed in the literature, which aim to provide path redundancy and QoS support by keeping the network overhead low. Papadimitratos, P., et. al., 2002 proposed a new ad-hoc routing protocol, called Disjoint Path Selection Protocol (DPSP), which supports communication between networked nodes over multiple diverse paths. The goal of DPSP is to determine a small number of diverse paths that remain operational with high probability, and can be used simultaneously by the communicating nodes. Initially, DPSP's path selection algorithm constructs a set of reliable disjoint paths, iteratively. The first step involves finding the most reliable path on the given graph. When no interlacing is present, the newly found path is appended to the existing path set. A non-interlacing path P_i is defined as a path that is already edge-disjoint to all $P_j \in D_k$,

which is an existing path set. In the case of an interlacing path, the algorithm makes the decision on whether by removing the interlacing, and consequently re-arranging D_k, leads to a more reliable path set, and it then proceeds, based on this decision. The authors claim that their protocol can discover a set of paths with significantly longer lifetimes than those found by previously proposed protocols, such as the signal stability adaptive (SSA) and associativity-based routing (ABR) (Toh, C., 1996). They further claim that DPSP is flexible, has easy-to-compute metrics, allows for fast convergence, and provides other benefits, such as less frequent route discoveries; significantly lower routing overhead; lower transmission delays; and improved load balancing.

A novel QoS-aware resource discovery framework for ad-hoc networks has been proposed by Liu, J., et. al., 2002, which is built on the application layer and aims to provide generic and efficient tools for QoS-aware resource discovery. Initially, each node is assigned to one or more of the following three roles:

- **Client**. A node that initiates a query for resource discovery and uses resources.
- **Resource provider** (**RP**). A node that provides resources for clients.
- **Discovery agent** (**DA**). A node that collectively maintains directory information of the resources using hash indexing. Also, dynamically partitions the network into dynamic domains, and monitors the QoS information of the RPs in its domain and responds to discovery queries from clients in the domain. Finally, they exchange messages between other DAs, concerning registration and query information.

The DA nodes are elected using an algorithm similar to lowest-ID, and periodically broadcast their addresses. A non-DA node sets its home to a DA node in close proximity, and joins that DA's domain. A DA is also responsible for QoS information collection and prediction. The information collected includes the CPU usage and available memory of a RP, and the path delay between two nodes. The authors proposed that each DA node should be equipped with a GPS and, that, by using the universal time coordinate (UTC) service along with time-stamps, they could efficiently measure the delay. Preliminary simulation results showed that the proposed framework enhances QoS-awareness compared to traditional centralised approaches, and further achieves lower query latency.

Furthermore, an innovative routing scheme called trigger-based distributed routing (TDR) for supporting real time QoS traffic in mobile ad-hoc networks (MANETs) was proposed by De, S., et. al., 2002. TDR is a hybrid routing algorithm, which incorporates link failure prediction, and provides real-time support, while keeping the network overhead low. The underlying principle is, that, in order to reduce network overhead, only active routes

need to be maintained, and GPS-based location information of the destination should be used to selectively broadcast reroute queries when a link failure is imminent. Another advantage of this failure prediction-based alternate route discovery is the fact that it avoids the maintenance of unnecessary routes, and thus reduces the size of nodal databases. However, this protocol imposes an extra nodal overhead which is attributed to the computation for selecting appropriate nodes to forward route requests. A number of simulation experiments were conducted in order to study and compare the protocol's performance with traditional QoS protocols for ad-hoc networks. The *prediction-based* TDR outperformed *prediction-less* QoS routing protocols, such as E-AODV (Perkins, C. E., et. al., 2000) and DQoSR (Chen, S. and Nahrstedt, K., 1999).

A different approach to dynamic QoS allocation for multimedia ad-hoc networks was taken by Wu, H. K. and Chuang, P. H., 2001, who propose the usage of carrier sense multiple access/collision avoidance (CSMA/CA) medium access protocol, along with reservation and control mechanisms in order to guarantee QoS in ad-hoc networks. Their scheme uses a link-state routing protocol, where each node broadcasts its neighbour list to all other nodes. The standard Dijkstra's algorithm (Sedgewick, R., 1983) is then used to find the shortest route. Along with neighbour lists broadcasted by nodes, the reservation tables are also broadcasted. In this way, a network node builds the network topology along with QoS information. In addition to periodic exchange of reservation tables, they are also dynamically broadcasted each time a new reservation is made. A node that accepts a new incoming request for a QoS reservation, checks its QoS table, and determines whether a new reservation, with the specified demand, can be established, or not. Two simulation experiments were conducted in their study, the first on a single-hop topology, and the second on a multi-hop topology. In the single-hop simulation, the system was shown to be able to guarantee transmission performance, however, in the multi-hop simulation, although the network traffic was controlled, the request to send/clear to send (RTS/CTS) message exchange was not proved to be sufficient for solving the hidden node problem. In addition, the average packet loss rate was higher than the single-hop simulation.

Relative research work in QoS, can be found in (Hu, Y. C. and Johnson D. B., 2004), who proposed SQoS, a secure form of QoS-Guided route discovery for on-demand, ad-hoc network routing, which uses symmetric cryptography to secure route discovery requests. According to the authors, the symmetric cryptography is preferable in this area, due to its faster execution, when compared to asymmetric cryptography, which is typically three to four times slower. In general, QoS-Guided route discovery has an exponential relation to the number of network nodes, for a route discovery to a single destination, however, SQoS reduces the overhead to a linear relation.

Even though these QoS methods are shown to improve over traditional techniques, they take little, or no consideration, of key metrics, such as node computation strength, current utilisation status, and battery level. In an attempt to prove the need for these parameters to be taken into account as well, Buchanan, W. J., et. al., 2004a showed that wireless devices not only have huge performance differences in calculation speed, but, also, in network transmission reliability, and throughput. Furthermore, it was shown that heavy routing imposes higher resource consumptions, in terms of CPU, memory, and battery discharge, to resource constrained devices, than in devices of a fitter category (Buchanan, W. J., et. al., 2004a). Similarly, it was demonstrated that the maximum throughput, which can be offered by an ad-hoc routing device, is highly-dependant on the device's hardware characteristics (Migas, N., et. al., 2004b, Migas, N., et. al., 2005). Throughput was also shown to be directly linked to the Operating System (OS) and Java Virtual Machine (JVM) used, in the case of the proxy software being developed in Java. Furthermore, both studies provided clear evidence that the battery discharge rate is dependent on the routing scenario the device is trying to accomplish.

B.10 Agent-based ad-hoc security

In contrast to their wired counterparts, mobile ad-hoc networks have no clear line of defence, and therefore traditional security solutions are hardly applicable. In addition to proposals for agent-based solutions to ad-hoc routing, there have been numerous proposals for enhancing security with agent-based solutions. Some of these include (Wang, Y. and Pang, X., 2003, Ping, Y., et. al., 2004, and Peysakhov, M., et. al., 2004).

Wang, Y. and Pang, X., 2003, investigated a parallel dispatch model with secure route structures for protecting the dispatch route information of mobile agents. In contrast to the sequential mobile agent migration model, the proposed model facilitates efficient dispatching of agents in a hierarchical manner, while simultaneously providing route security by exposing minimal route information to hosts. Briefly, the proposed model is a typical parallel dispatch model, where each parent agent, an agent that created a clone of itself, can dispatch two children agents, the cloned copy of the original agent, resulting in a tree structure. In order for the proposed model to provide security and robustness, cryptographic techniques were applied to the basic binary dispatch model. Employed mechanisms included the popular public-key encryption algorithm, signature generation algorithm, and X.509 authentication framework (Pfleeger, C. P., 1997). Although the authors suggested an e-commerce application of their scheme, where e-shops could possibly deny the dispatch of a mobile agent, if the agent's itinerary is known to the e-shops, to its next hop (e-shop) for competitive reasons,

another potential application could possibly be ad-hoc routing.

In routing protocols, such as Ant-AODV, and in protocols proposed in (Bandyopadhyay, S. and Paul, K., 1999, RoyChoudhury, R., et. al., 2000, Denko, M. K., 2003), where mobile agents are responsible for routing, the scheme proposed by Wang, Y. and Pang, X., 2003, could be beneficial. In particular, hiding routing agent's sensitive information, such as their dynamic itinerary, could enhance the security and robustness of an agent-based routing protocol, in the sense that a malicious node would not be able to exploit the routing agent's behaviour so as to alter its decision making. However, there are certain limitations in the proposed scheme, imposed by the assumption of an existing secure environment, including the generation, certification, and distribution of public keys. A totally-distributed and scalable security solution for ad-hoc networks has been proposed in Ping, Y., et. al., 2004. The architecture relies on a multi-agent system to provide functions similar to those of the body's immune system:

- It is totally-distributed, consisting of many components which interact locally in order to provide global protection, and there is thus no central control, and consequently no single point-of-failure.
- It is dynamic in the sense that, whenever, necessary, individual components are continuously being created, destroyed, and circulated throughout, allowing the system to discard useless and dangerous components, while improving on existing ones.
- It is adaptable by learning to recognise and respond to new enemies, and retains a memory to facilitate future responses.
- It is autonomous, as agents take autonomous decisions on invaders based on the collected information, and act independently.
- It is based on behaviour analysis and not on standard node ID to isolate the invader node, and, thus, even if the node, changes its ID, such as its IP address, it will be found, as long as its behaviour remains the same.
- It is scalable, as the computational requirements are not increased by the addition of new nodes into the ad-hoc network.

The overall architecture of the proposed system is illustrated in Figure B.6.

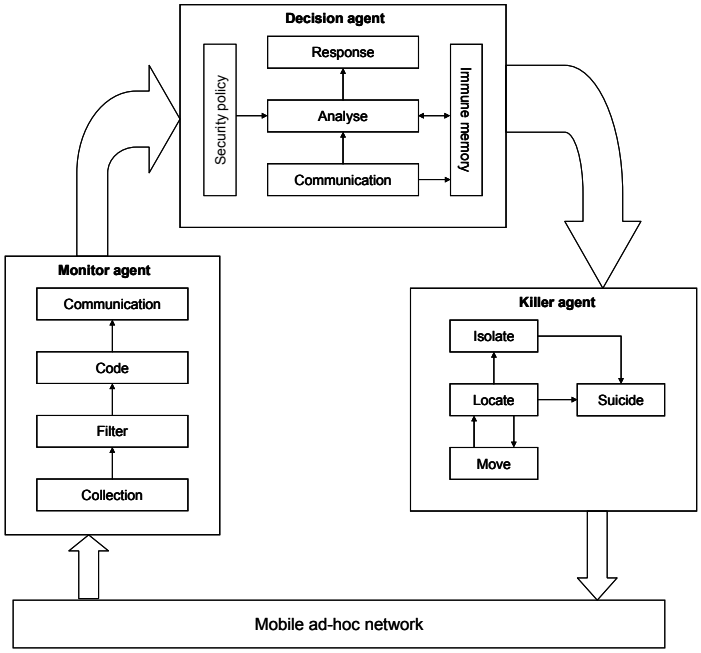

Figure B.6: An immune-like system with mobile agents for security in mobile ad-hoc networks

The system's three essential components are:

- **Monitor agent**. It resides on each node and monitors its neighbouring nodes' behaviour. The collected information is passed to the filter component, which reduces the amount of information collected. The filtered information is then passed to the code component, which analyses the information and codes them by number, such as 1 - RREQ sent.
- **Decision agent**. These are distributed over the network in order to save network resources, and compose the core of the system by analysing information collected by monitor agents.
- **Killer agent**. They are created on-demand by decision agents, to isolate found invader node(s). These agents can get into the invader's neighbouring area and surround it. They respond by cutting off the routing requests of the invader and dropping its transmitting data packets.

A utility-based model for balancing information availability and integrity in agent systems running on ad-hoc networks was initially proposed in (Artz, D., 2003). The model was further advanced to handle multiple compromised hosts and included the introduction of weight labels to the host and agent topology graphs (Peysakhov, M., et. al., 2004). The basic principle of this approach is, that, agents reasoning about how they communicate over the

mobile agent system's underlying network, and by having some sort of build-in network awareness capability, it could possibly lead to major improvements concerning the survivability and efficiency of the mobile agent system. Among other potential applications of this model, the *compromised-node* problem, that is, a node launching a denial-of-service attack, and specifically how agents should react after the intruder has been detected, was investigated by the authors. The process is:

- Agents cooperatively identify the magnitude of the effect which the compromised host has on the integrity of messages among agents.
- Agents cooperatively select a security policy with respect to message integrity, and the compromised host.

The set of security policies which can be implemented by the agents are the following:

- **Reroute**. This operation generates a new set of network routes which try to avoid the compromised host.
- **Ignore**. This operation has no effect on the compromised host or any other host in the network, and thus the state of the network remains the same.
- **Remove**. It has the effect of totally removing any links to/from the compromised host, so that it is no longer capable of participating in the agent's system underlying network.

Figure B.7 (a) illustrates an example topology with eight ad-hoc nodes, and eight links between them. It is assumed here that the compromised host is H_8. Figure B.7 (b) illustrates all routes to H_1 after the reroute operation has been applied to H_8. As shown, possible routes involving the host pair H_5 - H_8 were not included in the new set of network routes. In contrast, the *ignore* operator had no effect on the network state, as it can be seen in Figure B.7 (c). Removing the compromised host H_8 from participating in the agent system's underlying network, resulted in a single route to H_1 ($H_6 \rightarrow H_5 \rightarrow H_4 \rightarrow H_3 \rightarrow H_2 \rightarrow H_1$), while H_8 and consequently H_7 were cut off from being able to communicate with H_1 and any other node in the network.

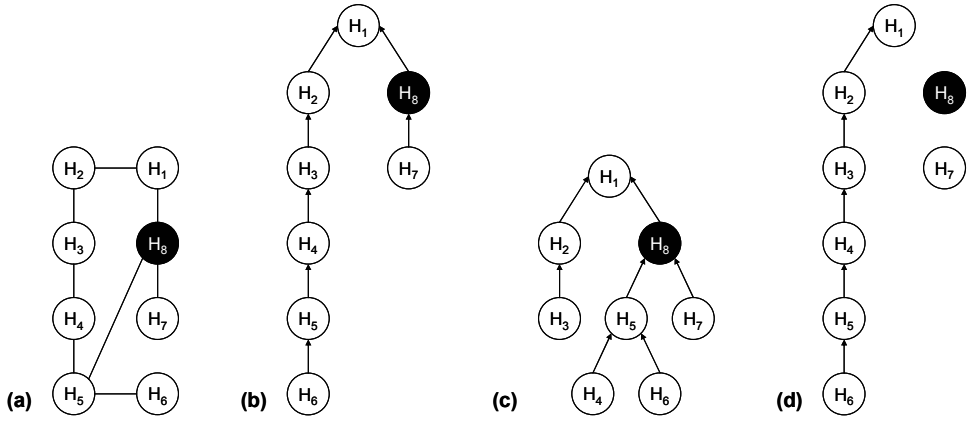

Figure B.7: Agent decisions on compromised host, (a) Initial topology, (b) reroute (N, h_c), (c) ignore (N, h_c), d) remove (N, h_c)

In order to provide the agents with the ability to reason on whether a host has been compromised or not, the agents can sense and observe the signal strength, signal to noise ratio, delay, and jitter. Agents can then decide on which security policy should be applied by performing a series of mathematical calculations which are thoroughly presented in (Peysakhov, M., et. al., 2004).

C Appendix - Intelligent software agents

C.1 Introduction

This Appendix presents the literature review in the area of intelligent mobile agents. Initially, a historical evolution of the mobile agent paradigm, along with its advantages in the context of distributed systems, and especially wireless ad-hoc networks, is presented. Overall, intelligent agents have novel characteristics, such as asynchronous communications, autonomy, goal-orientation, reactiveness, and mobility of code and data state, which makes them an ideal solution for unreliable and highly-dynamic environments, such as wireless ad-hoc networks, as their operation can remain unaffected by frequent communication disconnections, or, limited bandwidth, which is normally the case for ad-hoc networks.

