AUTOMATING THE PROCESS OF NETWORK DOCUMENTATION

BRYAN EDWARD CAMPBELL

Submitted in partial fulfilment of the requirements of
Napier University
for the degree of
Master of Science in Advanced Computer Networking

School of Computing
June 2007

Authorship Declaration

I, Bryan Campbell, confirm that this dissertation and the work presented in it are my own achievement.

1. Where I have consulted the published work of others this is always clearly attributed.

2. Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work.

3. I have acknowledged all main sources of help.

4. If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself.

5. I have read and understand the penalties associated with plagiarism.

_____ _____

Signature Date

_____

Matriculation Number

Data Protection Declaration

Under the 1998 Data Protection Act we cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name against *one* of the options below to state your preference.

	The University may make this dissertation, with indicative grade, available to others.
	The University may make this dissertation available to others, but the grade may not be disclosed.
	The University may not make this dissertation available to others.

Abstract

Knowledge of network topologies is invaluable to system administrators regardless of the size of an enterprise. Yet this information is time consuming to collect, and even more so to be processed into easily consumable formats (i.e. visual maps). This is especially so when the culture within which administrators operate is more concerned with operational stability and continuity as deliverables rather than documentation and analysis. The time-cost of documentation impinges upon its own production. This continues to be the case although documentation is of increasing importance to non-technical personnel in enterprises, and as a compliment/supplement to network management systems.

This thesis puts forth a framework to largely automate the process of documenting network topologies. The framework is based on issues raised in recent research concerning the needs of IT administrators, and network discovery methods. An application is also described serving as a proof-of-concept for the central elements of the framework. This application was realized in the Microsoft Visual C# 2005 Express Edition programming environment using the C#.NET language. The compiled result is supported by the .NET Framework 2.0 runtime environment. The application provides for an administrator to control, through a graphical interface, the sequence of discovering a network and outputting visual documentation. For testing, Cisco Systems routers and switches, along with a Microsoft Windows-based laptop, were used to construct a mock network. Measurements of the performance of the application were recorded against the mock network in order to compare it to other methods of network discovery.

Central to the application's implementation is a recognition that networks are more likely than not to be heterogeneous. That is, they will be comprised of equipment from more than a one vendor. This assumption focused the choices about the framework design and concept implementation toward open standard technologies. Namely, SNMP was selected for discovery and data gathering. XML is utilized for data storage. Data processing and document production is handled by XSL.

Built around these technologies, the application successfully executed its design. It was able to query network devices and receive information from them about their configuration. It next stored that information in an XML document. Lastly, with no change to the source data, HTML and PDF documents were produced demonstrating details of the network. The work of this thesis finds that the open standard tools employed are both appropriate for, and capable of, automatically producing network documentation. Compared to some alternate tools, they are shown to be more capable in terms of speed, and more appropriate for learning about multiple layers of a network. The solution is also judged to be widely applicable to networks, and highly adaptable in the face of changing network environments.

The choices of tools for the implementation were all largely foreign to the author. Apart from the *prima face* achievements, programming skills were significantly stretched, understanding of SNMP architecture was improved, and the basics of these XML languages was gained: XSLT, XPath, and XSL-FO.

Table of Contents

List of Tables

List of Figures

List of Code Snippets

Acknowledgements

My first thanks goes to Professor Bill Buchanan, my supervisor for this work. I appreciate his encouraging the pursuit of the concepts presented herein, and allowing me the latitude, freedom, and time to discover and explore them along with the dissertation and research process itself.

Most students of networking are less interested in software programming, and I am no exception. The programming achieved in this work would have been impossible without the well-designed and well-presented module on programming from Lecturer Alistair Lawson.

Additional thanks to Dr. Ahmed Al-Dubai for acting as the internal examiner on this work.

Finally, I must relate that my interest in effective documenting of IT systems stems from my first IT job under the direction of Ms. Sally Weldon. Her mantra upon finishing a project was, "Did you document it?" That idea, so often overlooked, has stuck with me ever since.

Bryan Campbell
31 May 2007

Chapter 1 Introduction

1.1 Context

Documenting complex systems has two central problems: expensiveness and error. Expensiveness is expressed through a large investment of time needed to record details of a system. More is needed to process and present that information in usable formats. And even more is required to maintain the ongoing accuracy of the system details, and the presentation mediums, as complex systems change through time. At any point in this cycle of gathering data, processing data, presenting data, and maintaining data, error can be introduced. When error is injected at one step it is inherited by the subsequent steps.

Attempts to document computer networks are not immune to the factors of expensiveness and error. Computer networks, after all, are highly complex systems that incur many changes through time. And, although automation has been embraced in many areas of computer administration, it has yet to become practiced as a means for producing network documentation. The spreadsheet and the Visio diagram still reign as favourites of administrators (Barrett, et al., 2004, Nance, 2005, & Rainge, 2006).

The problem, of course, with spreadsheets and Visio diagram is that they are manually constructed. Time is invested in gathering information, typing it into cells or icon captions, and then going back to change it by hand when a change occurs – if it is remembered. What's more is that each document is likely to be an island of information. One spreadsheet will contain data about switches while another has all the information about routers. One Visio diagram will show a LANs physical topology, and another the IP topology. Some changes might require a visit to every separate document that is kept.

Describing the problem of expensiveness created by manual efforts at documentation production also describes a methodology where the other problem of error is naturally occurring. The overriding source of error is humans themselves. Humans cannot, and should not, be wholly removed from any area of administration, but

> [o]ne thing to consider is whether a human will be involved with producing a document. No matter how careful we are, we humans tend to make a lot of mistakes. Validation can find those problems and save frustration later. But software-created documents tend to be very predictable and probably never need to be validated (Ray, 2003, p. 113).

Automating the production of network documentation can do much to mitigate both time expensiveness and human error. Additionally, having up-to-date and accurate documentation about computer networks is increasingly important to enterprises. Many are already using network management systems, but they are also beginning to adopt better, standardized, management paradigms. Both call for documentation to ensure quality computer network administration.

1.2 Aim and Objectives

The aim of this thesis is to explore open standard technologies that can be combined in a system that automates the production of network documentation. Objectives supporting the aim are:

1. To conduct a critical review of existing literature relating to network management and topology discovery systems.
2. To design a novel system framework for producing network documentation.
3. To use a non-proprietary protocol to retrieve information from network devices.
4. To store network information in an accessible, portable format.
5. To automatically produce network documentation as both paged and non-paged media.

1.3 Background

The Internet is a non-proprietary system comprised of administrative domains. An administrative domain is proprietary in that a single entity is responsible for it. The contents of administrative domains are further network sub-structures ranging from the large to the small. Examples, respectively, include WANs and LANs. Efforts to document or map any of these levels of organization have to expect a lack of homogeneity.

Various elements of all computer networks operate within layers of the OSI model (Table 1). This is a conceptual framework of seven layers describing the operation of computer networks. Of most interest for building a functional network are the first three layers. Layer 1 is the Physical layer dealing with how devices interconnect. Layer 2 is the Data Link layer. In a LAN, switches operate here to construct the logical paths along which data frames will travel. Layer 3 is the Network layer. Routers function in this layer to direct IP packets. Administrators documenting a network document these layers, often by way of producing topological maps.

Layer	Name	Common Function	Example
7	Application	Application services	HTTP
6	Presentation	Data representation	ASCII
5	Session	Connection management	SSH
4	Transport	End-to-end connections	TCP
3	Network	Logical addressing	IP
2	Data Link	Physical addressing	Ethernet
1	Physical	Media and bit transmission	10Base-T

Table 1 – The OSI reference model

In order to produce documentation about a computer network, data must be gathered first. This process of network discovery might be accomplished in an automated manner using one of three classes of tools, being proprietary management protocols, open standard management protocols, or semantic manipulations. A proprietary protocol is usually not a good choice as it has a hard time being applied to heterogeneous environments. An open standard protocol will have more success as it is non-proprietary and can be implemented on any network device regardless of the

manufacturer. However, the protocol must be deployed to make it useful. The third type of tool, a semantic manipulation, functions by taking advantage of some feature common to the environment in order to discover network elements. The chief example is the ping tool. Its execution allows for node discovery and identity validation to be performed in one step. A semantic manipulation generally bypasses the heterogeneity and deployment problems.