C.2 Intelligent Agent

According to Franklin, S. and Graesser, A., 1996, autonomous agents can be generally grouped into three categories: biological, robotic, and computational. A software agent is generally a sub-class of an autonomous computational agent, which can be further divided into task-specific, entertainment and virus. Figure C.1 illustrates the taxonomy using a tree-like structure.

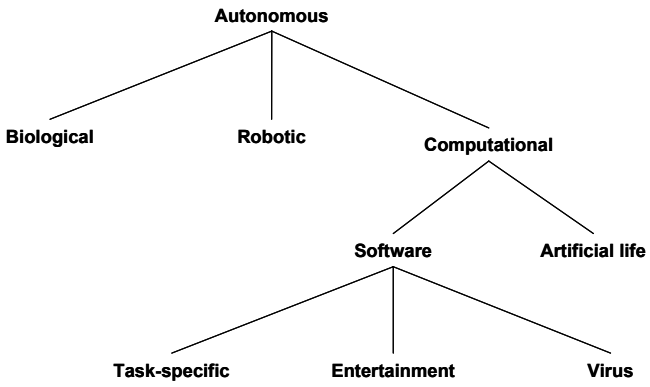

Figure C.1: Generic agent taxonomy

Traditionally, applications in distributed systems have been structured using the client-server paradigm, in which client and server processes communicate either through message passing, or remote procedure calls (RPC) (Tay, B. H. and Ananda, A. L., 1990). This communica-

tions model is normally synchronous, that is, the client blocks after sending a request to a server, waiting for the results of the call. A number of alternative communication models have been proposed and implemented in order to improve to traditional RPC for distributed programming, including: process migration (Powell, M. L. and Miller, B. P., 1983); remote evaluation (Stamos, J. W. and Gifford, D. K., 1990); and mobile objects (Jul, E., et. al., 1988).

According to process migration, an entire address space can be moved from one host to another. The basic drawback of this model is that it does not allow an easy way to return data back to the originator node, without the entire process returning, as well. According to remote evaluation, a host can send a request in a form of a program to another host on the network. The remote host would then run the program in its own address space, and return the results to the originating host. Remote evaluation improved on process migration, as process control data was not required to be transmitted by the source host to the destination host. The main drawback was the absence of state information in the executable program at the remote host. The mobile object communications model further improves on remote evaluation as it allowed objects to migrate from node-to-node while carrying: program code; data in the form of variables state; and, optionally, other information. The main drawback of mobile objects is that they are only suitable for homogeneous local area networks (LANs).

In contrast to its previous counterparts, the mobile agent paradigm allows the migration of an agent, with code, data, and, possibly, execution context, from node-to-node between heterogeneous networks, in an autonomous way. Thus, the agent decides the resources required and finds its way in a heterogeneous network, such over the Internet. It can then perform calculations and appropriate information filtering, and finally return home, to the host who created the agent, to present the results to the user. Figure C.2 illustrates the evolution of the mobile agent paradigm from its ancestors.

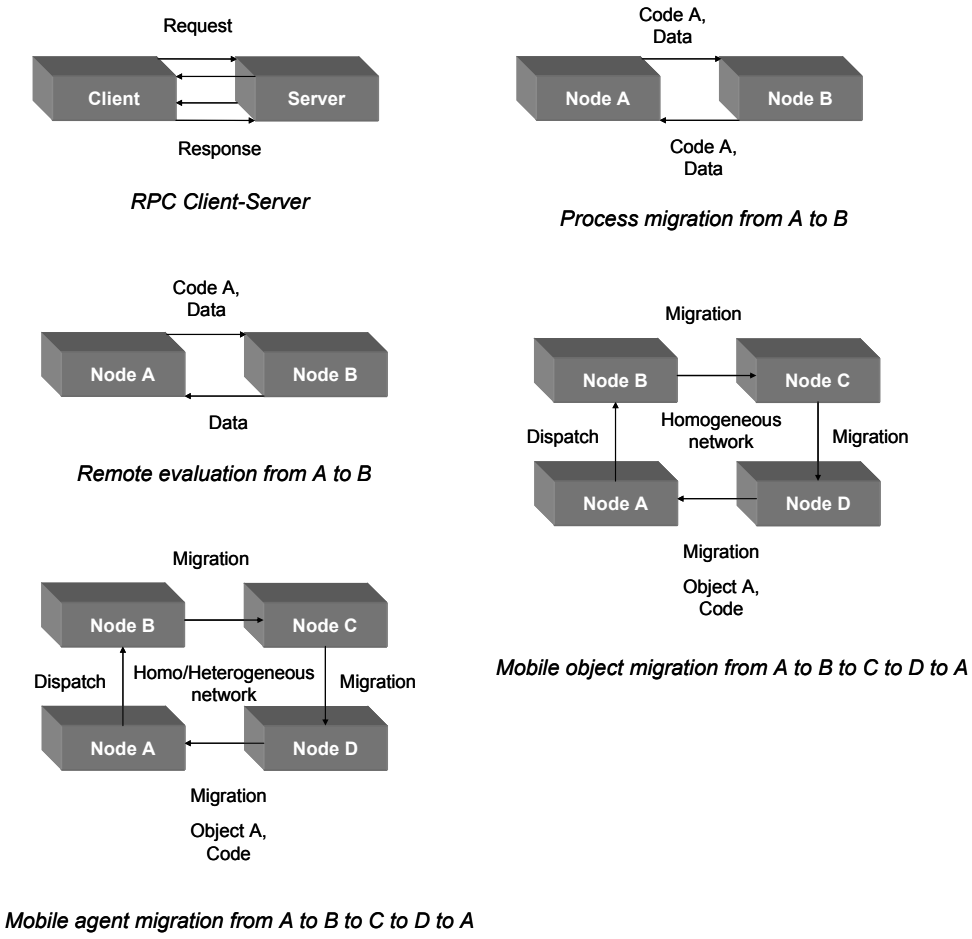

Figure C.2: Evolution of the mobile agent paradigm

C.3 Advantages of Mobile Agents

The main advantages of the mobile agent paradigm lie in its ability to move client code and computation to remote server resources, and in permitting increased asynchrony in client-server interactions (Harrison, C. G., et. al., 1995). By moving the computation close to the needed resources, mobile agents can reduce communications that would otherwise take place over the network, and thus reduce bandwidth and latency requirements. Two principal network features, which motivate the need for mobility, are (Marrow, P. and Ghanea-Hercock, R., 2000):

- Discontinuous network communication or limited bandwidth, such as wireless devices.
- Remote operations or distributed processing.

The mobile agent paradigm may offer a large amount of advantages compared to traditional

client-server approaches, including (Lange, D. B. and Oshima, M. 1999):

- **Reduce network load**. A mobile agent can be dispatched to a remote host and thus avoid multiple interactions of communication protocols that often result in a good deal of network traffic. In addition, instead of transferring large amounts of data over the network, the computation is moved to the remote machine.
- **Overcome network latency**. Control of critical real-time systems by a central controller through a network often involves significant latency. Instead, a mobile agent can be dispatched from the central controller to act locally, and execute the instructions directly.
- **Encapsulate protocols**. A mobile agent can be dispatched to a network carrying an implementation of a protocol in its payload, and thus use this implementation to establish *channels* between remote hosts, or, update remote host's protocols with a newer version.
- **Execute asynchronously and autonomously**. A user can delegate a set of tasks to a mobile agent, dispatch it from their mobile device to a fixed network, and then switch the device off. When the agent finishes its task, it can notify the user by means of an SMS message to their mobile phone, so that the user can switch the mobile device back on, and allow the mobile agent to return home, and present the results. After being dispatched, the mobile agent becomes independent of the process that created it and can operate asynchronously, and autonomously. In this way, the user can save valuable time, money, and also avoid the hassle of intermittent wireless connections.
- **Use parallel processing**. A mobile agent can clone itself, and dispatch copies to other remote hosts, and thus accomplish tasks in parallel.
- **Are robust and fault tolerant**. A mobile agent has the ability to react dynamically to unfavourable situations and events, which makes it easier to build robust and fault-tolerant distributed systems.
- **Adapt dynamically**. A mobile agent can sense changes in the execution environment and react autonomously.
- **Are naturally heterogeneous**. A mobile agent is generally host and transport layer independent, and thus suits network computing environments which are typically heterogeneous.

C.4 Mobile Agents Applications

Mobile agents offer a set of additional features that make them suitable for a number of application areas. In addition, mobile agent technology can be used in combination to traditional approaches, such as with the client-server communications model, to provide an

extensive number of applications. Although, none of the following applications require the use of mobile agents (Harrison, C. G., et. al., 1995), their usage can contribute to a simpler, and, more effective, development of these distributed applications (Puliafito, A., et. al., 2000). Mobile, or wireless computing, is the most frequently proposed application area of mobile agent technology (Kotz, D., et. al., 1997), and two most important features that make mobile agents ideal for dynamic environments are:

- **Task continuation**. An agent can autonomously migrate to a server and continue its processing task, while the user can disconnect from the network.
- **Minimal connection use**. The agent can pre-process information locally, at, either the server, or the mobile device, and thus reduce network overhead which would otherwise be required for large data transmissions.

In addition to these, an extra feature that makes mobile agents ideal for mobile computing is their ability to operate asynchronously (Hadjiefthymiades, S., et. al., 2002). Even if bandwidth is readily available in the near future, battery life on mobile devices is likely to be restricted (Ghanea-Hercock, R., 2001). Also, wireless bandwidth is expected to remain high-priced and thus worth making applications as more efficient as possible.

Information retrieval using mobile agents, suit the requirements of the new dynamic scenarios, such as the one derived from the Internet (Cardi, G., et. al., 2000). This is due to their capability of moving to the place where the information is stored, thus saving bandwidth, and to their robustness in the presence of unreliable connections.

E-commerce has become the focus of information technology development in recent years. For this, mobile agents can search, find, and purchase products and services, in favour of a user (Tianfield, H., 2001). Lee, T. O., et. al., 2001, proposed an agent-based model for processing micropayment transactions in a distributed environment, which is secure, and avoids calculation of complex cryptographic and authentication mechanisms that often require high-processing. Wang, H., 2000, proposes a mobile agent scheme for implementing secure business transactions on the Internet, which improves on security, in comparison to traditional business transaction protocols.

Network and Systems Management (N&SM) using mobile agent technology has been proposed as an answer to the scalability limitations of centralised models, and the flexibility problems of static hierarchical frameworks (Gavalas, D., et. al., 2001). Traditional network management protocols, such as Simple Network Management Protocol (SNMP) (Case, J., et. al., 1990), offer a largely centralised approach to network management, thus posing a challenge to scalability, and often cause network congestion. Baldi, M. and Picco, G. P.,

1998, investigated an alternative approach of network management using mobile agents, and compared the traffic generated by standard SMNP against their approach. They concluded that the mobile agent approach could save up to 95 (%) of network traffic, if, and only if, the mobile agents perform appropriate filtering in visiting nodes. Similarly, Marques, P., et. al., 2001, also came to the same conclusion after conducting research on the usage of mobile agents in network management.

Intrusion detection research has been conducted for nearly 20 years, but still remains in its infancy (Zhang, R., et. al., 2001). Unfortunately, traditional intrusion detection systems pose a number of limitations in terms of configurability, scalability, and efficiency. A number of novel intrusion detection systems have been proposed in the literature aiming to overcome limitations of traditional approaches. Autonomous agents for intrusion detection (AAFID) was the first architecture that proposed the use of autonomous agents for intrusion detection (Spafford, E. H. and Zamboni, D., 2000), whereas Zhang, R., et. al., 2001, proposed a multi-agent based intrusion detection architecture that improves on scalability, where intrusion detection is performed in a totally distributed manner. A network security system using an analogy of natural world immunology, where each cell is represented by a respective mobile agent, was proposed by Nishiyama, H. and Mizoguchi, F., 2001, of which additional research material can be found in: (Krugel, C., et. al., 2002), (Hwang, K. and Gangadharan, M., 2001), and (Dasgupta, D. and Brian, H., 2001).

Collaborative applications may also benefit from mobile agent technology, where complex tasks can be divided into smaller pieces, and be delegated to mobile agents that can migrate throughout the network to accomplish them. These agents could perform computations, synchronously share results, and, collaboratively, determine changes to future actions (Wong, D., et. al., 1997). These agents could behave in a totally automatic fashion, and thus require no further assistance subsequent to their dispatch (Tianfield, H., 2003). In this way, users could share data, documents, and various network resources effectively, and efficiently.

Phan, T., et al., 2002, introduced a challenging area of research and development, which involved the integration of wireless mobile devices into the global computational Grid (Phan, T., et. al., 2002), and Migas, N., et. al., 2003b, proposed a framework using mobile agents for routing in ad-hoc networks, which may be used in parallel with alternative architectures, such as the global computational Grid. The idea is based on the fact that static agents monitoring the mobile node's available resources could inform the Grid to use a small part of their computational power, when the node is inactive. Ceruti, M. G., 2001, proposed that the mobile agent paradigm could be used to command, and control, communications, in general, and to network centric warfare, in particular. The authors strongly suggested that agents could provide a key technology to achieve enhanced capabilities in future military informa-

tion system.

Furthermore, Buchanan, W. J., et. al., 2005a, propose a novel agent-based framework, which introduces the concept of automatic agent-based forensic investigators. The framework suggests that forensic investigator mobile agents, can migrate to terrorist computer networks, create their own execution environment, scan the host computers for terrorism material, and remaining unnoticed. The agents can then meet-up in trusted agent environments, and, cooperatively, decide on the validity of a terrorism threat. In the case of a positive outcome, the mobile agents can inform the appropriate authority.

C.5 Mobile Agent Systems

A system infrastructure is required to support stationary and mobile agents, which provides the functionality for the agents to move, communicate with each other, and interact with the underlying computer system (Baumann, J., et. al., 1998). Furthermore, this infrastructure must guarantee the privacy and integrity of agents, and the underlying system must prevent malicious agents from attacking other agents, or the execution environment (Baumann, J., et. al., 1998). Such an infrastructure is often called a mobile agent system, or an agent platform. For the purposes of this thesis, when referring to such a system infrastructure, the phrase *mobile agent system* will be used, since it is a more general one. In general, a mobile agent system is responsible for the execution, management, communication, migration, security, naming, persistency, interoperability, and monitoring of mobile agents. Figure C.3 illustrates a number of computer nodes within a network, all running a mobile agent system, where a single mobile agent visits each of the nodes, in a sequential order.

Nowadays, there exist many mobile agent systems for commercial and educational use. Some well-known mobile agent systems include: Telescript (White, J., 1997, White, J., 1996); D'Agents (Gray, R. S., 1998, Gray, R. S., 1995); Mole (Baumann, J., et. al., 1998); Aglets Workbench (IBM, Inc., 1997); Concordia (Wong, D., et. al., 1997, Mitsubishi, Electric, 1997); Voyager (ObjectSpace, Inc, 1997); Grasshopper (IKV++, Inc., 2003); and Ajanta (Karnik, N. M. and Tripathi, A. R., 2001, Tripathi, A. R., 1998).

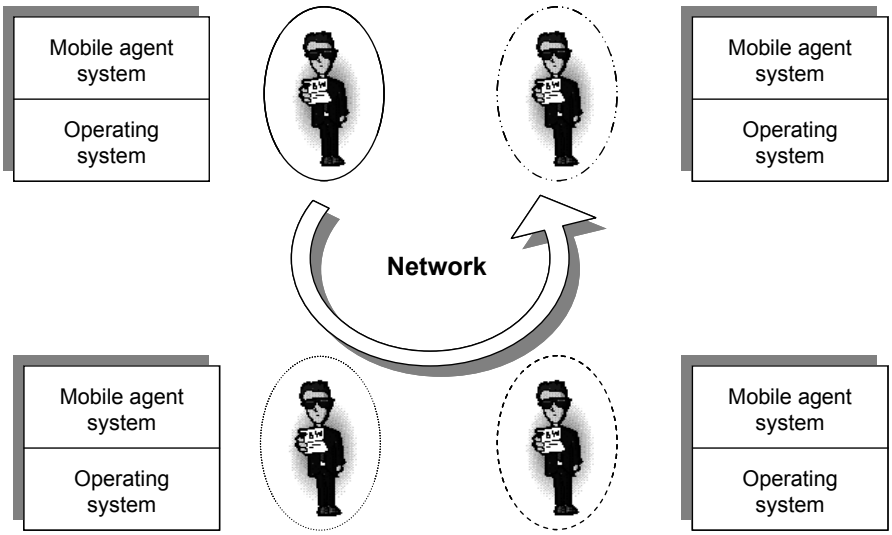

Figure C.3: Mobile agent migration among four agent-enabled hosts

Although these systems have similar features, and functionality, they contain important technical and even conceptual differences (Silva, A. R., et. al., 2001). The first commercial agent system was Telescript by General Magic, which developed their own language and a development environment for agents. However, Telescript failed because it was huge, and unstable, had poor performance, and included a difficult-to-learn programming environment (Silva, A. R., et. al., 2001). The widespread use of the Java programming language (Sun, Microsystems, 2003b), and its platform-independent features, favoured the design and creation of several mobile agent systems such as Aglets, Grasshopper, Mole, Ajanta, Concordia, Voyager, while they downgraded the usage of other languages for the same reason.

D'Agents (Gray, R. S., 1998), formerly called Agent-Tcl (Gray, R. S., 1995), is a mobile agent system developed at Dartmouth College in the USA. Although it was based on a Tcl interpreter, D'Agents was designed to be independent of virtual machines, and of their respective languages (Silva, A. R., et. al., 2001). Currently, D'Agents support Tcl, Scheme, Java, and C/C++. When an agent wants to migrate to a new machine, it calls a single function, which automatically captures the complete agent state and sends this state information to the server on the destination host. The destination host starts up an appropriate execution environment, such as a Tcl interpreter for an agent written in Tcl, and loads the state information into its execution environment, and restarts the agent from the exact point at which it was left off (Gray, R. S., 1998). Thus, D'Agents provides true independence between the mobile agent system, and the programming language the agents have been written into. Also, D'Agents supports strong migration, that is, it preserves agent code, data state, and execution state (*see* Section C.6). According to Gray, 1998, the security architecture is prone to denial-

of-service attacks, and may thus be considered incomplete.

Mole (Baumann, J., et. al., 1998) is the first mobile agent system that was developed in the Java language (Gosling, J., et. al., 2000). Weak migration is used for the transportation of mobile agents, as the Java language does not provide any mechanisms to capture execution state information. Mole provides an execution environment for agents, as all other mobile agent systems, which is mainly based on the concept of agents, and places. An agent system consists of a number of places, being the home of various services (Baumann, J., et. al., 1998). Agents are then active entities, which may move from place-to-place to meet other agents, and access services in these places (Baumann, J., et. al., 1998). Mole provides a naming service that uniquely identifies mobile agents, and, is location independent, that is, it does not change when the agents moves to a new place.

Aglets (IBM, Inc., 1997) is a Java-based mobile agent system developed by IBM Tokyo Research Laboratory, and names mobile agents as aglets, which are Java *Objects* that can suddenly halt their execution, be dispatched to a remote host, and resume execution to the new host (Green, S., et. al., 1997). Aglets support weak migration, that is, agent code and data state (*see* Section C.6). The aglets mobile agent system contains the following components: the Java Aglet API, the mobile agent system, and the Fiji. The Java Aglet API provides a set of classes, and interfaces, which facilitate the implementation of agents and agent-based applications. The mobile agent system provides the execution and computational environment for aglets, and Fiji allows for the creation of applets which support the aglets existence. It also offers the following: naming service; persistence service; navigation; communication; access to external resources; and security (Silva, A. R., et. al., 2001). Unfortunately, according to Tripathi, 1998, Aglets have a limited security support.

Concordia (Wong, D., et. al., 1997) is another Java-based mobile agent system, which was developed by Mitsubishi Electric (Mitsubishi, Electric, 1997). Like all mobile agent systems developed in Java language, it provides weak migration (*see* Section C.6). It has extensive support for agent communication, and also provides asynchronous event signalling, as well as a specialised group collaboration mechanism (Tripathi, A. R., 1998). It also addresses fault tolerance requirements with an object persistence mechanism that is used for reliable agent transfer, and can be used by agents, or servers, to create checkpoints for recovery purposes (Tripathi, A. R., 1998). Concordia implements security functions, such as access control to local resources and certified mobile agents. Each mobile agent is associated with a particular user, and carries a one-way hash of that user's password, however, it only applies to closed systems.

Voyager is a yet another Java-based mobile agent system developed by ObjectSpace (ObjectSpace, Inc, 1997), which is novel in providing location-independent access to an instance

of a class. A Java class, such as a mobile agent, is then transformed to a remotely-accessible equivalent, called a virtual class. Thus, a mobile agent can locate another remote mobile agent, in a similar way, as they both were on the same host. Voyager supports weak migration, that is, agent code and data state (*see* Section C.6). Agent communication is possible via method invocation of virtual references (Tripathi, A. R., 1998). Agents can also make synchronous, one-way, or future-reply-type invocations.

Grasshopper is a Java-based mobile agent system developed by IKV++ (IKV++, Inc., 2003). It was developed to be compliant with the first mobile agent standard of the object management group (OMG), the mobile agent system interoperability facility (MASIF) (*see* Section C.7). The MASIF standard has been initiated in order to achieve interoperability between mobile agent systems of different manufacturers. Grasshopper provides the following services to mobile agents: communication; registration; management; transport; security; and persistence. It supports multiple communication protocols, such as remote method invocation (RMI), secure socket layer (SSL), plain socket, plain socket/SSL, and Internet inter-ORB protocol (IIOP). Supported communication modes include: synchronous; asynchronous; dynamic; and multicast communications. Interestingly, at least from the perspective of this thesis, Grasshopper provides a version for resource-constrained devices, such as Personal Digital Assistants (PDAs), which are Java 2 micro edition (J2ME)-enabled (*see* Appendix A).

Ajanta (Karnik, N. M. and Tripathi, A. R., 2001, Tripathi, A. R., 1998) is a Java-based mobile agent system developed at Minnesota University, and provides a mobile agent infrastructure which supports basic mobile agent features, that is, agent hosting and execution, agent migration, and a binding to the host's environment and resources. Special attention has been focused on security-related features, as well as mechanisms that facilitate the programmer's task of creating robust, agent-based applications. Security features and mechanisms in Ajanta include: authentication for client-server interactions; class loading; thread grouping to implement protection domains for agents; a secure protocol for the transfer of an agent from one host to another; a secure binding mechanism which allows agents to access host resources in a controlled fashion; agent monitoring; and control mechanisms. Furthermore, Ajanta provides mechanisms to protect the agent's state against attacks, originating from the mobile agent system by means of a read-only and append-only container, and a selective encryption of the agent's elements.

C.6 Agent mobility

Agent mobility can be grouped into two categories: strong and weak mobility. The highest degree of mobility is *strong mobility* (Baumann, J., et. al., 1998), which allows a mobile agent

system to capture the entire agent state, that is, data and execution state, before agent migration. In other words, the migrated agent can resume execution at exactly the same point of its code just after the migration command. Although this approach is the most attractive one from the programmer's perspective, it can be inefficient, expensive, and time-consuming from the network's perspective, as the complete agent state can be large. Furthermore, only a few languages allow externalisation of state at such a high level, such as Tcl (Tcl, Developer, 2003). When strong migration is supported; capturing, transfer and restoration of the complete agent is done transparently by the underlying mobile agent system (Baumann, J., et. al., 1998). Unfortunately, in the case of Java language (Gosling, J., et. al., 2000), which is the favourable implementation language for most existing mobile agent systems, there are no such mechanisms that would allow the capturing of execution state information. This can only be achieved by modifying the Java Virtual Machine (JVM) (Lindholm, T. and Yellin, F., 1999), which would cause compatibility concerns. Thus, if a mobile agent system is developed in Java, it is most likely that it will only support weak migration instead of strong migration.

A novel approach for capturing and re-establishing the state of mobile agents is presented by Funfrocken (Funfrocken, S., 1998), where the entire state of an agent is achieved on the language-level, without modifying the JVM, by instrumenting the programmer's original code with a pre-processor. The automatically inserted code saves the runtime information whenever the agent requests state saving and re-establishes the agent's runtime state on re-start.

On the contrary, *weak migration* allows a mobile agent system to only capture the data state of a mobile agent, before dispatching the agent to a new location. This method significantly reduces the amount of agent's state, which is required to be transferred to another location. The size of the transferred state information can be further reduced by allowing the programmer to select the variables making up the *agent state* (Baumann, J., et. al., 1998). In the case of Grasshopper (IKV++, Inc., 2003), any variable, or method, declared as *transient* is not captured before the agent's migration. In addition to reduced overhead, strong migration is not a necessity, as it is always possible to provide the same program-functionality by explicitly coding a program specific migration mechanism on top of a non-strong migration system (Funfrocken, S., 1998). Thus, weak migration is the most favourable adopted approach in most recently developed mobile agent systems. However, the programmer needs to take the appropriate steps in the agent's code, on the basis of the enclosed state information, in order to instruct the agent where to start after migration.

C.7 Drawbacks of mobile agents - Interoperability

Interoperability is a crucial requirement in the Internet scenario: mobile agent applications should be capable of interacting with any other application and service, independent of the adopted programming style (Bellavista, P., 2000). Zhang, M., et. al., 2001, stated that most existing mobile agent systems have their own platform-specific service ontology, encoding mechanisms, and communication protocols, and thus implementation of interoperability as an extension, at this stage, could possibly be unfeasible. Some of these systems are: Agent Tcl (Gray, R. S., 1995); Concordia (Mitsubishi, Electric, 1997); and Voyager (ObjectSpace, Inc, 1997). Even though most of the existing mobile agent systems are well designed, their lack of interoperability services downgrades their usefulness. The absence of appropriate counter-measures for interoperability restricts the propagation of mobile agent technology.

OMG/CORBA

Distribution of applications and services over a network solves the problems of developing and managing huge, centralised applications. However, it imposes several challenges on application development. Autonomously-implemented application components tend to be heterogeneous, as a result of being implemented in different programming languages, and are targeted at different hardware and operating system (OS) platforms (Emmerich, W., 1997). In order to support distributed applications in globally distributed systems, appropriate middleware layers aim to reduce the problems of distribution transparent to developers and users[1]. CORBA is one of the most widely-used middleware in distributed environments, providing a distributed programming environment (DPE), according to which distributed objects can transparently interact based on the client-server model (Bellavista, P., et. al., 2001). It also hides the implementation and location of server objects from client objects, and thus provides transparent communication, and interactions. Furthermore, CORBA allows the integration of already-implemented software components by simply wrapping around an interface definition language (IDL) that describes their behaviour. CORBA and mobile agents can successfully complement each other, despite their differences. A major difference is that a mobile agent is a location-aware entity, whereas, CORBA hides the actual

[1] Examples of such layers include the distributed computing environment (DCE) of the Open Software Foundation, ISO's ODP standard, various standards of the CCITT (such as X.400, X.500, and X.722), the Common Object Request Broker Architecture (CORBA) of the Object Management Group (OMB), and the evolving Distributed Component Object Model (formerly called Network OLE) from Microsoft (Emmerich, W., 1997).

location of objects. In addition, CORBA has reached a wide acceptance, while the mobile agent paradigm has led to a great variety of different and non-interoperable mobile agent systems (Bellavista, P., et. al., 2001). Thus, CORBA integration is vital for emerging mobile agent systems.

OMG MASIF

The OMG group works in different specialised areas. One sub-group has defined the mobile agent system interoperability facilities (MASIF) standards (OMG, MASIF, 1997). MASIF is an agent interoperability standard, built within the CORBA framework, mainly to support agent tracking, mobility, and management (OMG, MASIF, 1997). As a result, it provides no support for communication. MASIF allows interoperability between mobile agent systems written in the same language, but, potentially, by different vendors and systems that are expected to go through many revisions within their lifetime. Language interoperability is difficult to achieve, and, furthermore, unnecessary, as the support for different languages can be replicated at each node. MASIF thus does not impose any requirements for rebuilding an already existing mobile agent system, but instead it requires the development of an *add-on* module which conforms to the MASIF specification. The module must then get plugged into the existing mobile agent system. MASIF does not deal with standardisation of local agent operations such as *interpretation*, *serialisation*, *execution*, and *deserialisation*, as these actions are application-specific, and there is no reason to limit mobile agent systems implementation (Bellavista, P., et. al., 2001).

MASIF proposes standardisation for agent and agent system names, for agent system types, and for local syntax. Two interfaces are proposed in the specification document: MAFAgentSystem and MAFFinder. The former provides operations for the management and transfer of agents, whereas the latter interface supports the localisation of agents and mobile agent systems in the scope of an administered locality (Bellavista, P., et. al., 2001). MASIF allows communication with a mobile agent system in a MASIF-compliant way, which allows interoperability, or in a platform-specific way, which may provide additional functionality.

FIPA

Foundation for Intelligent Physical Agents (FIPA) (FIPA, 1997) is an abstract architecture that can be shared by different platform implementations, and agents of different systems or providers, as far as they are all FIPA-compliant. Accordingly, agents can communicate and interact directly by Agent Communication Language (ACL) (Zhang, M., et. al., 2001). The main emphasis of FIPA specification is concentrated on the standardisation of agent com-

munication and proposes an ACL in order to support interoperable communications between heterogeneous FIPA-compliant agents. FIPA proposes three basic services (Bellavista, P., et. al., 2001):

- **Agent Management System** (**AMS**). This specification provides the normative framework within which FIPA agents exist and operate. Its purpose is the establishment of the logical reference model for the creation, registration, location, communication, and migration of agents.
- **Directory Facilitator** (**DF**). This service may be considered as the yellow pages directory. Agents that want to offer their services to other agents may register to a DF, and thus allow other agents to request their services. Agent registration to a DF is optional, while registration to an AMS is mandatory.
- **Agent Communication Channel** (**ACC**). This service allows communication between agents of possibly heterogeneous mobile agent systems, using a message forwarding service. This service requires ORB CORBA integration, which is considered mandatory for any FIPA-compliant mobile agent system. Agent messages are transferred by CORBA Internet Inter-Orb Protocol (IIOP).

Local communications between agents may be released by the mobile agent system's native communication protocol. However, any mobile agent system that is FIPA-compliant needs to implement an ACC in order to forward ACL messages between heterogeneous agents. FIPA-based communication between heterogeneous mobile agent systems requires an implementation via a message forwarding service over CORBA.

C.8 Drawbacks of the mobile agents - Security weaknesses

The fundamental security requirements for any computer system are confidentiality, integrity, and availability (Pfleeger, C. P., 1997). In order to ensure confidentiality, a computer system must prevent unauthorised disclosure of information, while to ensure integrity, it must prevent unauthorised modification of information. In order to ensure availability of a computer system the prevention of unauthorised withholding of information is required.

Although the mobile agent paradigm extends the capabilities of traditional methods of remote communication and distributed computing, it also raises new security issues (Chess, D. M., 1998). Compared to the client-server model, mobility of code increases the threat of security violations (Corradi, A., et. al., 2001). The reason for this lies largely on one intrinsic characteristic of mobile agents: execution on remote unknown mobile agent systems rather

than their safe home mobile agent system (the one a mobile agent originates from). This is especially true in large heterogeneous and open computing environments, like the Internet (Zhang, M., et. al., 2001). Figure C.4 illustrates a generic threat model of this new technology.