A semantic manipulation is generally only applicable to one of the three OSI layers of interest. Both proprietary and open standard management protocols are more powerful in that they are usually capable of discovering information about many layers concerning a network device.

1.4 Thesis Structure

Chapter 2 provides further background, detailing the need for documentation and tools. It concentrates on information from relevant literature about documentation's relationship with administrators, and network management systems. Work by Barrett, Kandogan, Maglio, Haber, Takayama, and Prabaker (2004) is especially important here in establishing the contextual problems of time expensiveness and human error.

Chapter 3 reports on recent literature's discussion of the mechanics of topology discovery. Attention is given to system architecture, network types, and communication methods. To provide a broader discussion of communication methods, the paper *Discovering Internet Topology* by Siamwalla, Sharma, and Keshav (1998) is given notable analysis. Their work was highly informative in the conception of this thesis.

Chapter 4 proposes a framework designed to gather network device information; store the gathered information; and finally, output documents valuable to administrators. The framework flows from the problems, questions, and ideas of the previous chapters.

Chapter 5 describes a software application implemented based on the principals of the framework.

Chapter 6 begins with an evaluation of the implementation. It continues with an overall appraisal of the thesis' work, points to other interesting ideas, and presents final conclusions.

Chapter 2 Background

2.1 Introduction

Nance (n.d.) begins his paper, invitingly titled *Know Your Network*, by encapsulating why networks should be documented, and documented well:

> You can do anything you want with a well-designed, well-documented network. With relative ease, you can expand it, enhance it and troubleshoot it. You can plan its future and even reduce its costs.

Embedded in this thought are the goals of most networks:

- They should be easy to manage, and more importantly, easy to fix when faults occur.
- They should operate with quality because the enterprises that rely on them must do the same.

The people that keep networks well managed and operating with quality are the administrators. Thus the job of documenting networks falls to them. Their challenges are presented in Section 2.2. The significance of effective tools available to administrators is the subject of Section 2.3. Section 2.4 will discuss the cost of network management, and Section 2.5 looks at documentation's role in network management systems (NMS).

2.2 Work Environment of System Administrators

Sandwiched between IT developers/architects, and end users, is the system administrator. Situated here, sysadmins might be thought of as the "expert end user" – both end user and super-user of the very things they must keep running at all costs. They are from one perspective an end user in the sense that they receive output from the developers. Yet the true end users, relying on systems for their day-to-day functions, see sysadmins as the provider of a service. Sysadmins are the car mechanics of computer systems. A service provider, the mechanic must both know how to use a system the way an end user does, *and* know in an expert fashion how it operates. In this way sysadmins straddle and bridge various continuums.

Consideration of the conclusions reached by Barrett, et al. (2004) in a study of this job role leads here to a synthesis of several scales, or continuums, at the nexus of which is placed the sysadmin (Figure 1).

Each scale's two ends push and pull at the centre because both demand attention. Neglect of one end leads to disequilibrium. Sysadmins must balance between the IT sphere and the business sphere, where technology must be aligned with organizational goals. They act as a bridge between highly complex technical and social arenas, often literally translating jargon-filled tech-speak to layman's terms and creative analogies. Sysadmins seek equilibrium between desires of development and demands of

production[1]. Finally, they are tasked to manage risk, being sure that neither too much, nor too little planning and rehearsal occurs at the cost of actually implementing necessary change. The work performed by sysadmins is usually extremely complex. Typically more complex than the sum of the parts, because complexity emerges in impossibly incalculable ways when so many hosts, servers, routers, switches, databases, applications, and networks are connected together. Competing pressures in a multi-faceted environment result in work that is decidedly non-linear.

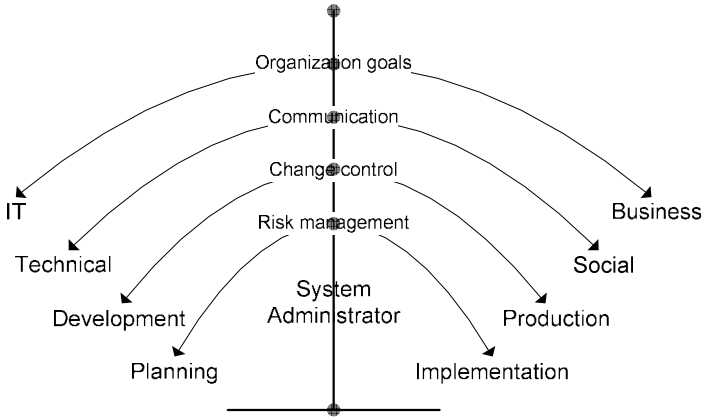


Figure 1 – Competing factors for the System Administrator

Along with being non-linear, work is of a "less formal style [for] sysadmins. The sysadmin work environment we saw was very different from that required for the design-develop-test cycle of software developers" (Barrett, et al., 2004, p. 394). As a comparison to another IT role, software developers rely on authoring environments where code is checked in and out. Their tools help manage and structure work, while also providing a framework for collaboration. Work content has the opportunity to be peer-reviewed, and is put through approval chains. Finally, it is sent to testing from where it may cycle back through the system, or be released for production. Tools that allow this (examples of which are Microsoft Visual SourceSafe or IBM's Rational suite) help to 'linearize' and set structures around the work of software developers (Figure 2). This characterization is not meant to say that software development is simplistic compared to system administration. Instead, the comparison attempts to reveal a contrast in the nature of the approach to the work performed.

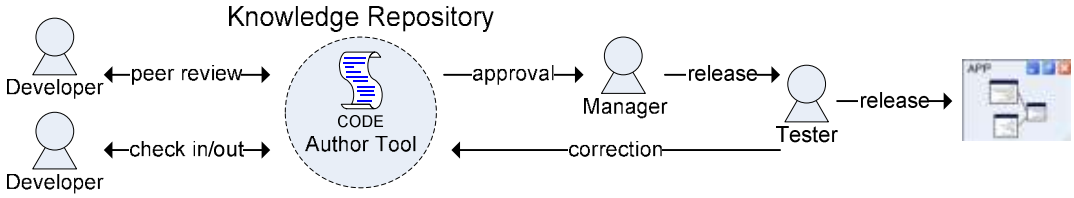


Figure 2 – Software Development centred on a structure-lending tool

2.3 Effective Tools

Meanwhile, sysadmins lack similar tools. Any tool intending to lend structure to the work of sysadmins will run up against the issues of scale of complexity, and variability between environments. Tool usage is obviously not alien to sysadmins,

[1] Enforcement of security policies is a contemporary example.

though. It is certainly not the case that they are doing everything manually. Automation is often achieved through an amalgam of purchased applications and self-written, function-specific tools and scripts. What is more is that the self-written tools contain an added value of trust (Barrett, et al., 2004). This is because the sysadmin themselves imbue a tool with a specific function to be performed in a specific way. Likely, this way will be exactly how the sysadmin would execute the action manually, but recorded in a scripted file. However, even highly trusted applications must be doubted by the conscientious sysadmin. To hypothesize: perhaps an indicator in a management or monitoring application stating "everything-is-OK" shows an old status, or its own underlying process quit running itself some time ago. Thus, most troubleshooting begins by verifying and/or reproducing the issue at hand. Ultimately it seems more difficult to fit one tool around the environment that sysadmins encounter as can be done for software developers. Conversely, what is observed is a myriad of small, bolt-on utilities each intended to accomplish a specific task for the sysadmin.

While ad hoc scripts and the like will always have a place in the arsenal employed by sysadmins, they present longer term challenges and risks, despite their being "trusted." First, programming principals are likely to be ignored as programming skills are not mandatory. Barrett, et al. (2004, p. 394) found only a third of their survey population of sysadmins had education in programming. In other cases a small tool or script is likely to go undocumented as to its purpose or function. The attitude is that "everyone knows about it," or that any *good* sysadmin will understand it. A parallel situation is that processes become dependent on a single tool's creator/owner. No recourse exists if that individual is on holiday, out sick, or permanently leaves the enterprise. On the other side of attrition, new employees have to be trained on, or possibly left to reverse-engineer for themselves, such solutions. This is a hidden (although very lightly so) cost of operations.