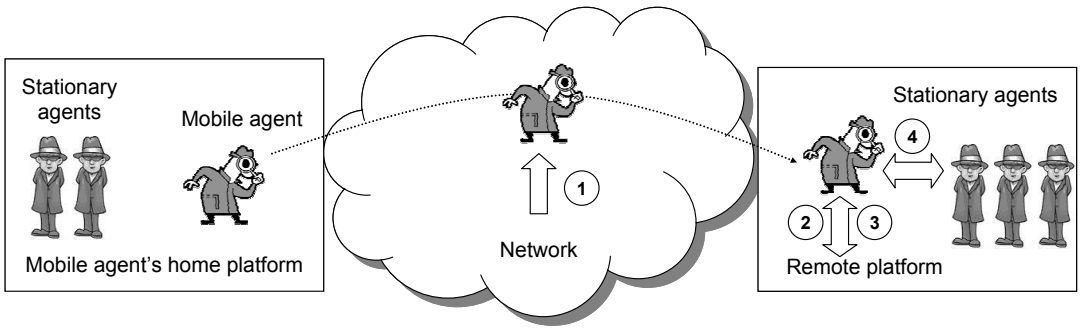

Figure C.4: A threat model of a mobile agent and a mobile agent system

Security threats can be grouped in the following categories (Jansen, W., 2000):

- **Agent against mobile agent system**. An incoming mobile agent has two main lines of attack. Firstly, it can gain unauthorised access to information residing at the agent platform, and secondly it can use its unauthorised access in an unexpected and disruptive fashion (Jansen, W., 2000). Attacks can be grouped into two categories: passive and active (Zhang, M., et. al., 2001). Passive attacks include communications monitoring and sensitive information pilfering, while active ones include the damage of the host's resources via deletion or modification. The problem of host protection against malicious agents has already been extensively investigated (Corradi, A., et. al., 2001).
- **Mobile agent system against agent**. During the execution of a mobile agent, the agent is in a very asymmetric relationship with regards to the server, since the server must be able to access the agent's code, data, and state, in order to execute it (Kotzanikolaou, P., et. al., 2000). A receiving mobile agent system can easily isolate and capture an agent and may attack it by extracting information, corrupting or modifying its code and state, denying requested services, or simply by reinitialising or terminating it completely (Jansen, W., 2000).
- **Agent against other agents**. An agent can target another agent using several approaches, including actions to falsify transactions, eavesdrop upon conversations, or interfere with an agent's activity (Jansen, W., 2000). In addition, an agent can respond incorrectly to direct requests sent by another agent or simply deny that a legitimate transaction oc-

curred. Furthermore, agents may exploit security weaknesses of other agents or launch attacks by repeatedly sending messages in an attempt to deny them the ability to communicate.

- **Other entities against mobile agent system**. Even when assuming that the locally active agents and the mobile agent system are well behaved, other entities both outside and inside the agent framework may attempt actions to disrupt, harm, or subvert the mobile agent system (Jansen, W., 2000). For instance, a mobile agent is at risk from the outside network when it is migrating or communicating with its home site (Zhang, M., et. al., 2001). Typical attacks include eavesdropping, traffic analysis, tampering, and forging.

C.8.1 Security countermeasures for mobile agents

Countermeasures refer to any action, device, procedure, technique, or any other measure that can potentially reduce the vulnerability of, or, the threat to a system (Jansen, W., 2000). Most agent systems rely on a common set of baseline assumptions regarding security:

- The home mobile agent system of a mobile agent is always trusted.
- The home mobile agent system and other equally trusted mobile agent systems are implemented securely, with no flaws or trapdoors.

To address the security issue, public key cryptography (Mohapatra, P. K., 2000, Buchanan, W. J., 2000), primarily in the form of digital signature, is utilised through certificates and revocation lists managed through a public key infrastructure. The following sections, briefly present a number of countermeasures aiming to protect a mobile agent and a mobile agent system.

Protection of the mobile agent system

A famous countermeasure to protect a mobile agent system against mobile agent attacks and also avoid interference between agents is the usage of a *reference monitor* (Wolthusen, S. D., 2002). A reference monitor can enforce separate isolated domains for each agent and the mobile agent system, and also control all inter-domain access. An implementation of a reference monitor applicable in the mobile agent framework may employ a number of security techniques (Jansen, W., 2000):

- Mechanisms to isolate processes from one another, and from the control process.
- Mechanisms to control access to computational resources.
- Cryptographic methods to encipher information exchanges.

- Cryptographic methods to identify and authenticate users and mobile agent systems.
- Mechanisms to audit security relevant events occurring at the mobile agent system.

Several techniques have been developed with purpose to provide protection for the mobile agent system, including:

- **Software-based fault isolation** (Wahbe, R., 1994). This is the method of isolating application modules into distinct fault domains.
- **Safe code interpretation** (Gong, L., 1998). This is a technique that denies execution of potentially harmful instructions in the agent's code.
- **Signed code** (Mohapatra, P. K., 2000). Signing of agents by the use of public-key cryptography (Mohapatra, P. K., 2000, Buchanan, W. J., 2000) provides a means of confirming the authenticity, its origin, and its integrity.
- **State appraisal** (Farmer, W. M., et. al., 1996). This method may be applied to an agent in order to detect if the agent has been subverted due to alterations in its state information.
- **Path histories** (Chess, D., et. al., 1995). This mechanism maintains an authenticable record of the prior platforms visited by an agent, so that a newly visited mobile agent system can determine whether to process the agent or not and what resource constraints to apply.
- **Proof-carrying code** (Necula, G. and Lee, P., 1996). It obligates the code producer to formally prove that the agent possesses safety properties previously stipulated by the code consumer.

Protection of the mobile agent

Security countermeasures on this class of attacks represent a new and challenging area of research (Corradi, A., et. al., 2001). The techniques to protect a mobile agent from host attacks can be grouped into two categories: prevention of agent tampering and detection. Prevention methods aim to stop an attack from actually succeeding. On the other hand, detection methods aim to detect agent tampering, after an attack has taken place, trace the identity of the illegitimate host, and prove its misbehaviour. These methods may provide partial solutions to particular problems, and thus are not sufficient (Kotzanikolaou, P., et. al., 2000). Well-known detection mechanisms include the following:

- **Partial result encapsulation** (Jansen, W., 2000). An agent encapsulates partial results of the actions taken place, at each platform visited, for verification to a trusted host such as

the home of the mobile agent.

- **Mutual itinerary recording** (Roth, V., 1998). It allows an agent's itinerary to be recorded and tracked by another cooperating agent and vice-versa, in a mutually supportive arrangement.
- **Itinerary recording with replication and voting** (Schneider, F. B., 1997). This approach suggests that multiple copies of an agent may be used to perform a task, thus even if a malicious mobile agent system destroys some of the copies, enough replicas will remain to accomplish the task.
- **Execution tracing** (Vigna, G., 1997, Vigna, G., 1999). Execution tracing is a technique for detecting unauthorised modification of an agent, through the faithful recording of the agent's behaviour, during its execution on each mobile agent system.

Prevention methods can be grouped into two categories:

- **Passive prevention mechanisms**. These protect the agents by using organisational or architectural solutions. This approach either makes strong arguments on the trustworthiness of a host or compromises many of the advantages of mobile agents, such as autonomy (Kotzanikolaou, P., et. al., 2000).
- **Active mechanisms**. These try to provide solutions without making any hard assumptions or compromising the advantages of mobile agent technology.

Some of the most well-known prevention mechanisms include the following:

- **Environmental key generator** (Riordan, J. and Schneider, B., 1998). It defines a scheme for allowing an agent to take predefined action when some environmental condition is true.
- **Computing with encrypted functions** (Sander, T. and Tchudin, C. F., 1998). This aims to determine a method whereby mobile code can safely compute cryptographic primitives, such as a digital signature, even though the code is executed in untrusted computing environments, and operates autonomously without interaction with the home mobile agent system.
- **Obfuscated code** (Hohl, F., 1998). This aims to scramble the code of an agent in such a way that no one can completely understand its function, or to modify the resulting code without detection.

D Appendix - Ad-hoc networks

D.1 Problems and challenges of wireless networks

Wireless networks are fundamentally different from conventional stationary, wired computer networks (Elaarag, H., 2002). Despite the great benefit of users to access information *any-time-and-anywhere*, wireless networks have certain inherent problems. For instance, the Quality of Service (QoS) is dramatically reduced compared to their wired counterpart. A number of these problems and key challenges are (Elaarag, H., 2002, Zorzi, M., 1998, Qi, H. and Wang, F., 2001):

- **Channel unreliability**. This is the most tarnished characteristic of wireless communications, due to a number of physical factors, such as signal propagation through such channels being subject to severe impairments.
- **High bit error rates**. In some situations, considerable number of data packets and acknowledgements may be lost.
- **Disconnections**. These may happen due to a number of reasons. For instance, when a mobile device moves from one access point to another, the new access point takes over - this is called handoff. During hand-offs, there is a brief disconnection period.
- **Limited and variable bandwidth**. The available bandwidth is often not large, as the radio spectrum, itself, is an inherently public resource, and is already crowded due to the presence of other services, such as broadcast TV, military communications, and point-to-point radio links. Thus, wireless systems are always limited by interference, which often dictates the amount of bandwidth available.
- **Dynamic network topology**. Movement of mobile devices causes rapid changes in the topology of the network.
- **Fixed routing is impossible**. This is the devices being mobile, as fixed routing becomes impossible, and new routing strategies need to be adopted.

D.2 A general model for Ad-hoc networks

Wireless networks, using the IEEE 802.11 standard, allow greater flexibility, and mobility, and can create ad-hoc networks. The wireless transmission range of IEEE 802.11 has a certain propagation limit, and beyond this point, a wireless device is considered not to be in the

direct communication range of other devices. Thus, in an ad-hoc network, it may be necessary for a mobile device to seek the aid of others in forwarding data packets to their destination, due to the limited propagation range of each mobile device's wireless transmissions (Hassanein, H. and Zhou, A., 2001). In case that all wireless devices are in direct communication range of each other, there is no need for routing, and the ad-hoc network is, by definition, fully-connected. However, this is rare in practice, since wireless devices may be spread over a large geographical area. The electrical power required to obtain full connectivity, when mobile devices are spread over a large geographical region, may be impractical, wasteful of important battery life, and too vulnerable to security threats. Thus, routing is considered as the most essential, however, it is a challenging issue in the field of ad-hoc networking.

There is a set of basic assumption, often taken for granted, in the context of ad-hoc networking. According to Perkins, this set includes the following (Perkins, C. E., 2001):

- The nodes are far enough apart so that not all of them are within range of each other.
- The nodes may be mobile so that two nodes within range at one point, in time, may be out of range moments later.
- The nodes are able to assist each other in the process of delivering packets of data.

As a simplified example of an ad-hoc network, Figure D.1 illustrates a collection of eight mobile nodes belonging to a wireless network ($WLAN_1$), and a collection of eight more mobile nodes belonging to another wireless network ($WLAN_2$). The double arrows represent the links between mobile devices. The absence of double arrows between mobile devices denotes that these devices are not in direct communication range. Initially, MH_1, MH_2, MH_3, …,MH_8 belong to $WLAN_1$, and MH_9, MH_{10}, MH_{11}, …,MH_{16} belong to $WLAN_2$. Devices from $WLAN_1$ cannot communicate with devices from $WLAN_2$, because there is no intermediate device to route traffic. The nodes are able to move relative to each other and, as that happens, the links between them are broken and other links are established. As Figure D.1 shows, MH_3 moves away from MH_2 and establishes new links with MH_8 and MH_{16}. In this way, MH_3 joins $WLAN_1$ to $WLAN_2$ and thus MH_3 can be used as a router to forward network traffic, originating from $WLAN_1$ to $WLAN_2$, and vice-versa. Thus, if mobile device MH_6 wants to communicate with MH_{15}, it may send network traffic along a number of routes. These routes include MH_6 MH_7 MH_3 MH_{16} MH_{15}, or MH_6 MH_5 MH_8 MH_7 MH_3 MH_{16} MH_{15}. Most routing techniques identify the shortest path as the path with the fewest hops from source node to destination node. Thus, in this example, MH_6 would use the first route as this route is the shortest, in terms of hops.

In addition, in such an environment, if a mobile device has a connection to the Internet, other devices may send network traffic through it, so that the Internet-connected device forwards it out to the Internet. For instance, in Figure D.2, MH_{14} has a connection to the Internet and MH_4 wants to send an e-mail to a recipient somewhere on the Internet. The e-mail may be routed from $WLAN_1$ through node MH_3 to $WLAN_2$ and then out onto the Internet.

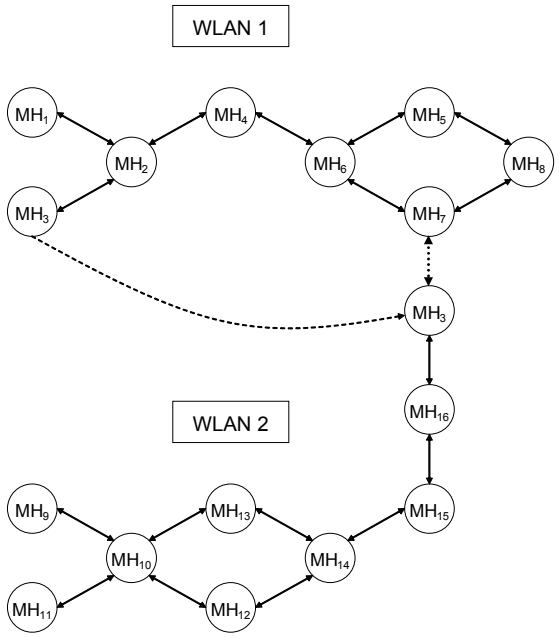

Figure D.1: An example of an ad-hoc network topology with partial mobility

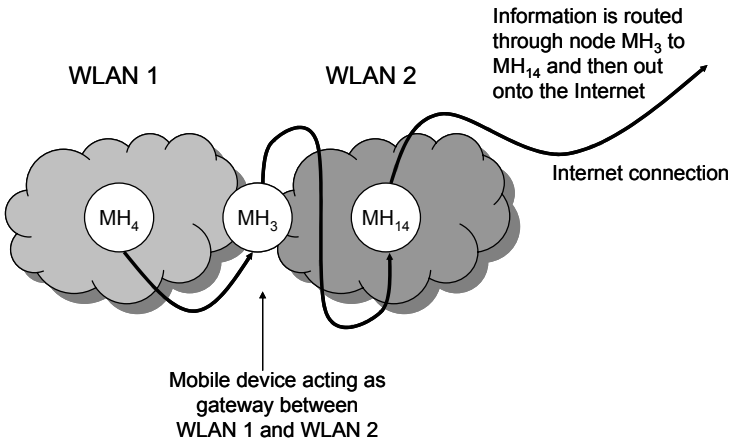

Figure D.2: An example of an ad-hoc network topology with partial mobility

D.3 Ad-hoc network applications

Ad-hoc networks have been proposed as a networking solution where the network setup time is a major constraint, and/or where a network infrastructure is either not available, or nor desirable (Ramarathinam, V. and Labrador, M. A., 2002). The initial motivation for ad-hoc networks was based on military applications, and while military applications still dominate a great part of research in this field, the recent rapid development of mobile communications brought a number of commercial applications of ad-hoc networks (Migas, N., et. al., 2003a). Perkins describes some of the potential application for ad-hoc networks that might provide the basis for commercially successful products (Perkins, C. E., 2001). Some of these include conferencing, disaster relief, health care systems, personal area networks (PANs) and Bluetooth (Appendix A), embedded computing applications, sensor dust, and inter-vehicle communications.

Conferencing is a typical application, where the establishment of an ad-hoc network is necessary. In a conference-meeting, participants may want to exchange information in a form of a document, presentation file, or database file, and so on, without using a fixed network infrastructure, as it may not be available, or desirable. In addition, ad-hoc networks are especially attractive to disaster relief scenarios, where an existing infrastructure is damaged, or out-of-service. In emergency situations, such as in an earthquake, ad-hoc networks may save many human lives, as emergency services can still remain in touch and exchange necessary information, by the use of ad-hoc equipment.

A PAN is usually considered as a highly-localised network, consisting of a number of network nodes that are closely associated with a single person. For example, these nodes could be attached to a person's clothes, or carried in a bag. As these devices are associated with a particular person's activities, they will most probably need to communicate, and even further, to be attached to the Internet. Mobility becomes crucial when interaction between several PANs, or different people, is needed, as users do not stay in fixed locations with respect to each other for long. Methods for establishing communications between nodes on separate PANs could benefit from ad-hoc networks. Furthermore, ad-hoc networks could prove very helpful in health-care system. In hospitals, for example, busy doctors and nurses may want to rely on an administrative infrastructure, at some times, and to utilise direct links, outside the infrastructure, at some other times. Tasks, such as retrieving a patient's records, can be achieved without interaction with the infrastructure, and, in certain cases, these tasks may be accomplished more effectively, and efficiently, by allowing hospital personnel to carry ad-hoc equipment with them at all times.

Another interesting possible application of ad-hoc networks is inter-vehicle communica-

tions, such as FleetNet (Franz, W., et. al., 2001), which deals with inter-vehicle communications, where vehicles could dynamically create ad-hoc networks based on the IEEE 802.11 standard. The key features include (FleetNet, 2003):

- **Cooperative driver assistance**. This aims to provide the driver with emergency notifications, obstacle warnings on roads and overtaking assistance, which are designed to make a journey safer.
- **Decentralised journey data**. This aims to provide information for traffic jam, road problems, dynamic navigation, and even route weather forecasts.
- **User communications and information services**. This aims to provide Internet access, mobile advertising, distributed games, and, even, inter-vehicle chat making journeys more enjoyable for passengers.

Overall, the world is full of intelligent machines that can be mobile and able to process information about the environment in which they operate. Even though ad-hoc networks may not be a necessity, this technology is likely to provide more flexibility and convenient embedded-computing applications.

D.4 Ad-hoc routing protocols

Ad-hoc routing protocols can be categorised by the way they maintain routing information. For example, routing protocols which maintain routing information for each node on the network, at all times, can be categorised as *proactive*, while protocols which discover routing information only when it is required, can be categorised as *reactive*. A third category is *hybrid*, which shares common characteristics with *proactive* and *reactive*. A classification of routing protocols is illustrated in Figure D.3.

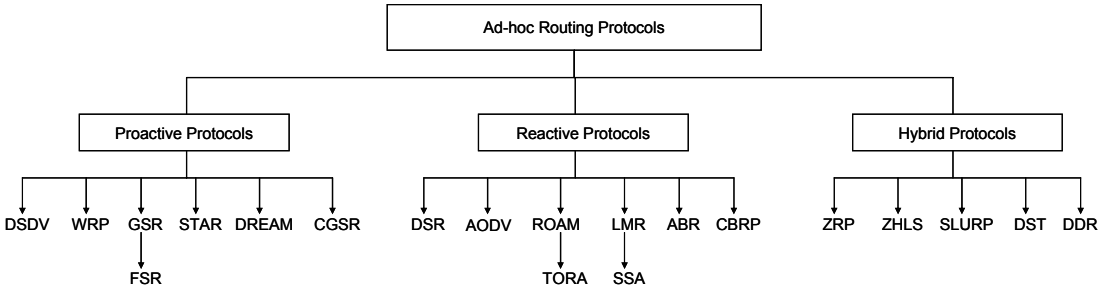

Figure D.3: Classification of ad-hoc routing protocols (Royer, E. M. and Toh C. K., 1999)

D.5 Proactive (table-driven) ad-hoc routing protocols

With proactive ad-hoc routing, the routes to all the destinations are found at the start-up, and, maintained by periodically broadcasting route updates. Thus, each node maintains a route to all other nodes within the network, including those to which no packets are sent. Nodes also react to dynamic topology changes, even if these changes have no effect on the traffic. Routing information is thus maintained at each node and stored in a number of tables. The main disadvantage of this method is that each device is required to store large routing tables in its memory, and they often produce high network overhead by periodically exchanging frequent routing updates. These updates are typically appended into a message, commonly known as *HELLO* message, which is periodically broadcasted by each node in the network. However, in reactive routing protocols, as there is no requirement for routing information exchange, the *HELLO* message contains minimum information, mainly for connectivity verification purposes, which are typically known as beacons. Thus, proactive routing protocols introduce scalability problems, as large amounts of data are often broadcasted, and thus prohibit their use by resource-constrained handhelds. An extensive review of proactive routing protocols can be found in (Abolhasan, M., et. al., 2004), but for the scope of this thesis, only the main features of each protocol are presented here.

The Highly Dynamic-Sequenced Distance-Vector (DSDV) is a modification of the DBF algorithm (*see* Appendix B.3), which unlike DBF, it guarantees loop-free routes (Perkins, C. and Bhagwat, P., 1994). However, scalability problems have not been addressed properly, whereas Wireless Routing Protocol (WRP) improves on scalability, and also guarantees loop-free routes (Murthy, S. and Garcia-Luna-Aceves, J. J., 1995). However, it requires from each node to maintain large routing tables in their memory, and frequently broadcast these, which results in significant memory usage, especially for resource-constrained devices, and further consumes significant bandwidth and electrical power.

The Global State Routing (GSR) enhances the traditional link-state algorithm (see Appendix B.3) by reducing the propagation of dissemination updates to neighbouring nodes only (Chen, T.-W. and Gerla, M., 1998). Even though the propagation of these updates is restricted, the size is still relatively large, and thus imposes scalability issues for large ad-hoc networks. The Fisheye State Routing (FSR) bases its main functionality in GSR, however, it provides a more scalable solution, as it reduces the size of the update messages by increasing the frequency in which the nearby nodes are updated. However, as mobility increases, nodes become less updated for routes to remote nodes, which results in an overall decrease in accuracy (Gerla, M., et. al., 2001). Another protocol which bases its functionality in the link-state algorithm is the Source Tree Adaptive Routing (STAR), where each routing device maintains

a source-tree structure, which encodes the preferred paths to destinations (Garcis-Luna-Aceves, J. J. and Spohn, M., 1999a, Garcis-Luna-Aceves, J. J. and Spohn, M., 1999b). STAR employs two mechanisms: least overhead routing approach (LORA); and optimum routing approach (ORA), where the former significantly reduces the amount of routing information disseminated through the network, while the latter eliminates the requirement for periodic updates, which are commonly found in the link-state algorithm, by allowing updates to be disseminated conditionally. Thus, STAR provides a more scalable solution with reduced latency, however, there are still high-memory and high-processing problems, especially in large, and dynamic networks, mainly because each node is required to constantly maintain a partial view of the topology. In contrast, the Distance Routing Effect Algorithm for Mobility (DREAM) uses GPS, and thus each node knows its geographical coordinates, at any given time. Thus, nodes are only required to exchange location information, instead of complete link-state, or distance vector. In this way, the bandwidth-overhead is even further reduced, which means that it is more scalable (Basagni, S., et. al., 1998).

Unlike the flat-structured routing protocols mentioned above, the Multimedia support in Mobile Wireless Networks (MMWN) organises the network into clusters, where each cluster contains two distinct types of mobile nodes: switches; and endpoints (Ramanathan, R. and Steenstrup, M., 1998). Each cluster has a location manager (LM), which is responsible for managing the location of each node in its cluster. The main advantage is that location-finding and updating is sorely left to LMs, and has been shown to significantly reduce the routing overhead compared to standard link-state and distance-vector approaches. However, the strong association of location-finding with the hierarchical structure makes MMWN hard to cope in the presence of frequent changes in the hierarchy. Cluster-head Gateway Switch Routing (CGSR) is similar to MMWN, as it organises the network into clusters, however, its main advantage is that there is no requirement for cluster-hierarchy maintenance (Chiang, C.-C., et. al., 1997). Although this provides a simpler overall structure, and nodes are only required to maintain routes to their cluster-heads, there are overheads involved in cluster maintenance.

The Hierarchical State Routing (HSR) is based on the link-state algorithm, and maintains a hierarchical addressing and topology map (Pei, G., et. al., 1999). Clustering algorithms can be used in conjunction with HSR in order to organise nodes into clusters and assign key roles to each node within the cluster including: cluster-head, gateway node(s), and member(s). Each node has a hierarchical ID (HID), which is a sequence of the MAC addresses from the top hierarchy to the source node, and can be used to send a packet from any source to any destination in the network. The main advantage of this routing protocol compared to other hierarchical protocols is the separation of mobility management from the physical hier-

archy, while the main disadvantage is the overhead imposed by clustering formation and maintenance. The Optimised Link State Routing (OLSR) is a point-to-point routing protocol, which is also based on the link-state algorithm (Jacquet, P., et. al., 2000, Jacquet, P., et. al., 2001). The novelty of OLSR is that it employs a multipoint replaying (MPR) strategy which alleviates the size of the control messages, and minimises the number of nodes which broadcast at each route update. Routes to every destination are maintained into each node's routing table, which selects the *optimal* route based on the numbers of hops.

The Topology Broadcast Reverse Path Forwarding (TBRPF) is based on the link-state algorithm, and extends it with the concept of reverse-path forwarding (RPF), which is used to distribute the update packets in the reverse direction along the spanning tree (Ogier, R., et. al., 2003). Thus, each node is required to construct a source tree by applying a modified version of the Dijkstra's algorithm (Sedgewick, R., 1983) on the node's partial topology information. Network overhead is minimised by nodes exchanging only parts of their source trees with their neighbours, in a periodic and differential manner, where the latter requires reporting only the changes.

Even though all these protocols base their functionality on the standard proactive methodology, they differ in the way route updates are detected, and disseminated through the network. Thus, they provide different performances in terms of bandwidth-overhead, nodal utilisation, battery consumption, and packet-transmitted-to-packets-received ratio. However, pure proactive methods have been shown to have major scalability problems for large network topologies, and have thus been ruled-out by many recent ad-hoc routing protocol proposals.

D.6 Reactive (on-demand) ad-hoc routing protocols

The on-demand methods aim to reduce the high network overhead imposed by the proactive methods. Network nodes only react when a route is required between a source and a destination node, and there is no need to maintain routes to destinations in which they are not communicating with. Route discovery is normally performed by flooding the network with route-request packets. When the destination, or a node that has a fresh route to the destination, receives such a packet, it reverses the route that the packet took (in case of intermediate node, it also appends the route to the destination) and sends a route-reply along the route. When the route-request propagated through bidirectional, as well as unidirectional links, the route-reply typically contains the route-request piggybacked in the route-reply packet, which is flooded to the network. The reactive method has been proven to be simpler, and more efficient and scalable, in comparison to the proactive method. However, it can introduce high-

latency, as a node is required to initiate the route discovery process each time a data packet needs to be transmitted to a destination for which the node does not have a route.

Reactive routing protocols can be grouped into two categories: hop-by-hop routing; and source routing.

- **Hop-by-hop routing**. Each node in the ad-hoc network maintains a routing table listing the *optimal* next hop for all reachable destinations. Thus, when a node receives a data packet, it determines the optimal next hop for the destination found in the packet's header, and transmits it.
- **Source routing**. Each data packet carries the complete, and ordered list of nodes in its header, which the packet must traverse to arrive at the desired destination. Thus, intermediate nodes do not need to maintain up-to-date routing information, as they can always identify the next hop by examining the packet's header.

According to hop-by-hop routing, when a node receives a data packet destined for node *D*, it consults its routing table, and forwards the data packet to the preferred neighbouring node for destination *D*. This process iterates, with each receiving node forwarding the data packet to the next hop, until the data packet eventually arrives at the destination. In contrast, source-routing requires that each originator node wishing to transmit a data packet supplies the complete route which the data packet must take, as an extension to the IP header, in order for the packet to arrive at the destination. Source routing has the advantage that routing nodes do not need to maintain up-to-date routing information in order to route the packets they forward, as the packets, themselves, contain the complete route, and thus eliminates the network overhead caused by periodic route advertisement in hop-by-hop approaches.

A large number of on-demand routing protocols has been defined in the literature (Abolhasan, M., et. al., 2004). The Dynamic Source Routing (DSR) belongs to the category of source routing, as each data packet is required to carry the full-route address from the source to the destination, including the intermediate hop addresses as an extension to the IP header (Johnson, D. B. and Maltz, D. A., 1996, Johnson D. B., et. al., 2004). This means that the overhead imposed by the source route included in the packet rises proportionally to the number of nodes the packet is required to transverse, and, may, thus, impose a high overhead for large ad-hoc networks. On the other hand, nodes are not required to maintain next hop routing information, and thus the network overhead is significantly reduced by eliminating the need of periodic *HELLO* messages exchange. The Ad-hoc On-demand Distance Vector (AODV) is partially based on DSDV, and partially based on DSR (Perkins, C. E., et. al., 2003, Perkins, C. E. and Royer, E. M., 1999). It borrows the periodic *HELLO* broadcasts,

and the use of sequence numbers from DSDV, while it uses a route discovery process similar to DSR. In contrast to DSR, routing is accomplished by requiring each data packet to carry the source and destination IP only, which considerably reduces network overhead. In addition, route replies carry only the destination IP address and the sequence number of that destination, while routing decisions are left to the intermediate nodes. Even though AODV may be adaptable to highly dynamic networks, high latency may be imposed by dynamic route construction, and link failures may result in initiating new route discoveries, which can introduce additional delays and network overhead.

The Routing On-demand Acyclic Multi-path (ROAM) is a multi-path distance vector algorithm, which uses inter-nodal coordination based on directed acyclic graphs (Raju, J. and Garcia-Luna-Aceves, J. J., 1999). The advantages of this protocol are that it eliminates the search-to-infinity problem, and that routers are only required to maintain routes to destinations for which they actively forward data packets. In addition, ROAM defines a threshold value for each router's distance to a destination, which, once exceeded, causes the router to broadcast update messages to its neighbouring nodes. The main disadvantage of this protocol is that it requires the maintenance of state information at each node during route discovery, and may thus not be suitable for highly dynamic networks. Another on-demand routing protocol is the Light-weight Mobile Routing (LMR), which uses a standard flooding route discovery process (Corson, M. S. and Ephremides, A., 1995). The advantage of LMR is that it allows nodes to maintain multiple routes to a single destination, and thus improves on the protocols overall performance, as nodes can transmit through alternative routes, in case of primary failure, whereas its disadvantage is that route-requests propagate throughout the complete network, which results in significant bandwidth overhead. In addition, this protocol also suffers from temporary invalid routes which can introduce delays. The Temporally Ordered Routing Algorithm (TORA) is based on LMR, and improves it by restricting the propagation of route-request to only neighbouring areas in which topological changes have occurred. Similarly to LMR, TORA suffers from temporary invalid routes.

A novel stability-driven routing protocol is the Associativity-Based Routing (ABR), which unlike standard shortest-path algorithms, it primarily bases its route selection on stability (Toh, C., 1996). With this, each node maintains an associatively tick with each of its neighbours, and routes comprised of higher associatively ticks are considered more optimal than others with lower ones. This often results in routes that generally last for a longer time, and thus route discovery is expected to be invoked less frequently than in other methods, resulting in bandwidth conservation. The disadvantage of this protocol is the lack of support for multiple routes to a single destination, and the periodic *HELLO* message exchanges. Signal Stability Adaptive (SSA) is ABR's successor, which also uses route stability for optimal

route discovery, however, the technique differs from ABR, in that SSA determines route stability by measuring signal strength and location stability (Dube, R., et. al., 1997). Compared to DSR, it has the disadvantages that intermediate nodes cannot reply to route-requests, even if they have a route to the requested destination. There is also lack of support for route repair mechanisms.