Therefore, knowledge must be gotten out of the heads of the experts and gurus and into documentation (Nance, 2005) (Figures 3 and 4). Because IT support functions provide a service, there is less expectation of tangible deliverables outside keeping everything functioning as it is supposed to. A car owner simply needs a mechanic to keep their car running. Likewise, it is often of little interest to the customer of the sysadmin the intricate details about how a system vital to them is repaired – just that it runs as expected. In comparison with developers again, we see that they create actual products: software applications. Documentation, inclusive of performance metrics and analysis, must become a deliverable product of system administrators. Their traditionally less formal work environment, though, provides a cultural impediment to such a transition. Additionally, the historical lack of effective tools bringing documentation into system monitoring and management processes, essential work of system administration, provides another significant barrier.

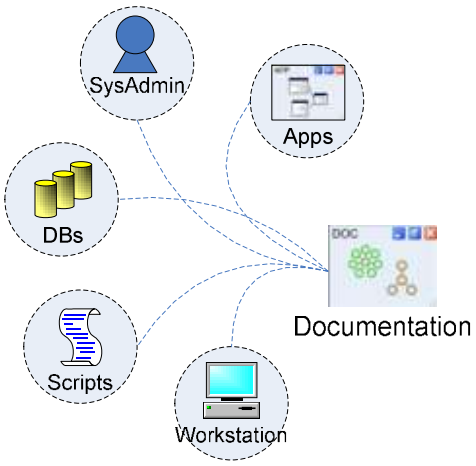


Figure 3 – Disjointed Knowledge Repositories of the System Administrator

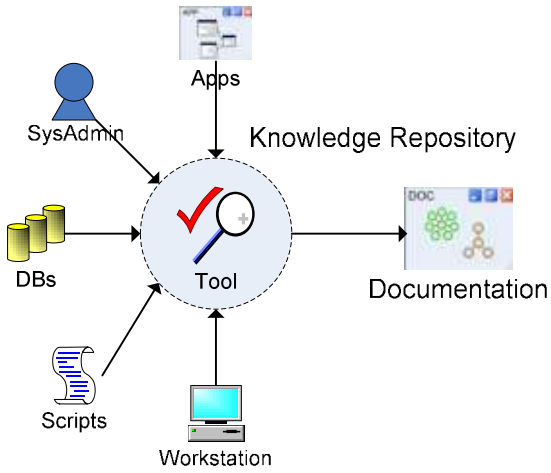


Figure 4 – Transferring Knowledge into a Management Tool

Despite the necessary role that system administrators provide to enterprises as the "expert end user," there is perhaps an assumption they are just that: a special case of the end user. Subsequently, Barrett, et al. (2004), report that "little can be found in the literature about the practices and problems of these highly specialized computer users" and that "[d]espite the importance of sysadmins, few HCI [Human-Computer Interaction] studies report on their particular problems and practices" (p. 388). That team's field studies of tools and practices of sysadmins helps fill the observed paucity. One of their main conclusions is that in relation to the sysadmin, tools are often lacking correct information, are malfunctioning, or are not aligned with the observed work practices. Sysadmins themselves reported (albeit in a highly informal survey) that one of the most pleasing cases was to find a useful tool, while one of their self-criticisms was a desire to "plan, manage, communicate, document, or organize better" (Dijker, 1998, p. 18). Thus a useful tool that is designed with the sysadmin in mind is likely to be welcomed by the community.

A telling case is related by Barrett, et al. (2004) from their field observations of sysadmins *in situ*. In the instance described, a sysadmin spent many hours attempting to achieve communication between two servers through a firewall. While troubleshooting, he interfaced with eight other persons via direct meetings, phone, email, and instant messaging. All these individuals were dependent upon the sysadmin as their source of information about the systems involved. Ultimately the situation was resolved when a colleague was able to independently analyze and discover the root cause – one seen, but misinterpreted initially by the sysadmin, and then propagated to the eight others. He "managed this complexity by rapidly moving among multiple management tools and working together with many experts, but there was no single view of the entire system. A simple drawing of the configuration might have made the situation clear and avoided hours of troubleshooting" (Barrett, et al., 2004, p. 393). Documentation could have prevented the case from living as long as it did. Automated documentation may also have proved less immune to the human error. Any sysadmin stuck working a critical issue for hours would welcome tools that help avoid such cases.

Without question, there are numerous tools that are very advanced and effective at assisting the sysadmin with monitoring and maintenance. It is less certain that there are solutions able to assist equally as well with the aspect of documentation. Researchers (see Barrett, et al., 2004) and industry analysts (see Nance, 2005, and Rainge, 2006) all point out early in their presentations that the use of spreadsheets to store and organize critical data is a ubiquitous practice despite the manual effort involved in creating, maintaining, and interpreting them. The automation of documentation helps to address the several points raised. It fights the issue of keeping knowledge in workers heads, or on their own workstations, where it is unshared and not auditable. It eases attrition events. Its presence and usage helps to formalize the environment by providing common reference for fault management, and enables common processes. When demonstrably accurate, tools that provide for the automation of documentation can become trusted. A well-trusted tool will negate, to some degree, the need for ad hoc tools with their attendant long term issues. Ultimately, good documentation balances the scales by allowing transparent and translated communication between IT and business, technical and social vocabularies, development and production, the expert and the non-expert.

2.4 Cost of Network Management

The management of IT networks is expensive and potentially daunting in scale. Integrating automated documentation into management tools and processes has the potential to lower the costs and tackle problems of scale.

Administration of systems is the most resource-consuming expense of operating IT systems [see Barrett, et al., (2004), IBM, (2001), Kephart, (2003), McDonough, (2003), and Patterson, et al., (2002)]. Some studies and estimations put administration at or over 50 per cent of the total cost of ownership (TCO) of IT systems. Furthermore, costs are exacerbated beyond what is simply necessary to run and support systems due to the absence of, or ignoring of, policies and processes. In a study spanning 29 months of work orders of a university IT support team, it was found that fully one third of man hours spent correcting problems could be attributed to an initial cause that failed to comply with established policies (Madigan, Petrulich, and Motuk, 2004). As discussed in the previous sections, documentation exists to cut down on costs by cutting down the time needed to train employees, maintain systems, and recover from failures. But, manually produced documentation does little to help efficiency efforts in these areas.

In many IT systems, especially large and complex ones, change occurs too fast for manually maintained documents to keep up. They are not accurate compared to the actual states of systems. Change is so quick that efforts at visualization (i.e. maps), asset management and tracking, and task automation become a challenge and struggle. A report on interviews conducted with IT executives already employing a tool for network visualization and diagramming suggests automation can help to lower TCO while supporting administration activities. One comment gathered was that using sets of "traditional map drawing tools, the result 'is almost outdated the day that you finish it'" (Rainge, 2006, p. 12). Furthermore, all too often administrators are called to the next urgent project before they document what was done in the previous one. On the other hand, automation and integration of documentation into daily activities of system maintenance and management provides for self-reinforcing,

synergistic, practices. If a tool used for administration helps create the documentation that is relied upon for reporting, problem solving, and fault recovery, an imperative exists to use the tool. Documentation can even *be* the tool. Changes to a system can cause a change in the tool/documentation. Changes in the tool/documentation can cause a change in the system. However accomplished, automation provides a vector for quick and accurate information capture.

Mentioned already is the idea that work cultures are a contributing factor in resisting change in general. Implementing new methodologies and tools is a form of change and is susceptible to the same resistance. A study of an IT organization beginning to address the area of systems change management confirms this aspect. It is observed that:

> It was difficult to get people to change their habits to plan ahead far enough to submit change requests in enough time for them to be approved and reviewed. Changes were often implemented regardless of the status of the request. Sharing the outcome of a change, even those completed successfully within the timeframe allocated, was also a new procedure to instill [*sic*] (Dietel, 2004, p. 190).

Still, working in IT means usually that the only constant is change. Standardizing and formalizing practices is increasingly a necessity and a demand both by entities internal and external to an enterprise.

2.5 Documentation's Function in Management Systems

Internally, IT teams need to manage not just systems, but wrap their whole operations up in change, configuration, and fault management paradigms. Each of these three areas certainly overlaps with one another. It is possible to even cover all of them with a powerful application (Figure 5). There are subtle differences, though, and documentation adds value to each in different ways.