The Relative Distance Micro-discovery Ad-hoc Routing (RDMAR), uses a relative-distance micro-discovery procedure for route discovery, which has a local effect, and thus limits the propagation of route-requests to a localised region (Aggelou, G., et. al., 1999). It achieves this by measuring the distance between the source and the destination, however, this can only be applied if there is at least one record of communication between the source and the destination. The advantage of this technique is that the propagation of route-requests is generally low, conserving significant bandwidth and battery power. Another protocol which attempts to reduce the control-overhead of far-reaching route-requests is the Location Aided Routing (LAR) (Ko, Y.-B. and Vaidya, N. H., 1998), which uses location information obtained by GPS. The protocol defines two schemes: a boundary-restricted, where the propagation of a route-request is bounded, within a certain area; and a coordination-oriented, where the route-request contains the actual coordinates of the destination, allowing the packets to travel only towards the destination. Both proposed schemes reduce the route discovery overhead, and thus save bandwidth and conserve battery life. However, the main disadvantage is that each node is required to be equipped with GPS.

A novel agent-based routing protocol is the Ant-colony-based Routing Algorithm (ARA), which utilises light-weighted mobile agents, called ants, and ants' basic food search behaviour, for route discovery and route maintenance (Bouazizi, I., 2002). When a source node requires a route to a destination, it broadcasts a forward ant (FANT) to all of its neighbouring nodes. The ants propagate according to standard flooding algorithms, and leave a pheromone at each node they visit. The pheromone value at each node is equal to the number of hops the ant took in order to reach this node. Once the destination node is reached, a backward ant (BANT) is created with purpose to return to the source. Route maintenance involves the increase and decrease of pheromone values kept on intermediate nodes. For example, each time a data packet is routed through an intermediate node the pheromone value of that node is increased, otherwise the pheromone value is decreased over time until it expires. A basic drawback of this approach is the relatively slow migration of ants, and consequently the delays experienced by route discovery.

Another novel routing protocol, which uses link-failure prediction, is the Flow Oriented Routing Protocol (FORP), which aims to forecast, when a route is going to be broken, and thus assist the transmitting node to switch to an alternative route before experiencing route

failure (Su, W. and Gerla, M., 1999). It achieves this by calculating the link expiration time (LET) for each pair along the route (using GPS) in which the Flow_REQ packet propagates, and appends this to the packet. Once the Flow_REQ reaches the destination, a route expiration time is calculated using the minimum of all the LETs, and a Flow_SETUP is sent back to the source. The source can then initiate data transmission over the route provided by the Flow_SETUP packet. This allows the destination to predict when a link failure is likely to occur and informs the source by transmitting a Flow_HANDOFF message to the source, which can then switch its data transmission to an alternative route. This strategy can significantly improve real-time data transmissions, however the flooding nature of the protocol results in scalability problems in large ad-hoc networks.

The Cluster-Based Routing Protocol (CBRP), organises the participating nodes into logical clusters (Jiang, M., et. al., 2001), in a similar manner to CGSR, where in each cluster there can exist only one cluster-head which is responsible for location management and intra-cluster routing, whereas gateways typically lie on the edges of two, or more clusters, and are responsible for inter-cluster routing. Unlike CGSR, this protocol uses an on-demand route discovery process, where route-requests are always propagated along a repeated sequence of alternating cluster-head and gateway node pair(s). This is CBRP's main advantage, as control packets are not flooded throughout the complete network topology, and thus network overhead is far less compared to traditional flooding techniques. However, as in most clustering protocols, the clustering formation, and maintenance, imposes additional network overhead. In addition, this protocol does not support multiple routes for a single destination, nor it provides a means of selecting *optimal* routes.

Although the on-demand method has been proved to reduce high network overhead imposed by the proactive method, most routing protocols which belong in this category share high routing overheads when considering the worst case scenario. This is a result of the underlying flooding mechanism, which most protocols conform to, according to which route-request packets have to be disseminated throughout the whole network. Hierarchical routing protocols, such as CBRP, attempt to minimise control overheads by partitioning the network into a number of logical domains. In particular, CBRP organises the nodes into small adjacent clusters in which a central cluster-head node is mainly responsible for routing within its cluster and maintaining location management among its members. During the route discovery process, only cluster-heads and gateway nodes (nodes situated at the edges of two or more clusters) exchange route-request messages, resulting in an overall significant overhead reduction. However, in highly dynamic networks, CBRP may incur significant amounts of overhead due to frequent reorganisation of clusters. On-demand network discovery can also cause increased latency, as data packets are normally buffered until a suitable route to a des-

tination is discovered.

D.7 Hybrid

The hybrid routing method is based on a combination of the proactive and reactive methods. Hybrid routing protocols are innovative, which aim to increase scalability, and, at the same time, reduce the on-demand route discovery overhead. Protocols in this category normally organise the network into a number of logical structures, such as zones, trees, or clusters, where, at each structure, nodes exchange routing information proactively, while they initiate route discovery for distant nodes, that is, for nodes that do not belong to the same domain. Only a few hybrid routing protocols have been defined in the literature, so far, as it is a fairly new concept (Abolhasan, M., et. al., 2004). A well-known hybrid protocol is the Zone Routing Protocol (ZRP) that defines a routing zone in hops, which is fixed for every participating node (Haas, Z. J., et. al., 2002a). Within this routing zone, nodes maintain topological knowledge proactively, and thus a route from one node to another within the routing zone is always available. In contrast, nodes that require a route to a destination, which is outside of their routing zone, use an on-demand approach. The main advantage of ZRP is its ability to reduce the amount of information exchanges when compared to pure proactive protocols, and, at the same time, reduce the delays associated with pure reactive protocols. The main disadvantage of ZRP is its limited flexibility of the threshold value which defines the routing zone, and forces the protocol to behave more proactively for large values and more reactively for small. Nodal movements does not cause burn.

The Zone-based Hierarchical Link State (ZHLS) uses the zone concept in a different way to ZRP, that is, zones are non-overlapping, and GPS is used to calculate the node- and zone-ID, which are used for location management (Joa-Ng, M. and Lu, I.-T., 1999). Thus, it dismisses the requirement for a central node, and eliminates the processing overhead often encountered with cluster-heads. ZHLS has been shown to reduce route discovery overhead, when compared to pure flooding techniques, by allowing a source node searching for a remote destination to broadcast a zone-level broadcast request to all other zones. In addition, nodal movements, within the current zone, do not cause sources to initiate a new location search, which generally comes in contrast with standard reactive protocols. The main disadvantage of this protocol is that all nodes must be equipped with GPS, and have a pre-programmed static zone map. Similarly to ZHLS, the Scalable Location Update Routing Protocol (SLURP organises the nodes into non-overlapping zones (Woo, S.-C. and Singh, S., 2001). The novelty of SLURP is in its route discovery process, where, unlike ZHLS, it assigns a home region to each node in the network, which is determined using a static map-

ping function, and provides location information for its registered nodes to requesting nodes, and thus eliminates a full-scale route discovery. Similarly to ZHLS, the main disadvantage is the pre-programmed static zone map.

A tree structure is used by the Distributed Spanning Trees-based routing protocol (DST) that aims to organise the nodes under the control of a central node, called *root*, which decides on whether the tree should merge with another tree or not (Radhakrishnan, S., et. al., 1999). The routing algorithm uses the forest of spanning trees to perform routing, using a process called *shuttling* in combination with holding the packets at the intermediate nodes. Simulation experiments have shown that increasing the holding time in stable and high-connectivity systems can significantly improve reachability, while for systems with high-disconnectivity which are highly-dynamic, increasing holding time does not significantly improve reachability. The main disadvantage of this algorithm is that the *root* node is a single-point-of-failure. Unlike DST, the Distributed Dynamic Routing (DDR) does not require a *root* node, as it gathers the required information by periodic *HELLO* message exchanges between neighbouring nodes. The trees in the network are linked together through gateway nodes, and thus form a forest. The DDR algorithm consists of six fundamental phases: preferred neighbouring election; forest construction; intra-tree clustering; inter-tree clustering; zone naming; and zone partitioning. Route discovery is accomplished by the hybrid ad-hoc routing protocol (HARP) (Nikaein, N., et. al., 2001), which uses the intra- and inter-zone routing tables created by DDR to determine stable paths. The advantage of DDR when compared to ZHLS is that it does not rely on a static zone map, and, unlike DST, it does not rely on root nodes for coordination. However, nodes in DDR choose preferred neighbours, which are responsible for routing of most of the data packets, and, thus may, become performance bottlenecks.

Hybrid protocols improve on scalability, while significantly reducing the frequency of route discovery requests, and thus decrease the amount of routing overhead. As previously mentioned, protocols in this category usually define a structure, which acts similarly to a network backbone, allowing nodes belonging in that structure to work together by exchanging routing information. When a node wishes to communicate with another node which belongs to a different logical domain, route discovery is initiated. The direct benefit from this approach is that route-requests can mainly be issued by nodes that are more suitable than others, for example, nodes that are situated at the edges of their logical domains, and thus force the minimum propagation of the route-request packets.

Ten of the most popular routing protocols are being examined into the following sections, in greater detail. These protocols include the proactive DSDV, GSR, DREAM, and CGSR, the reactive DSR, AODV, TORA, CBRP, and hybrid ZRP and DST. The analysis is based on the protocols' route discovery and maintenance processes, as well as their fundamental

properties, and data packets routing.

D.8 Highly Dynamic-Sequenced Distance-Vector (DSDV)

As previously mentioned, this protocol is a modification of the DBF (Bertsekas, D. and Gallager, R., 1987) routing algorithm, and addresses problems of BDF related to poor looping properties in the face of broken links, and also related to time dependencies of the interconnection topology. The DSDV protocol proactively builds and maintains routing tables at each network node, which are used to route data packets along the most optimal routes. A node's routing table lists all available destinations and the numbers of hops required to reach each of them. Each routing table entry is tagged with a sequence number which is originated by the destination node. It achieves loop-freedom by tagging each route table entry with a sequence number so that nodes can distinguish stale routes from the new ones. DSDV tries to maintain routing tables, completely updated for all connections, at all times, by enforcing that each node to periodically broadcast its routing table, and dynamically transmit updates when significant changes occur. The data broadcasted by each node contain the node's new sequence number, along with the following information for each entry:

- The destination's IP address.
- The number of hops required to reach the destination.
- The destination's sequence number, as originally stamped by the destination.

In addition, the routing tables also contain the hardware address, and the network address of the transmitting node, within the headers of the packet. Sequence numbers, coupled with the number of hops that are required to reach a destination, indicate how fresh and short a route is. According to DSDV, routes with more recent sequence numbers are always preferred for making forwarding decisions, but, are not necessarily advertised. Of the paths with the same sequence number, those with the smallest number of hops are used. Due to mobility, broken links are likely to occur, which are detected by either the layer 2 protocol, or inferred, if no broadcasts have been received for a while from a former neighbour. When a node senses, or infers, that the link to the next hop has broken, it immediately assigns an infinity metric (∞) to each route in its routing table that was using that link as an intermediate hop, and also assigns an updated sequence number. Situations like this qualify as a substantial routing change, and therefore modified routes are immediately disclosed in a broadcast routing information update, and are disseminated through the network.

In an attempt to reduce network overhead created by periodic broadcasts, DSDV defines

two types of broadcasted packets. The first one is called *full dump*, which includes all routing information, while the second one is called *incremental*, which includes only the updated routing information since the last *full dump*. When nodes receive routing updates, they compare the information of the update to the information already existing into their routing tables. In case that an updated route has the same, or a higher sequence number, but smaller number of hops, it replaces the existing entry with the entry found in the update. Figure D.4 illustrates a movement scenario in an ad-hoc network topology.

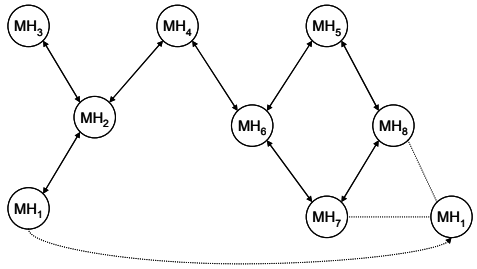

Figure D.4 (Perkins, C. and Bhagwat, P., 1994): DSDV - movement in an ad-hoc network scenario

As an example, the routing table maintained at MH_4 (*see* Figure D.4) is presented in Table D.1. Assuming that the address (MAC/IP) of each mobile host is represented as MH_i, and sequence numbers are denoted as *SNNN_* MH_i. The installation time, flags, and stable data fields have been removed for simplicity compared to the original example which can be found in (Perkins, C. and Bhagwat, P., 1994).

Table D.1: DSDV - Structure of the MH_4 forwarding table

Destination	*Next Hop*	*Metric*	*Sequence number*
MH_1	MH_2	2	S406_MH_1
MH_2	MH_2	1	S128_MH_2
MH_3	MH_2	2	S564_MH_3
MH_4	MH_4	0	S710_MH_4
MH_5	MH_6	2	S392_MH_5
MH_6	MH_6	1	S076_MH_6
MH_7	MH_6	2	S128_MH_7
MH_8	MH_6	3	S050_MH_8

If it is assumed that MH_1 moves away from MH_2, and towards the general vicinity of MH_7 and MH_8, then the routing table maintained at MH_4 will looks as shown in Table D.2. The difference in the metric, next hop, and sequence number for destination MH_1, is obvious. The example assumes that in the intervening time, many new sequence numbers have been received. The first entry must thus be advertised in subsequent incremental routing informa-

tion updates until the next full dump occurs. The new routing information for MH_1, maintained by MH_4, has been received along a chain of broadcasts, starting from MH_7 and MH_8, which sensed the addition of a new node (MH_1), and thus broadcasted the new information which was received and re-broadcasted by MH_5 and MH_6, and finally received by MH_4. Accordingly, routing updates are disseminated throughout the whole network.

Table D.2: DSDV - Structure of the MH_4 updated forwarding table

Destination	*Next Hop*	*Metric*	*Sequence number*
MH_1	MH_6	3	$S516\_MH_1$
MH_2	MH_2	1	$S238\_MH_2$
MH_3	MH_2	2	$S674\_MH_3$
MH_4	MH_4	0	$S820\_MH_4$
MH_5	MH_6	2	$S502\_MH_5$
MH_6	MH_6	1	$S186\_MH_6$
MH_7	MH_6	2	$S238\_MH_7$
MH_8	MH_6	3	$S160\_MH_8$

D.9 Global State Routing (GSR)

The global state routing protocol (GSR) was especially designed for routing in ad-hoc wireless environments. The underlying principle of this protocol is that each node is required to exchange vectors of link-state information with its neighbours during routing information exchange. Based on the link-state vectors, nodes are able to maintain a global knowledge of the network topology, and optimise their routing decisions locally. GSR is based on the link-state algorithm (McQuillan, J. M., et. al., 1980), and thus every node in the network maintains the knowledge of the full network topology, at all times. However, in contrast to link-state algorithm, it uses an alternative method to standard flooding for disseminating link-state updates, and is based to DBF algorithm (Bertsekas, D. and Gallager, R., 1987). Overall, the DBF algorithm requires no flooding, and may thus be an improved solution for reduced network overhead.

With GSR, each node i maintains one list and three tables: a neighbour list (A_i); a topology table (TT_i); a next hop table ($NEXT_i$); and a distance table (D_i). A_i is defined as a set of nodes which are adjacent to node i. Each destination j has an entry in table TT_i which contains two parts: $TT_iLS(j)$ and $TT_iSEQ(j)$. $TT_iLS(j)$ denotes the link state information reported by node j, while $TT_iSEQ(j)$ denotes the timestamp, which indicates the time node j required to generate this link-state information. Similarly, for a destination j, the $NEXT_i(j)$ denotes the next hop to forward packets destined to j on the shortest path, and $D_i(j)$ denotes the distance of the shortest path from i to j.