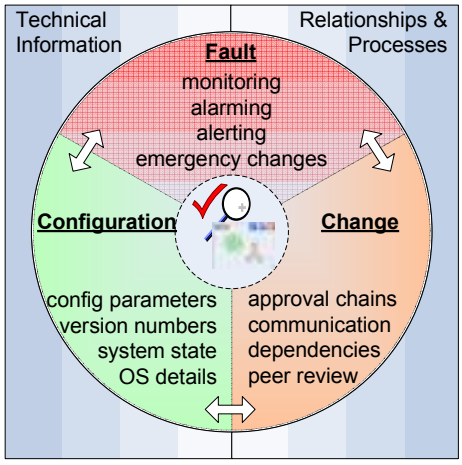


Figure 5 – A Model of Management Systems featuring Tools & Documentation at the Centre

Change management involves defining a process to be followed for executing changes, documenting system inter-dependencies, creating communication channels between involved teams/parties/user groups, and continual process improvement.

This piece of the management spectrum is able to do the most to lend structure to the work environment. It creates peer review, and supervisory approval chains. Interdependencies are identified. Change management systems ask that implementations be planned and contain back-out strategies. Planning causes a need to know what the "now" and "later" pictures of the IT landscape look like. A change system fosters that all of these details be input or captured in a tool. When that is done, documentation exists so that responsibility domains are delineated and work performed is auditable. Change management is probably the least concerned with technical details of systems. The most important results are the construction of communication pathways and establishment of common processes.

Configuration management handshakes with change management, but deals with the technical side of a change activity. Documentation needed here shows details of systems' overall configuration states, specific variable values, and version numbers of firmware, files, or operating environment. Strung together, historical snapshots of system states can demonstrate change through time. Documentation from change and configuration tracking systems becomes very valuable for fault management.

Fault management is usually the first type of management systems that are setup by and for IT teams. This is because it encompasses monitoring, alerting, and alarming. These activities are essential so that IT workers know about problems before the enterprise, or worse external customers, detects them. They are just the first phase though. When email, pagers, or mobile phones alert administrators of an error condition the fault management process has just begun. To be completed, the fault must be corrected. Documentation from change and configuration management is relied upon here. Invaluable information is present in the change history and in the heretofore stable system state about conditions before a fault. That data allows quicker identification of what has changed. It is more straightforward to return systems to a previously known, correctly running state. The alternative is to perform troubleshooting in an undisciplined manner, possibly injecting new problems into an already malfunctioning system. Full resolution of a fault must also include its own documentation. Usually post-mortem reports and meetings provide analysis and root cause elucidation. The whole process also plugs back into change and configuration management making information capture a wheel that does not to be reinvented. Faults can thus be handled as special types of changes (i.e. emergency/unplanned changes). It should not be left unsaid that the sum of documentation from systems management is highly valuable for business continuity/disaster recovery strategies. In fact, it may be absolutely necessary in situations where facilities and hardware are physically destroyed.

Fortunately automation already plays a large role in most systems management tools. But, as pointed out while discussing administrators, integration of documentation is more tenuous. Especially without this integration, policies need to be in place to align practices with organizational goals. In regards to security, we now instinctively recognize the veracity that failure of polices to be extant, be followed, or have consequences is deleterious (Madigan, et al., 2004). Perhaps this is so due to the nearly immediate negative effects of security breaches. Likewise, lack of and failure of policies to guide change, configuration, and fault management, as well as produce documentation for computer networks, even in the presence of automation, translates to higher operating costs for the enterprise.

While this section limits itself to a few elements of systems management, documentation plays an increasingly vital role in larger NMS's. The ISO OSI model itself contains a Network Management Model which applied the FCAPS paradigm[2] (ISO, 1989) to data networks (Parker, 2005). FCAPS does little more than delineate functional areas of management activities. It does not mention the role of documentation in effective IT systems management, but the FCAPS elements are building blocks to more complex management frameworks. Generally, today's management frameworks and standards are described as IT Service Management systems. ITSM systems are usually bound in their scope to an enterprise's back-office infrastructure. Two of the most popular ITSM systems today are the Information Technology Infrastructure Library (ITIL) and the ISO 20000 standard. The ISO 20000 standard itself aims to be an ITSM standard which aligns itself with components of the ITIL.

The ITIL is much larger in scope than a back-office-centric ITSM. It is a set of best practices divided into eight conceptual disciplines. One, the Service Management discipline, seems to map onto most of the scope of an ITSM. Its Service Support sub-discipline contains a chain of processes (Incident, Problem, Change, Release, and Configuration Management) that all feed a Configuration Management Database (CMDB). The tool or application found at the centre of Figure 5 could be thought of as a potential CMDB for recording the process chain and outputting documents for Quality Management. A second ITIL discipline covering areas of general ITSM is the Information and Communication Technology (ICT) Infrastructure Management discipline. Its sub-disciplines are Design and Planning, Deployment, Operations, and Technical Support. To support these there is the Operational Documentation Library (ODL). The ODL, too, is a repository of information about IT systems, and a platform for operational and technical support personnel to document and share their expertise (OGC, 2002).

2.6 Conclusion

System administrators work in a highly technical environment, but cannot ignore their position at a boundary with the non-technical world to which they provide a service. Documentation helps to bridge the communication gap between these areas. System administrators also face a diverse environment where a diverse set of tools is employed to manage systems. Effective tools can structure operations and potentially lower TCO. Moreover, tools can integrate with processes and procedures vital to NMS's. NMS's themselves call for documentation of systems, processes, and procedures in order to achieve quality. As much as management activities are a function of the system administrator role, so too is the production of documentation.

[2] FCAPS is a popular acronym for Fault, Configuration, Accounting, Performance, and Security – five key areas of systems management.

Chapter 3 Literature Review

3.1 Introduction

Network administrators need maps of their networks. This type of documentation serves to communicate information about devices and their relationships. Trying to relate such data through text is tedious and counter-productive to speedy understanding. Prose or tables are simply less conducive to rapid consumption by humans. Intuition guides us instead to visualize information, especially when it represents large sets and multiple dimensions[3]. Thus network administrators produce maps both for their own reference and for use as a transmission medium to other entities. Moreover, businesses and researchers want to understand topological information for the purposes of evaluating performance of protocols; assessing security techniques; and provisioning and capacity planning (Alderson, Li, Willinger, & Doyle, 2005). Automating the visual representation of networks requires data concerning the objects to be represented. A choice is then to be made about how that data is collected: manually or automatically. A visualization tool could act on manually gathered data stored in databases and spreadsheets. In this situation, people are still involved, along with their typographical and misreading errors, in populating data fields and spreadsheet cells. Choosing this method does not really alleviate the long-term cost of generating documentation, though. CA's netViz is a tool that depends on these types of sources (netViz, 2003). Although demonstrated to be very powerful and cost saving (Rainge, 2006), it has no discovery engine[4]. Solutions reliant upon databases and spreadsheets will only ever be as accurate as their sources. The sources with the most obvious authority are devices/network nodes themselves.

Tools whose aim is to visualize network data and to collect that data from the environment directly, must answer several design questions. How will they be architected? What is the intended scope of the tool? What will be the mechanisms by which data is collected? Depending on those mechanisms, what are the limitations and challenges? How will the tool measure and present success? Section 3.2 looks at the classes of topology discovery systems. Next examined are the target scopes in Section 3.3. Mechanisms and protocols are discussed in Section 3.4. An extended analysis of protocols is given in Section 3.5 by way of a detailed review of the paper *Discovering Internet Topology* by Siamwalla, et al. (1998). Lastly, Section 3.6 presents some measures by which topology discovery can be evaluated.

---

[3] See Han (2005), for a technical description of a multi-touch screen device whose interface is predicated upon the intuitiveness automatically realised out of a highly visual and tactile presentation. It is seemingly so intuitive that Han himself describes it as "interface-free" in live presentations such as one given at the TED 2006 conference (offered at http://www.ted.com/index.php/speakers/view/id/65). Technology such as this is a very powerful demonstration of the ease with which, and affinity humans have, to interact with data visually.

[4] CA's netViz is now a suite of eight products per descriptions at http://www.netviz.com. The historically core product enabling diagramming, netViz Pro/3D, does not contain a discovery engine. The suite has gained this ability through a bolt-on offering from Concord. CA acquired Concord which previously acquired netViz.

3.2 Types of Discovery Systems

The discovery system itself can consist of a single agent, a network of agents, or a distributed population of agents reporting back to a central system (Figure 6). Four factors will be used to discuss these types: redundancy of data; potential to mimic an attack; resource constraints; and applicability to networks of different scales.

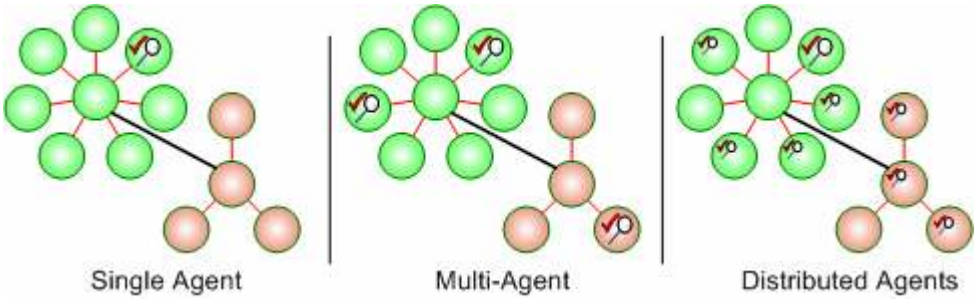


Figure 6 – How Agents are situated in a topology in different Types of Discovery Systems

3.2.1 Four Factors for Analyzing Discovery Systems

Redundancy

Redundancy does not refer in this context to an ability to remain operational in a fault condition. It is a measurable value of how often a discovery system repeats work by rediscovering or revisiting already seen nodes in a network. Admittedly, revisiting nodes is desirable for polling or verifying existing data. However, the discovery process itself becomes more expensive when it begins visiting nodes more than once. Donnet, Raoult, Friedman, and Crovella (2005), utilize the term monitor equivalent to a discovery agent. They go on to define and examine two types of redundancy. The first is intra-monitor redundancy. This term describes the fact that nodes will be visited or probed by a single agent multiple times. It is shown that nodes near a monitor will be visited many times. This measurement gradually declines as a node is further away. The second type of redundancy identified is inter-monitor. This term refers to the measurement of how many monitors in a multi-agent system visit the same node. It is shown that the set of nodes with the highest values of this measurement are neither close nor far, but those at an intermediate distance. Redundancy in general introduces questions about how to perform discovery in a way that is network-friendly. Minimalization of redundancy values leads to a less expensive discovery process that requires lower utilization of network resources.

Attack Mimicking

Somewhat correlated to the 'expensiveness' of a discovery session is the potential for that session to be perceived as an attack by nodes being probed. High utilization of available bandwidth might appear as a flood attack. Use of ping might look like some Denial of Service (DoS) attacks. "Whether or not it triggers alarms, it clearly is not desirable for a measurement system to consume undue network resources" (Donnet, et al., 2005, p. 328). Such situations become more likely when redundancy is not considered. A single agent may poll a node too frequently. Multiple agents may poll the same node simultaneously, mimicking a Distributed DoS (DDoS) situation. The likelihood of attack-mimicking relates to the allocation of available resources.

Resources
The architecture of a discovery system will determine its sum pool of resources. Resources can be processing power, aggregate network bandwidth, available memory, etc. In order to avoid mimicking attacks, and to remain friendly to the network, the use of resources may have to be metered. This will usually be a trade off between a resource expense and a time-factor expense.

Applicability of Scale
The resource pool boundaries make different systems applicable to different scales. It would be overkill, for example, to use a multi-agent or a distributed system to discover all the nodes on a LAN. By the same token, a single agent would be hard-pressed to discover the topology of the Internet by itself.

3.2.2 Single Agent Systems

A discovery system comprised of just one agent is mostly static along the four characteristics. For redundancy, it will always suffer from some degree of the intra-monitor type. This is in part because it has no external input to notify it not to pursue probing down a known path, but mainly because all activity originates from a single point. Any choice to not probe further down known paths can lead to an anaemic tree-graph, with missing branches or far away leaves. On the other hand, a single agent system does not suffer at all from inter-monitor redundancy. It is inescapable, though, that nearby nodes will be subjected to a larger share of traffic generated by the discovery process. The analysis by Donnet, et al. (2005) found all monitors displaying similar intra-monitor redundancy patterns, with nodes closest experiencing the most visitations. This redundancy might be reduced by controlling the frequency at which probing occurs.

Potential of mimicking an attack is restricted by the single agent's resources. Again, control of a discovery tool's activity and its consumption of resources will curtail this potential, although it is likely low to begin with. The single agent may have an advantage in this respect in that it is easily identifiable by administrators as a trusted source. Minimal overhead is encountered if the probe source were allowed special access to networks, or through firewalls. Doing so obviates any misinterpretation that an attack is occurring. Such a practice would quickly encounter problems of scale in a multi-agent discovery system, though.

Resources available are limited by the agent's host server. In fact they are static barring adding hardware or higher capacity network connectivity. A tool will have to work within the static constraints offered by the server. Mainly the limit on resources causes the single agent solution to be most appropriate to smaller networks. Applicability to medium or large networks is certainly possible, but involves a trade off with time and perhaps completeness. Larger environments will undoubtedly contain paths unavailable to a single agent since it sees the tree-like nature of a network from a static vantage point.

3.2.3 Multiple Agent Systems

A multi-agent solution inherits some of the properties of its constituent single agents, as they act in concert. Each individual agent faces intra-monitor redundancy,

certainly. The whole system also faces inter-monitor redundancy. Since each agent may probe the same area of a network, work is duplicated, expensiveness increases, and efficiency decreases. Costs can be kept down by allowing the agents to update each other with their findings, or assigning discreet jobs to each one (Donnet, et al., 2005).

The risk of appearing as an attack is increased because there are more agents involved in discovery. The agent population may still be small enough to set trusts and permissions in a network or in firewalls, but this quickly becomes infeasible. It is expected, after all, that each agent will likely be resident on different networks to capture a clearer picture of the topology. If solutions are used that help reduce the inter-monitor redundancy, the likelihood of false alarm DDoS attacks is lessened since the system would attempt to avoid multiple monitors hitting the same node. Attack mimicking may not even be a real consideration given that "the present generation of systems operates on largely dedicated hosts, numbering between 20 and 200" (Donnet, et al., 2005, p. 327). It actually might be difficult for such sets to mimic a DDoS attack given that today's malicious instances employ zombie botnets of tens of thousands of hosts (Ratliff, 2005).

The resources available to the multi-agent system are, like the single agent system, relatively static, yet able to be increased linearly. Flex in the sum total of resources exists since additional servers can be added to (or subtracted from) the pool in a modular fashion. In this way, the multi-agent solution is extendable and might be adapted to various tasks, both small and large. Applicability to small networks seems like overkill, though, seeing that the original impetus for discovery agents running in parallel was to tackle the job of mapping the largest network of all: the Internet. Thusly, projects such as CAIDA's Skitter (Huffaker, Plummer, Moore, & Claffy, 2002) and Route Views (University of Oregon, 2005), using 25 or less monitors, have demonstrated the ability to map medium and large spaces. Although, results of newer distributed discovery systems contest the completeness of such attempts.

3.2.4 Distributed Systems

Description of Distributed Discovery Systems

Distributed discovery systems potentially have no limits against the factors under consideration. First, it should be presented what constitutes and characterizes a distributed and decentralized discovery architecture. One example is the DIMES project (Shavitt & Shir, 2005). This project allows any computer to become a discovery agent by running a very small piece of software. Often the software takes advantage of free processor cycles, activating along with a screensaver. Alternatively, the software runs as a background process that limits itself to the host system's resources (processor, memory, network bandwidth). The parcelling out of jobs/work in this fashion was highly popularized by the SETI@Home project (University of California, Berkeley, 2006) that allows any computer to help analyze data in the hunt for intelligent radio signals reaching the Earth. Data collected, or results computed, from each agent are reported back to a central system.

Generally, there are two methods of work distribution in such systems. In one, every agent can run the same algorithm. The second option is that the central system

assigns jobs to each agent. The advantage of the first is that there is very little overhead to manage the agents. The trade-off is that the central system will likely have to spend significant time dealing with duplicate information. This is the inter-monitor redundancy. In any multi-agent discovery system, more than one agent might discover the same node. The central system will have to try to correlate duplicate reports so that the node does not appear more than once in the topology. New agents may even continue to report already-found nodes. Thus, the issue is one that is ongoing for the life of the operation. The job assignment scenario shifts the trade-off putting significant time commitment into deciding what each client should be doing and communicating with them. Redundancy might then be avoided in a calculated way. Agents in the same local area of a topology might each be told to concentrate on different branches, or ignore nodes within a specific hop-count radius, for instance.