Initially, each node has an empty A_i and TT_i. After proper initialisation of its local variables, node *i* examines its inbound queue for incoming messages. If the inbound queue is not empty, the node examines the sender field of each incoming message, and places the address in its neighbour list A_i. Node *i* then processes all received routing messages, which contain the link-state information broadcasted by its neighbours. GSR requires that each node *i* compares the *freshness* of the embedded sequence number (*pkt.SEQ(j)*), with the ones stored in *i*'s local storage, for each destination. If any entry in the incoming message has a newer sequence number regarding destination *j*, then $TT_iSL(j)$ will be replaced by *pkt.LS(j)*, and $TT_iSEQ(j)$ will be replaced by *pkt.SEQ(j)*. When all routing messages are examined, node *i* rebuilds the routing table, based on the newly-computed topology table, which is then broadcasted to its neighbours. This process is then performed periodically.

The key difference between GSR and the standard link-state algorithm is the way routing information is disseminated. In link-state, whenever a node detects topological changes, it generates, and floods the network, with link-state update packets. In contrast, nodes in GSR maintain a global knowledge of the network topology, based on the up-to-date information received from neighbouring nodes, and, periodically, exchange it with their local neighbour nodes, only. Information *freshness* is guaranteed by the use of sequence numbers, similar to DSDV (Perkins, C. and Bhagwat, P., 1994). GSR's FindSP algorithm is used to create the shortest-path tree rooted at node *i*, and is based on the Dijkstra's algorithm with modifications, so that the next hop table ($NEXT_i$) and the distance tables (D_i) are computed in parallel with the tree reconstruction.

D.10 Distance Routing Effect Algorithm for Mobility (DREAM)

The distance routing effect algorithm for mobility protocol (DREAM) is designed for mobile ad-hoc networks. This protocol is based on two ideas, the distance effect, and the triggering of sending location updates. The first one is based on the fact that the greater the distance separating two nodes, the slower they appear to be moving with respect to each other. In relation to this observation, it is possible to update location information found in routing tables as a function of the distance separating nodes, without compromising routing accuracy. The second mechanism requires that each node autonomously initiates the sending of location updates, based on its own mobility rate. As a result, nodes with low-mobility patterns are required to transmit their routing information less frequently as opposed to nodes with high mobility patterns. Thus, each node can optimise the frequency at which it sends updates to the network, and thus reduce the bandwidth and energy used, leading to a dis-

tributed and self-optimising system.

In contrast to proactive and reactive routing protocols, where a node either stores the source route to a destination or just to the next hop, respectively, DREAM requires that each node stores the location information for every other node into its own routing table. In order to achieve this, each node must be equipped with GPS (Kaplan, E. D., 1996), which provides geographical coordinates. Briefly, when a node *A* wants to send a message *m* to node *B*, it uses the location information for *B* to obtain *B*'s direction, and then transmits *m* to all its one-hop neighbours in the direction of *B*. This process is repeated by each neighbour, until *B* is eventually reached. The location information dissemination process can influence the probability of finding *B* in the computed direction. As previously mentioned, each node transmits its current location to all the other nodes, and, at a frequency, which results from the following:

- **The distance effect**. Nodes that are far-apart need to update each other's location less frequently than nodes which are closer. This is realised by associating with each control message an *age*, which corresponds to how far from the sender that message travels.
- **The mobility rate**. Nodes moving faster than others need to communicate their location more frequently. This allows each node to make precise judgements of its dissemination frequency, and thus reduce overhead by transmitting location information only when needed, and without sacrificing the route accuracy.

DREAM may be considered as bandwidth- and energy-efficient, especially when compared to other protocols of the same category (proactive). This is due to the fact that each control message caries only the coordinates and the identifier of a node, and thus the message is small compared to messages used by proactive protocols, which often require the transmission of the complete routing tables. DREAM is inherently loop-free, as data packets always travel away from the source and towards a certain destination, and it is robust as data messages can reach the intended destination by following, possibly, independent routes. Finally, and most importantly, it is adaptive to mobility, as the frequency with which the location information is disseminated depends on the mobility rate.

D.11 Cluster-head Gateway Switch Routing (CGSR)

The cluster-head gateway switch routing protocol (CGSR) was designed for multi-hop, mobile wireless networks, and is based on a cluster-head token infrastructure, which uses the least cluster change (LCC) algorithm in order to provide a stable clustering structure and ra-

dio channel code allocation. CGSR's cluster-head controlled token protocol allocates channel access within a cluster-head, and facilitates data packet forwarding. The CGSR scheme delivers packets efficiently and provides cluster-head token scheduling and gateway code scheduling. In addition, path reservation support makes token and code scheduling more efficient.

CGSR uses a distributed clustering algorithm for cluster formation, which can either be lowest-ID or highest-connectivity (Gerla, M. and Tsai, J. T.-C., 1995). Accordingly, a node is elected to be a cluster-head among a set of network nodes, and, thus, a cluster is formed. All nodes within transmission range of the cluster-head belong to the cluster defined by the cluster-head. Thus, all nodes that belong to the same cluster can directly communicate with the cluster's cluster-head, and, possibly, with each other. When a node is directly linked to more that one cluster-heads, it is called a gateway, and can be used for inter-cluster routing. The complexity and overhead associated with cluster formation is related to cluster-head selection. For example, a cluster-head which is stationary, such as dedicated workstation, is likely to cause fewer cluster reformations, as frequent cluster-head changes adversely affect the performance of the routing protocol, and thus CGSR uses a least cluster change clustering (LCC) algorithm. This defines only two conditions which can cause the cluster-head to change:

- Two cluster-heads come within range of each other, such as due to mobility.
- A node becomes disconnected from any other cluster.

CGSR takes advantage of clustering organisation, and allocates wireless channels among different clusters. Across clusters, it enhances spatial reuse by different spreading codes, such as CDMA (Gerla, M. and Tsai, J. T.-C., 1995). Within a cluster, CGSR uses a cluster-head controlled token protocol, such as polling, in order to allocate the channel among competing nodes. This approach is designed to give higher priority to cluster-heads over ordinary nodes, and thus maximise channel utilisation and decrease delays. This is especially true, as cluster-heads require more chances to transmit, as they are in charge of broadcasting within the cluster, and for forwarding messages between member nodes which are not directly linked. The channel access scheme in CGSR works as follows:

1. The cluster-head gets the permission token to access the radio channel, and transmits any messages it has in its transmission queue.
2. The cluster-head passes the token to one of its neighbours, according to a separately-defined scheduling algorithm.

3. The member node returns the token to its cluster-head after it has transmitted its message(s).
4. Repeat 1 to 3.

In CGSR, when a gateway node wishes to communicate with one of its clusters, it must select the code used by that cluster. In particular, the gateway can tune its code to match another cluster's code, receive the permission token, communicate, and, when finished, change its code again to match the other cluster's spreading code. Code scheduling can dramatically affect the message delivery performance, as, if gateway nodes are tuned to different codes than the codes in which data packets are transmitted to them, results in loss. An alternative approach is to allow multiple radio interfaces to be used by each gateway node. In this way, a gateway node could access multiple cluster channels by selecting corresponding codes for each wireless interface, and thus reduce gateway conflicts and improve packet delivery performance.

Routing in CGSR is based on a modified DSDV scheme (Perkins, C. and Bhagwat, P., 1994), which uses a hierarchical structure to route data packets. In particular, each node maintains a cluster member table which records the destination cluster-head for each node, and broadcasts it, periodically. A node updates its cluster member table when it receives new routing information from its neighbours. Routing information is then tagged with sequence numbers (Perkins, C. and Bhagwat, P., 1994) in order to avoid stale tables, in a way similar to DSDV. In more detail, each node maintains two tables: a cluster-member table; and a routing table. The cluster-member table is used to map a destination address to the destination cluster-head address, while the routing table is used to select the next node to reach the destination. When a node gets the permission token, it initially selects the shortest (minimal hop) destination cluster-head according to the cluster member table and routing table, and then selects the next node to transmit for that destination cluster-head, according to the routing table.

D.12 Dynamic Source Routing (DSR)

The dynamic source routing protocol (DSR) was specifically designed for multi-hop wireless ad-hoc networks with high mobility, and allows nodes to dynamically discover a source-route across multiple network hops to any destination in the ad-hoc network. A source node, wishing to transmit network traffic to a destination, creates the data packets and appends the complete, ordered list of nodes through which the packet will pass, in their headers. This way it allows packet routing to be loop-free and avoids the need for up-to-date routing informa-

tion in the intermediate nodes through which the packet is forwarded. Thus, the protocol imposes significantly less network overhead, and is able to react quickly to changes in the network's topology.

The route discovery and route maintenance processes operate on an on-demand manner. In particular, DSR does not require any periodic routing advertisements, link-status sensing, or neighbour-detection packets, and does not rely on any underlying network protocol. This entirely, on-demand behaviour, and lack of periodic activity, allows the number of overhead packets caused by DSR to scale all the way down to zero, when there is no mobility in the network, and all routes needed have been discovered. If mobility is introduced in the network, the routing overhead is only concentrated in an effort to maintain the routes currently in use. Another advantage of DSR is that, in addition to route discovery, a node can learn and cache multiple routes to a destination by investigating the packets' source addresses which are a promiscuous overheard. This allows the reaction of changes to be much more rapid, as a node with multiple cached routes can immediately use an alternative route to a destination in case of primary route failure, and can thus reduce overhead and latency by multiple route discovery calls. Furthermore, DSR supports unidirectional and asymmetric routes, as well as bidirectional and symmetric, thus improving overall performance and network connectivity.

Figure D.5 shows a simple case of a route discovery initiated by node *A* for node *E*, and a route-reply message sent back to *A* by *E*. Initially, node *A* transmits a route-request message as a single local broadcast packet, which is received by all nodes currently within *A*'s wireless transmission range, including *B*, in this case. Each route-request packet contains the address of the initiator and target, and also contains a unique identification number created by the source. Along with this, a route-request packet also contains the ordered list of intermediate node addresses by which this particular copy has been forwarded, assuming that the packet has now been received by *B*. Initially, node *B* will examine the unique identification number of the route-request packet and compares it to the packets' of the same kind which it has recently seen. In case of a match, the packet will be dropped, otherwise, it will proceed by examining the list of addresses contained in the route-request. If node *B* finds its own address in the list, it drops the packet. Otherwise, it proceeds by examining the destination address of the route-request. If node *B* is the target of the route discovery, it will return a route-reply back to the initiator, giving a copy of the accumulated route record from the route-request. When the initiator receives this route-reply, it caches this route in its route cache for use in sending subsequent packets to this destination.

In the case that node's *B* address does not match to the destination address of the route-request packet, node *B* appends its own address and transmits it as a local broadcast packet

with the same identification number. This process will be iterated by all nodes along the chain to node *E* (*C* and *D*), until, eventually, the route-request packet arrives at node *E*.

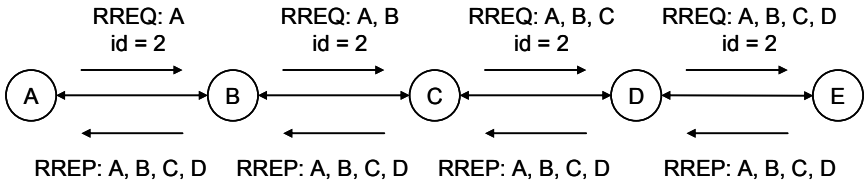

Figure D.5: DSR - Route-request & Route-reply propagation

Once node *E* receives the route-request, it searches its own route cache for a route to *A*, and if one is found, it uses it as a source route for delivering the route-reply, which contains the route from *A* to *E*. Node *E* could have just reversed the route which was taken by the route-request. However, this is not allowed, as there is no guarantee that the links along the route from *A* to *E* are bidirectional. In the case that node *E* does not find a route to *A* in its route cache, it will initiate a route discovery to *A* by piggybacking on the packet containing its won route-request for *A*, in order to avoid infinite recursions.

When originating, or forwarding a packet, using a source route, each node transmitting the packet is responsible for confirming that data can flow over the link from that node to the next hop. For example, in Figure D.6, node *A* has originated a data packet for node *E* with the source route through intermediate nodes *B*, *C*, and *D*. Node *A* is responsible for the link from itself to *B*, node *B* for the link from itself to *C*, and so on. DSR relies on MAC protocol acknowledgments, such as IEEE 802.11 (IEEE Standards, 802.11, 1999)) or *passive acknowledgments* (Jubin, J. and Tornow, J. D., 1987) for confirmation of the capability of a link to carry data. In the case where a built-in acknowledgment mechanism is not available, the node transmitting the packet can explicitly request a DSR-specific software acknowledgment to be returned by the next node along the route. If the acknowledgement request has been retransmitted, the maximum allowed number of times, without any acknowledgment having been returned, the sender treats the link to this next-hop destination as currently *broken*. This means that it removes this link from its route cache and returns a route error to each node that has sent a packet routed over that link since an acknowledgment was last received.

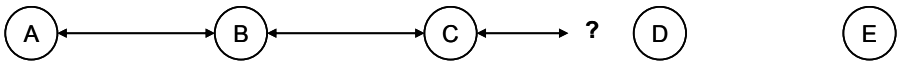

Figure D.6: DSR - Route maintenance

D.13 Ad-hoc On-demand Distance Vector (AODV)

AODV is intended to be used by mobile nodes in an ad-hoc network, and offers quick adaptation to dynamic link conditions, low processing and memory overhead, and low network utilisation. With this, sequence numbers are used to ensure loop freedom at all times, even in the face of the anomalous delivery of routing control messages, and eliminate problems, such as *counting-to-infinity*, often associated with classical distance vector protocols. AODV allows the rapid discovery of routes for new destinations, and does not require nodes to maintain routes to destinations that are not in active communication. When a link breaks, AODV causes the affected set of nodes to be notified, so that they are able to invalidate routes using the lost link. Destination sequence numbers is an essential feature of the protocol, as nodes choose between multiple routes to a single destination, based on the route with the greater sequence number.

This protocol defines three message types: route-request (RREQ); route-reply (RREP); and route error (RERR), which are received by UDP, and normal IP header processing applies. For example, RREQ message originating from node *A*, and intended for node *C*, will have *A*'s source IP and *C*'s destination IP in the packet's header, and will be broadcasted to the IP limited broadcast address (255.255.255.255). Fragmentation of these packet types is usually not required.

When a node requires a new route to a destination, it broadcasts a RREQ message, which is propagated until it reaches the destination or an intermediate node which has a *fresh enough* route to that destination. A *fresh enough* route is a route entry for that destination whose associated sequence number is at least as great as that contained in the RREQ. If this is the case, the intermediate node unicasts a RREP message back to the originator of the RREQ. Each node receiving a RREQ message caches a route back to the originator of the request, so that the corresponding RREP can be unicast from the destination along a path to that originator, or likewise from any intermediate node that is able to satisfy the request.

Nodes are required to monitor the link status of next hops that belong to active routes. In the event where a node senses a link breakage with a neighbouring node, it creates a RERR message to notify other nodes, which are likely to use the broken link, that the link is no more available. In order to enable this reporting mechanism, each node keeps a record (*precursor list*) of neighbouring nodes' IP addresses which used this node as a router to a destination which required the now unavailable link.

As previously mentioned, each node is required to cache a reverse route for each originator of a RREQ packet. Routing tables are used for caching, and have the following fields for each route table entry: destination IP address; destination sequence number; valid destination

sequence number; interface; hop counting, that is, the number of hops needed to reach the destination; next hop; list of precursors; lifetime; routing flags; and state.

D.14 Temporally Ordered Routing Algorithm (TORA)

TORA is a distributed routing protocol for multi-hop wireless networks, which attempts to uncouple the generation of far-reaching control messages propagation from the dynamics of the network topology. The underlying algorithm is a member of a class referred to as link-reversal algorithms, which is used for building a loop-free, multi-path routing structure which is used as the basis for forwarding traffic to a given destination. A key advantage of TORA is its support for both source-initiated, on-demand routing for some destinations, and destination-initiated proactive routing for other destinations.