The DIMES project began in 2004, and added 5,000 agents in its first year. Almost two years into operations, nearly 11,000 agents were reported (DIMES, 2006). As a point of comparison and another example of the scale distributed systems can reach, the SETI@home project was reported to have gained 300,000 computers in its first week (Whitehouse, 1999, May 25). It now has over 1,000,000 hosts (BOINC Combined Statistics, 2006). These statistics help show the large populations capable of being recruited.

Analysis of Distributed Discovery Systems

Turning back to the four characteristics, distributed architectures easily outstrip single and multi-agent systems. Occurrence of redundancy, especially the inter-monitor type, becomes of special concern when the number of agents is so high. The Donnet, et al. (2005) analysis of CAIDA Skitter data, where there are only 24 agents, found that interfaces 4 to 14 hops away were visited by half (12) of the agents, as a median. This means that of all interfaces catalogued by Skitter, of those between 4 and 14 hops away, at least 50% were visited by half the monitors. Given that the population of interfaces at each of these hop distances all numbered between 10^3 and 10^5 it is easy to see how fast the redundancy issue grows. Using the conservative value of 10^3, 5,000 interfaces were hit by half the monitors. The size of most of the populations was on the order of 10^4, though, which causes that figure to jump to 50,000. In the centre of the heavily-visited populations, interfaces at 8 to 11 hops actually had a median inter-monitor redundancy value of 24—all monitors in the system. If these levels of redundancy can be reached by only a small population of discovery agents, the potential levels achievable by thousands of agents could be seen to rise to non-network-friendly, or even abusive heights.

The sheer number of agents possible and actually participating in systems today, makes very real the threat of attack mimicking. The emergence of distributed discovery systems is stated to be the motivator for the analysis of redundancy:

> Unless carefully controlled, these new systems have the potential to impose a heavy load on parts of the network that are being measured. They also have the potential to raise alarms, as their traffic can easily resemble a distributed denial of service (DDoS) attack (Donnet, et al., 2005, p. 327).

This will especially be the case if agents are clones of one another. That is, if every agent performs the same work, clusters of them might spark regional false alarm attacks. System designers can hope to avoid offensive results by better coordinating the actions of their agents. SETI@home, along with the myriad of other distributed computing projects based on the same client software, already provides a model for parcelling specific jobs amongst the agent set.

The resources available to a distributed system are limited only by the number of clients it is able to recruit. Statistics presented above already show how large some agent sets can become. Some of the real power realised from distributed systems is their processing power. Indeed, CPU cycles are what agent owners are donating to help crunch through large data sets. Again looking at the SETI@home project, it was calculated recently to have a processing rate of 149.8 teraFLOPS[5] (Anderson & Fedak, 2006). Other non-verifiable sources state its power is as high as 250 teraFLOPS. Of course, any host is really lending a bit of all its other resources, too: memory, disk space, and network throughput. The same study by Anderson and Fedak showed an average throughput of 289 Kbps made available by a SETI@home host. Multiplied by the number of clients that current projects boast distributed computing need not worry over lack of resources.

Obviously, a distributed topological discovery system is most applicable to large scale networks, principally the Internet. Results indicate that they have an advantage over multi-agent systems in their ability to discover more topological information. After only its first year of operations, DIMES found "about 61000 AS edges connecting over 15000 AS nodes. Out of these 61000 edges, almost half are edges that are not present in main BGP tables repositories such as the RouteView project, making the Internet 50% denser than previously thought" (Shavitt & Shir, 2005, p 74). A possible conclusion is that the randomness of location of each agent allows viewing of previously unseen paths, versus the non-random placement of dedicated servers. The magnitude of the operation contributes as well. This fact is demonstrated in the data by Shavitt and Shir (2005), and helps show that as time progresses, existing and new agents continue to discover novel elements, in this case Autonomous System nodes and edges.

3.3 Types of Networks

A given topology discovery system will not be applicable to all scales and types of networks. Design choices will influence and constrain each other. Major characteristics of a system will be:

- The type of network(s) to which it is to be applied.
- What type of map it aims to output. Physical and logical maps are examples.
- The mechanism for collecting network information. Route tables or the ping tool are examples.
- How the tool deals with 'node dynamics.' That is, it addresses the fact that information about nodes can change over time.
- The degree of 'completeness' that is achievable. Put plainly, the portion of network elements discovered versus the total.

[5] FLOPS – Floating Point Operations Per Second

These five elements are interrelated such that there are inherent trade-offs amongst them. There are very many characteristics of networks, links, and nodes that might be used in measuring network structures. The necessity to choose some, while not others, affects the nature of how well the result represents the actual measured environment (Alderson, et al., 2005). This issue becomes more conspicuous in larger networks. The larger potential item set presents a proportionally greater need to select a definite key characteristic. A system with the mission to map the Internet would not want to concern itself with end node hosts (a volatile and overwhelmingly large set), but with a less fine-grained element (such as AS boundaries) in order to achieve a representation of the structure upon which hosts rely.

Quickly, the fourth characteristic, node dynamics, should be illustrated. This idea is synthesized from several sources. One aspect of node dynamics is "a process known as *alias resolution*" (Spring, Dontcheva, Rodrig, & Wetherall, 2004, cited by Alderson, et al., 2005, p. 1214) which recognizes the issue of correlating multiple names to a single node. Names could be IP addresses, DNS names, etc. Another aspect of node dynamics is identity volatility. This aspect recognizes that a node's names or aliases can change with time. Volatility in a network is a strong theme of the analysis of Siamwalla, et al. (1998), although it is not extended all the way to the level of a single node.

Concerning the first characteristic of types of networks, reviewed literature indicates three major divisions of interest. Unsurprisingly, they are the Internet, WANs, and LANs. Each will be discussed in terms of the other four characteristics.

3.3.1 Internet

Presented in Section 3.2.4 were several examples of systems attempting to map the Internet. They all aimed to do so by discovering AS nodes and edges. Considering there are 64,511 possible publicly routable AS numbers (Hawkinson & Bates, 1996), and, theoretically, a limitless number of interconnections between them, discovering and mapping their relationships is exceedingly a problem of scale. Also a problem is the fact that Autonomous Systems are exactly that—autonomous. Each is under its own administrative control where the only responsibility is for the AS to advertise how to get traffic in and out. It is counter to the mission of administrators, especially in light of current security and governmental requirements, to share more information than necessary for operation (Alderson, et al., 2005). Scale and lack of transparency are the root issues driving how systems attempt to discover and map the Internet.

Production of a physical map of the Internet is difficult given the number of elements involved and the discouraging effect that security has on openly sharing topological information about AS's. Logical maps are still viable, though. Discovery systems revealed in the literature focus on the IP (Layer 3) level, and attempt to map AS boundaries and router interfaces. This is only achievable by reverse-engineering data gathered from measurement techniques.

Interested parties must "discover" how the Internet is hung together as there is no central authority able to query Internet devices for information, nor an incentive for the distributed authorities to share topological information. The mechanisms to do so,

therefore, are semantic manipulations of what is allowed to be known (in fact, must be allowed) in order for Internet communications to function. Traceroute is one such tool that reveals hop-by-hop routes. The ping tool is still valuable, although to a lesser and declining extent. Both will be discussed more in Section 3.4. A primary method of discovering Internet topologies is the action of reading or reconstructing route tables. This method provides a picture of how a given node (a router or an AS) is connected to, or reaches other nodes. The RouteView project is an example of an effort to map the Internet at the AS level, whereby BGP tables are collected, which is stated by Shavitt and Shir (2005). Their competing system, DIMES, employs traceroute as the means to explore routing paths and interfaces. Donnet, et al. (2005) introduce the Doubletree algorithm – a traceroute-like tool that attempts to improve upon simple traceroute by minimizing discovery costs. Existing research is full of examples like these three where ping, traceroute, and route tables are used, combined, modified, or employed in novel algorithms.

Regardless of the tool, Internet topology discovery will be largely confined to gathering Layer 3 information due to the Internet's own decentralized nature along with security and proprietary concerns of its constituent parts. Discovery mechanisms will therefore operate mostly at this layer and reverse-engineering will be relied upon to deduce the actual Internet.

A discovery tool applied to the target scope of the Internet will encounter node dynamics mostly of the alias resolution sub-type. Discovered nodes will usually be routers. This node type in this domain can be expected to be relatively static – that is, ISPs and AS's will be loathe to make frequent changes to devices providing backbone or network-to-network services. Such devices, however, are likely to possess multiple interfaces, each participating on disparate networks. How is it to be determined that two IP addresses, for instance, represent a single router/node? Determining this answer will be especially a challenge for the case of traceroute being used as the discovery mechanism. Its resultant data is comprised of nothing but IP addresses. Coupling data from a higher layer (DNS is an oft used source) can help with resolving router aliases, but the selected source will always import its own incompleteness, inaccuracies, or need for interpretation. Should the target node type be AS's instead of routers the issue of node dynamics changes from examining IP's to AS numbers. Likely, it is of a smaller scale as there is a smaller population of AS's and numbers than routers and IP's. Node dynamics is therefore seen to be related to the target OSI layer, node type, and discovery mechanism.

Trying to map the complete Internet is a problem of scale sufficiently tempting to attract much research on the topic. It is highly likely given the sheer number of elements comprising the Internet that complete topologies are impossible to discover. Furthermore, the largely proprietary and decentralized nature of the Internet guarantees a knowledge boundary. So, even if a topology were to be complete, how is it then to be verifiable also? All of this does not mean to say that it is not possible to produce highly accurate, although incomplete, maps of the Internet.

3.3.2 WANs

Wide Area Networks present a hybrid of challenges if a discovery system is to attempt to gain information about them. Constraints arise from the point of view

relative to the target network. First, one could be completely external to the target. This scenario converges with the discussion above about the Internet as the target where one is in a position of little power, only allowed to deduce information from the required information exchange that allows the network to function. Greater power to know a WAN topology exists from perspectives inside the WAN. Here, there are two significant perspectives. One is that of a customer's and the other that of the service provider. An organization might build its own WAN infrastructure, although doing so is expensive and rare compared to paying for WAN links from a service provider. Thusly, even inside a WAN there are knowledge boundaries.

One such boundary exits at the physical layer. A customer can only represent a WAN link as a service provider cloud in between their own nodes. This is even true logically as a customer only sees end points of any circuit through the cloud, as in Frame Relay, ATM, or MPLS services. Conversely, the service provider is fully empowered to graph physical and logical layers of their own service delivery network. But, they too would be limited to only knowledge of the customer's interface device and not beyond. For either perspective it should be recognized that even inside an Autonomous System there is likely to be divisions of administrative access and responsibility that would impede physical or logical discovery. As such, the statement that "[a]n administrator knows their entire network topology in advance, and can freely choose where to place their monitors" (Donnet, et al., 2005) may not hold. So, both topology types can be mapped while the success of doing so will be dependent on the point of view and access rights.

Now that access, and even administrative access, can be considered, additional means become available to gather information about a target network. Instead of gathering data and inferring the network structure from it, nodes can be queried directly. Ping and traceroute fall away as favourite tools. They are replaced by proprietary methods for communicating with network devices directly, or by the most notable open standard for doing so, Simple Network Management Protocol. SNMP will be discussed in Section 3.4.1.

Like the Internet, challenges of node dynamics are likely to be of the alias resolution type. A topology must continue to avoid representing a node multiple times because it has multiple identities. The other type of node dynamics, identity volatility, is not fully muted, but WAN devices and architectures are designed to be more static than dynamic.

Discovering a complete WAN topology is no longer a problem of scale even though it may contain many elements. The primary obstacle for this network scope is now the themes mentioned above: perspective relative to the target, and access, administrative or otherwise.

3.3.3 LANs

Knowledge of LAN topologies is as important to network managers as WAN or Internet structure. But research has mostly focused on the larger domains, and indeed on the higher OSI layer of IP topologies.

> [T]here has been less work on the automatic determination of LAN topology than WAN topology. A number of projects have looked at discovering the topology between IP routers, but because the most interesting portions of LAN topology are generally formed by level 2 devices, these projects have been unable to address the majority of LAN topology issues (Lowekamp, O'Hallaron, & Gross, 2001, p. 238).

OSI Layers 1, 2, and 3 will all be of interest when documenting a LAN. For the IP layer, similar techniques used for Internet and WAN discovery can serve equally well. However, discovery of the IP layer alone will miss the very important LAN components operating solely beneath Layer 3. Layer 2 devices – switches and bridges – are invisible to IP, but nonetheless, understanding of this logical network topology is of great value. Maps of Layer 2 will show a LAN's Spanning Tree revealing the actual paths on which data travels. Finally, the Layer 1 – physical – topology is equally valuable. Knowing it allows for LAN design to provide robustness even in the face of physical failures. For instance, in conjunction with Spanning Tree Protocol, physical links can be provisioned for redundancy and fail-over.

Hubs present a special issue in discovering a network's physical topology. A hub has no IP address. It has no ability to communicate or be queried, say by SNMP. Their presence can only be inferred in switched Ethernet by finding, simultaneous in time, multiple devices on a given port of a switch. This would be seen as multiple entries associated with a port in the forwarding table. Inferring their presence on a true bridge might be impossible. A true bridge will always have multiple entries for its two ports (assuming a port is not connected directly to a host).

The same mechanisms discussed up to this point are available for automatically discovering LANs: ping, traceroute, routing tables, forwarding tables, proprietary applications, and SNMP. Work by Lowekamp, et al. (2001) presents SNMP as a way to effectively learn the LAN topology. The argument they set forth is that topologies of LANs contain devices of switches, bridges, and hubs that are ignored by Layer 3 approaches. Past efforts to discover 'transparent' Layer 2 devices have relied upon forcing forwarding (bridge) tables to be populated by all other bridging devices throughout the LAN. This is not native behaviour. The weakness of SNMP is that it must be universally deployed. The analysis by Siamwalla, et al. (1998) concludes that SNMP is not as effective regarding completeness as relying on ping, traceroute, and DNS data sources either alone, or in combination in novel algorithms.

The idea of completeness as applied to LAN topology discovery is malleable. True completeness would necessitate finding every node connected to the network. This would include PCs, servers, and printers. It is likely that servers and printers will be as easy to find as routers and switches. They are nearly always on and usually assigned a static address. On the other hand, PCs are likely to be shutdown, moved around, being rebooted, and have their address change through time. All these factors are magnified for laptop hosts due to their increased mobility. If they are truly mobile, discovering clients participating on a wireless network may become even more problematic. DHCP helps contribute to identity volatility, while DNS, and now Dynamic DNS, introduce alias resolution issues.

Network managers might not need to be concerned with such a perfect definition of completeness. They are likely to be involved with assigning IP ranges for hosts (by way of DHCP scopes). They are less likely to be involved with managing the connected devices; a client/desktop support team is usually responsible for those duties. It is sufficient then to know which routers serve which subnets, and which switch ports are assigned to which subnets or VLANs. Moreover, maps need to present concise information. Showing every host in a LAN will be information overload for even a fully populated class C network. Iconographically, sets of hosts on subnets can be represented by single graphics, clouds, or simply annotated.

3.4 Discovery Mechanisms

Topology mapping programs must either communicate directly with a node to gather information from it, or infer characteristics about it based on responses to stimuli or ancillary information available in the network. The next sections overview the most common mechanisms used to gather data about a network.

3.4.1 SNMP

Realistically SNMP is but the dominant protocol used to gather information from, and modify configuration parameters within, networked devices. Other examples are Windows Management Instrumentation (WMI), Common Information Model (CIM), and Desktop Management Interface (DMI). SNMP's ubiquity may lie in the fact that it was one of the first management protocols. Its first definition came in 1988 (see IETF RFCs 1065, 1066, and 1067). The present day strengths and weaknesses stem from this version's design.

Because of its lack of security, the protocol is purposefully disabled if not used. It is well known that the first version of SNMP carried with it little in the way of security features. It simply employed a community string that served as a password. The string was communicated in clear text. It would be sent as such to any defined target, even if that target was incapable of responding. The second version, appearing in 1993 and now referred to as v2p, attempted to correct security issues, but was not received well due to its complexity. An alternate version, v2c defined in 1996, was instead widely adopted, but still relied on the clear text community string. SNMPv2c is still the *de facto* version implemented in networks. The year 2002 saw the release of v3 which addresses the long-standing security issues, but has been slow to be adopted.