In TORA, network nodes need only to maintain routing information about adjacent nodes, and maintain state on a per-destination basis similar to the distance-vector routing approach. However, the metric used to establish the routing structure of the network does not represent distance, which is a common case for distance-vector approach. The dual support for on-demand and proactive routing provides greater flexibility, and allows TORA to be configured according to requirements. For example, in highly dynamic topologies, TORA may be configured to act reactively, and thus have a sparse network overhead, as it is likely to be inefficient to maintain routes between every source/destination pair, at all times. On the other hand, in cases where routes are essential to a number of destinations at all times, such as with servers or gateways in infrastructure networks, proactive operation can be beneficial. TORA is thus designed to minimise the communication-overhead associated with adapting to network topological changes. It achieves that by reducing the scope of control messaging to a localised manner, affecting a small set of nodes near a topological change.

For proper operation, TORA requires lower-layer mechanisms, or protocols, that provide the following basic services between neighbouring nodes: link status sensing and neighbour discovery; reliable, in-order control packet delivery; link and network layer address resolution and mapping; and security authentication. TORA assigns directions to the links between nodes to a routing structure that is used to forward data packets to the destination. A node assigns a direction to the link with a neighbouring-node based on their relative metric values, which can be either *upstream* or *downstream*. The metric associated with each node can be conceptually interpreted as the node's *height*. Links are directed from the *higher-node* to the *lower-node*, and, thus, a node may only forward packets downstream. Unknown, or undefined link directions, prohibit their usage for packets forwarding. Collectively, the nodal heights and the link-directional assignments form a loop-free, multi-path routing structure in

which all directed paths lead downstream to the destination.

An example of a multi-path routing structure is presented in Figure D.7. Suppose that the relative heights associated with the nodes *A*, *B*, *C*, *D*, *E*, and *DEST* are the following:

- H (C) > H (B) > H (E) > H (DEST)
- H (D) > H (A) > H (E) > H (DEST)

According to Figure D.7, it can be noted that although node *C* is closer to the destination than node *B* in terms of hops count, the height metric of *C* is greater than that of *B*. Thus, there is absolutely no guarantee that routes selected for any particular destination will involve the minimum number of hops, as TORA does not base its route selection on a hop-counting mechanism. TORA's main functions include: the creation; maintenance; deletion; and optimisation of routes. Creating routes corresponds to the selection of heights to form a directed sequence of links leading to the destination in a previously undirected network, or portion, of the network. Maintaining routes refers to the dynamic adaptation of the routing structure, in order to respond to topological changes. For instance, the loss of a node's downstream link may result in a structure that has no possible path leading to the destination. Such an event, triggers a sequence of link reversals which re-orient the routing structure, such that all directed paths lead to the destination again. In the case that the network becomes partitioned, resulting in some links becoming partitioned from the destination, these links must be marked as undirected in order to erase invalid routes. Optimisation is then the process in which nodes reselect their heights, in order to improve the routing structure.

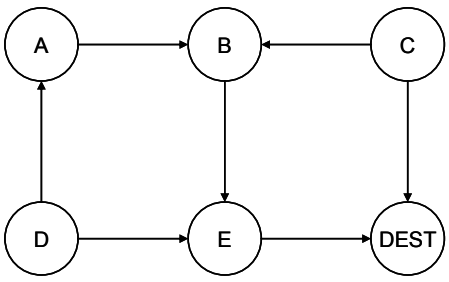

Figure D.7: TORA - A multi-path routing structure

D.15 Cluster-Based Routing Protocol (CBRP)

CBRP is designed for mobile ad-hoc networks, and imposes a hierarchy to the network by organising the nodes into a number of overlapping, or disjoint, two-hop diameter clusters, in

a distributed manner. A single cluster-head is elected for each cluster, and is responsible for maintaining cluster membership information. CBRP discovers inter-cluster routes, on-demand, using the cluster membership information kept at each cluster-head. The main benefit of clustering is that flooding traffic is efficiently minimised during route discovery, and that the process requires less time to complete. CBRP also takes into account both bidirectional and unidirectional links, and uses them for both intra- and inter-cluster routing. CBRP bases its clustering formation process to the lowest-ID algorithm (Gerla, M. and Tsai, J. T.-C., 1995), which is presented in Appendix B.7.

In order to maintain the clustering formation, CBRP requires, from each node, to periodically broadcast a *HELLO* message, which contains the node's address and role, and its neighbour table. Assuming that node *A* receives a *HELLO* message from node *B*, it performs the following actions:

- It checks if *B* is already in the neighbour table, and, if it is not, it adds one entry for *B*.
- If *B*'s neighbouring table contains *A*, then node *A* marks the link to *B* as bidirectional in the relevant entry.
- If *B* is a cluster-head, node *A* marks *B* as a cluster-head in the entry.

Each entry in the neighbouring table is associated with a timer. Once the timer expires the entry is removed from the table. In order to avoid frequent cluster-head changes, CBRP uses the following rules:

- A non cluster-head never challenges the status of an existing cluster-head.
- Only when two cluster-heads move next to each other, one of them looses the cluster-head role (to one with the higher ID).

Figure D.8 presents a route discovery scenario, initiated by node *S* for destination node *D*. As shown, only cluster-heads (*1*, *6*, and *8*) and gateways (*4*, *9*, and *2*) participate in the route discovery process. When node *D* receives the RREQ packet, it immediately creates a RREP and fills the cluster-head address entries with the list found in the RREQ. Thus, the RREP is sent back to the source along the same line of cluster-heads, which was previously used by the route discovery (Figure D.9). Each cluster-head along the way checks reachability between the node that received the RREP and the node which is the next hop along the route. If reachability is verified, the current node sends the packet to the next hop without recording its own address in the source route returned back to *S*. Figure D.10 shows the source route now being used by *S* as '$S \rightarrow 4 \rightarrow 9 \rightarrow D$', instead of '$S \rightarrow 1 \rightarrow 4 \rightarrow 9 \rightarrow 8 \rightarrow D$', which was the original route that the RREQ took.

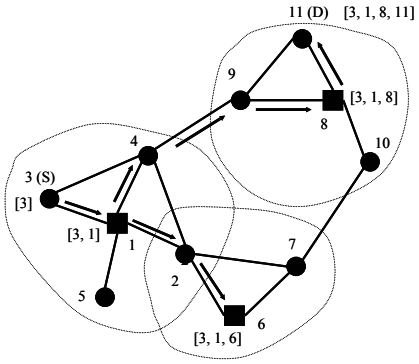

Figure D.8: CBRP - Route Discovery (S to D)

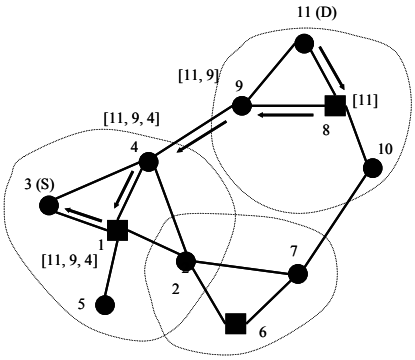

Figure D.9: CBRP - Route-reply (D to S)

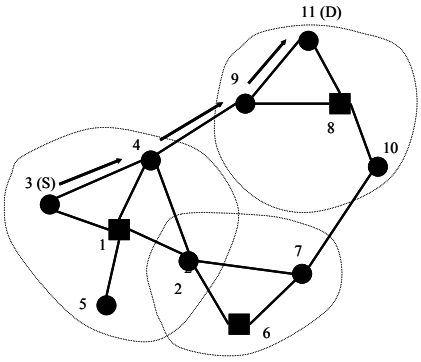

Figure D.10: CBRP - Transmission (S to D)

The actual routing of data packets in CBRP is achieved by source routing, similar to (Johnson, D. B., et. al., 2004). The most recent version of CBRP provides two additional mechanisms: route shortening; and local repair. The first mechanism is used to shorten the source route of the data packet being forwarded, and informs the source node about the shortest route, in terms of hops. The second mechanism is used to automatically repair a broken route, and thus avoid route re-discovery by the source.

D.16 Zone Routing Protocol (ZRP)

ZRP is designed for a wide variety of mobile ad-hoc networks, especially those with large network spans and diverse mobility patterns. It uses the routing zone concept, which is defined as a geographical region (in hop counts) in which a network node has full knowledge of the entire zone's topology. However, a node which requires a route to a destination outside of its own routing zone, reactively discovers a route by an on-demand route discovery

mechanism. By combining these principles, the protocol aims to improve the efficiency of a globally-reactive route discovery mechanism, as well as improving the quality of discovered routes, by making them more robust to changes in network topology. ZRP can be configured to adapt the needs of a wide range of distinct ad-hoc networks, by proper selection of a single parameter: the routing zone radius.

Some of the parameters that need to be taken into consideration, for optimally selecting the routing zone radius, are: routing information demand; and mobility. A number of cases where these two parameters can be varied are:

- High routing information demand and slow mobility, in this case, large routing zones are preferred.
- Fixed topology and consequently no movement, in this case, the ideal routing zone radius would be infinite.
- Low routing information demand and moderate mobility, in this case, small routing zones may be preferable.
- High routing information demand and high mobility, in this case, the most appropriate would be routing zone of one-hop radius).

In addition, ZRP can be fine-tuned by implementing individual adjustments on each node's routing zone, in order to adapt to network scenarios whose behaviour may vary across different regions. It thus requires that each node periodically exchanges neighbouring discovery messages for routing zones maintenance. In addition to bidirectional links, ZRP also provides support for unidirectional links, as long as the link source and link destination lie within the other's routing zone. A node's routing zone is then defined as a collection of nodes whose minimum distance in hops from the node is no greater than the zone radius. For example, Figure D.11 shows node *A*'s routing zone, as well as node *B*'s, for a radius of 2-hops. The area drawn by the dashed line illustrates node *A*'s routing zone, while the area drawn by the dotted line illustrates node *B*'s routing zone. Each node within node *A*'s and node *B*'s routing zone is a maximum of 2-hops away. For instance, node *D* and node *F* are two hops away from *A*, while node *C* and node *F* are 2-hops away from node *B*. The intra-zone routing protocol (IARP) is responsible for proactively maintaining routes to destinations within a routing zone (Haas, Z. J., et. al., 2002c). For instance, node *A* knows the route to every node, such as node *E* and node *F*, within its own routing zone, at all time. It should be noted here that node *G* is outside node *A*'s and *B*'s routing zone, and thus, for instance, if node *A* requires a route to node *G*, it would have to initiate an on-demand route discovery for node *G*.

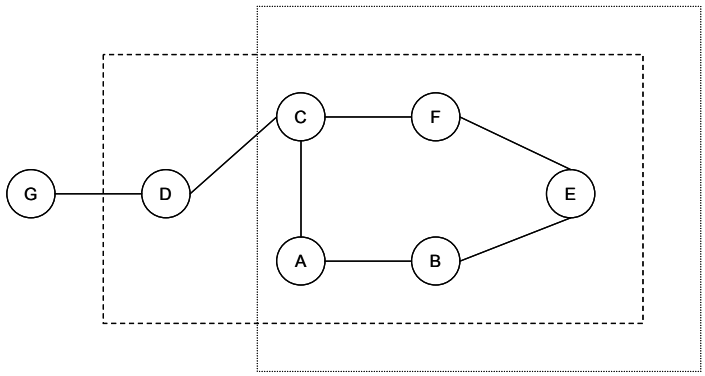

Figure D.11: ZRP - Node's *A* routing zone (2-hops zone radius)

The ZRP's global route discovery mechanism is reactive, and can be used by a node which requires routing information that is not immediately available in its routing table. In this case a node initiates a route query packet, on-demand. The query packet propagation used by ZRP is a modification of standard flooding, and is based on a special packet delivery service called *bordercasting* (Haas, Z. J., et. al., 2002b). This delivery service uses knowledge of local network topology to direct route queries away from the source. In more detail, the source initiates a query packet, which is uniquely identified by a combination of the source node's address and request number. The packet is then transmitted to a subset of the source's neighbours as determined by the bordercast algorithm. When a node receives a route query packet, it checks if the destination belongs to its routing zone, or, alternatively, if it has a valid route to the destination in its route cache. In the case that the node has such a route, it sends a route-reply back to the source. Otherwise, it transmits the query packet using the same approach as above.

Upon receipt of a route query packet, a node checks if the destination lies in its zone or if a valid route to it is available in its route cache. If this is true, a route-reply is sent back to the source. If not, the node broadcasts the query again. Overall, the operation of the IERP is sufficiently general so that many existing reactive protocols can be used as an IERP, with minimal modification.

D.17 Distributed Spanning Trees based routing protocol (DST)

DST is designed for mobile ad-hoc networks, and bases its functionality on a distributed algorithm which adapts to the topology by utilising spanning trees in the regions where the

topology is stable. It also uses an intelligent flooding-like approach in highly dynamic regions of the network. DST concentrates on networks where the topology can be highly dynamic in some regions, and stable in others, at least during certain periods. Routing is then performed by either *hold-and-forward* or a *shuttling* method, based on the information generated by the spanning trees. It also introduces a new concept called *connectivity-through-time*, in which a path exists both along the links, and, in time. Accordingly, a packet is held, or shuttled, along a tree, which achieves a compromise between packets delivery and the population of packets in the network at any given time. This scheme provides no guarantees for successful packet delivery to the destination, however, it creates a balance between packet traffic and connectivity requirement.

Overall, DST comprises a forest construction algorithm and a routing algorithm, which are both executed in parallel on each node within the network. The basic idea underlying the dynamic forest construction algorithm is the construction of a set of dynamic trees T_1, T_2, … , T_k , where each T_r, at a given time, has mobile hosts h_1, h_2, … , h_p such that any node h_i can communicate with a given host h_j. It is assumed that each mobile host h knows the IDs of its neighbouring nodes, and thus a node can communicate to one of its neighbours, directly. However, if a node wishes to send a message to a node which is beyond its wireless reachability, it would have to send its message via the tree edges.

In DST, each node h_i is required to maintain the following information: its own ID (h_i); its parent's ID ($p(h_i)$); the ID of the root tree node ($RootID(h_i)$), which h_i belongs to; and its childrens' IDs ($h_i.child(j)$ for the j^{th} child of h_i). A node within a tree can be in one of the following three states:

- **Router**. A root node, or an internal node, in this state follows the routing algorithm.
- **Merge**. A node $g \in T_i$ is in merge state when it comes in contact with a node $h \in T_j$, such as that g and h are in direct communication range, and a merge attempt to combine the trees T_i and T_j has been initiated. The joint tree structure is then re-aligned.
- **Configure**. A node in this state performs updates to its data structures when its parent or child(s) move beyond its communication range, or when a new node comes in contact with it.

In addition to theses, a bridge is defined as a connection between two nodes that are in direct communication range, but belong to two spanning trees. Bridges are formed to achieve connectivity between two distinct spanning trees likely to merge. However this condition is preferred in situations where neither of the two nodes forming the bridge is likely to move away, in the short term. Thus, by just forming a bridge, the heavy cost involved in re-

aligning the tree is avoided.

Routing of data packets is accomplished by forwarding the packets via the tree edges, to the most possible extent. A node h which receives a data packet for a destination *j*, forwards the packet on the spanning-tree based on one of the two following algorithms:

- **Hybrid-Tree-Flooding** (**HTF**). Data packets are sent to all possible neighbours in the tree including adjoining bridges in the spanning tree. In addition, packets are stored at each node for a period of time called *holding-time*, after which the packet is deleted. During *holding time*, if new bridges are being created at the node, then packets are sent along the bridges.
- **Distributed-Tree-Shuttling** (**DTS**). Packets are sent along the tree edges, starting from the source node. When a packet reaches a leaf node in the tree, it is sent back up the tree until a certain height is reached. This is called the *shuttling level*. The packets are then sent down to the tree again, or to adjoining bridges.

The rationale for the holding time in the HTF algorithm is that, as a network is becoming more stable and connected, it might be sensible to buffer data packets and route them as the network connectivity increases over time. In contrast to the HTF algorithm, DST requires smaller number of messages to accomplish its routing tasks.