SNMP is found in nearly any device that can be networked today. Even sub-components of PCs, such as NICs and other expansion cards, are likely to be able to speak SNMP even if the host system cannot. Somewhat recently, in 1998 (a decade after its introduction), according to Siamwalla, et al. (1998), SNMP was still not so common. On a "department" LAN they examined, consisting of approximately 500 nodes, SNMP was able to discover all of the routers and switches, and 99% of the hosts. When the same discovery was executed against a larger LAN of approximately 7500 nodes, the values dropped to 90% of routers and 8% of hosts. The results reflect the areas where SNMP has been adopted – mainly on network devices and not on PCs. It is recognized that "hosts" in these results included servers, which are more likely to have been running the protocol. Always an advantage, though, SNMP is

fast. For a short explanation of SNMP's architecture and components, see Appendix B.

If used as a vector to literally discover network nodes, SNMP encounters several issues. It will not discover devices where:

- there is a legacy device that does not support SNMP
- a capable device simply does not have the protocol enabled or running
- communication cannot occur because of security configurations (i.e. router/firewall rules disallowing the protocol)
- community strings to access the device are not known
- a device is configured to respond only to specific query-senders

Because of the lack of universality of SNMP at the time of Siamwalla, et al. (1998), they went on to use combinations of tools instead of management protocols to achieve discovery. Those were ping, traceroute, and DNS. These tools are common to many topology discovery systems.

3.4.2 Ping

Siamwalla, et al. (1998) begin discussing the use of ping by stating "every IP host is required to echo an ICMP 'ping' packet back to its source." This might have been the case at some historical point of the Internet, but is certainly not true today. The advent of viruses and DoS attacks that take advantage of this past truth have forced ping to be corralled tightly by a network's routers and firewalls. Indeed, Microsoft's Windows XP, arguably the most deployed operating system in the world, now contains its own firewall software that prevents responses to pings. So, ping is becoming less and less available as a tool to produce accurate results even inside our own LANs. When it is available, ping is fast. If not, there is a significant cost paid while waiting for a timeout to occur.

Broadcast ping is the ability to address a ping to a whole subnet. Each node on the subnet should receive and then reply to the request. This feature seems quite attractive as discovery costs can be handsomely reduced. Only one message must be sent instead of 254 in the case of a classic C-class, or /24 bitmask, network. The use of broadcast ping is limited by the same factors that limit ping. Moreover, networks and routers may be especially configured to disallow a ping addressed to a subnet in order to stop smurf[6] attacks. In the Siamwalla team's analysis of cost, broadcast ping was only marginally more or less efficient as algorithms using combinations of traceroute and DNS.

3.4.3 Traceroute

Traceroute is not susceptible to just being disabled as ping can be. Routers must report when time to live values reach zero by sending ttl-expired ICMP messages. Thus the tool can incrementally discover hops in a path from source to destination. One of the limitations faced is that traceroute can only provide us information about

[6] The smurf attack, named after its exploit program, is a denial-of-service attack that uses spoofed broadcast ping messages to flood a target system (Wikipedia, 2007a).

the Layer 3 hops in the path. It will be unable to discover lower layer devices like switches. Administrators may also manipulate ICMP responses in order to prevent information about the topology from leaking, collapsing the actual structure of the network from traceroute's view. Traceroute is costly because it sends multiple probes per hop and these probes are further spaced in time. It is accurate, but slow.

3.4.4 DNS

DNS almost seems to be a rough map on which we might base attempts to construct accurate new maps. It contains IP addresses that can be targeted for validation with probes by ping or traceroute or SNMP. In that sense, it may help reduce cost in time by avoiding the probing of non-assigned addresses. On the other hand, there is no guarantee that every IP enabled node is recorded in DNS. Essentially, DNS is a secondary source that always will require validation at the device/interface itself. Any discovery based on it will have its accuracy or completeness called into question. Instead of reducing costs the use of DNS has the potential to double the overhead of a discovery operation were each address first checked for presence in DNS before probing. This practice could lead to saving time only if there is a high incidence of non-present addresses. That is on the high end of cost of utilizing DNS. On the low end we can simply request all records of a specific type (A records in this case), or just execute a zone transfer. The zone transfer option is less and less likely to be available since DNS systems now have to be secured from attacks and to hide information. Some also disallow the listing of sets of records only allowing singe queries.

3.4.5 Other Lookup Tables

DNS is really just a name to IP table. Other tables available to exploit for documenting topologies are routers' route tables, switches' forwarding tables, and ARP caches. Each suffers from their own unique limitations. Route tables only show the relative position of networks, not a full set of network nodes. Forwarding tables and ARP caches will only contain information about nodes active on the network. Entries for inactive nodes will timeout and be purged from the records. Attendant limitations demand a lookup table's data to be interpreted, or recognized as possibly incomplete.

3.5 Analysis of work by Siamwalla Team

After reviewing literature concerning network topology discovery, the work by Siamwalla, et al. (1998) stood out as being particularly relevant to the scope and aims of this thesis. The team's central motivation was to measure ping, traceroute, and DNS as discovery mechanisms, and compare them to SNMP. They recognized that SNMP was faster and much less costly, but also that it was not present in legacy equipment, not universally deployed, and sometimes disabled due to lack of security in the protocol. The data produced reflected a reality influenced by those factors. A decade after its introduction, SNMP was less successful in terms of completeness at discovering network elements.

The initial interest in Siamwalla, et al. (1998) was to reproduce their work. It aimed to discover only the Layer 3, or IP, topology of a LAN environment. Results of a reproduction would have led to a direct comparison of the state of SNMP after another eight years in the marketplace. The blockade to performing a reproduction of their work was access to an analogous environment. The Siamwalla team first had access to their own department's LAN of 490 nodes on 7 subnets. Next, to test their algorithms on a larger scale they had access to an entire university network of "about 8200 hosts and 140 routers in more than 500 subnets" (Siamwalla, et al., 1998, p. 9). It can be inferred this access was fairly extensive since their algorithms required SNMP community strings, full DNS tables, and abilities to ping, all across the entire network.

Being unable to reproduce the Siamwalla team's work for direct comparison, the rest of Section 3.5 contains critical analysis focusing on the validity of key assumptions eight to nine years later.

3.5.1 Miscellaneous Factors

The team's discovery algorithms rely on three heuristics (their term). Two work to guess the subnet mask, and the third at guessing valid addresses in a domain. These heuristics take a seed IP value and generate a list of IP addresses to be validated by the main algorithm they feed. The heuristics themselves are in turn based on a set of assumptions. These assumptions were derived from observations about the team's test environment (i.e. the Cornell University network). One of the assumptions is that router interfaces acting as gateways for a LAN segment will always be assigned the first available address for the segment/subnet. This idea is repeated several times in various forms. Expecting that routers will be configured in this way may only be valid for a subset of all networks, therefore limiting the applicability of the team's algorithms. There is no standard, or even best practice, regarding LAN gateway addressing. Equally as valid, and quite common to observe, is the use of last valid IP address in a range for the gateway. This design choice is detectable on Napier University's own wireless and Ethernet LANs.

3.5.2 ICMP-based

Ping and traceroute are the two ICMP-based tools the team use in their algorithms.

Pinging a host is the act of sending an IP address an ICMP packet whose type is 8, corresponding to an echo-request. The team state "[e]very IP host is required to echo an ICMP 'ping' packet back to its source" (Siamwalla, et al., 1998, p. 2). This is patently not the case anymore. Security demands have pushed tactics reserved in the past for routers to be deployed at the PC host level. Now PC's regularly run their own firewall software. One of their most basic defences is to not respond to echo-requests. This is the case with Microsoft's Windows XP SP2 operating system, arguably one of the most ubiquitous in the world. Its default configuration turns on a software firewall and denies the system from responding to ping requests.

Pinging each individual host induces a predictable overhead value proportional to the number of hosts and how many echo-requests are sent to each. One way the Siamwalla team attempted to control this overhead was to use broadcast ping. A

broadcast ping permits a single echo-request to be addressed to every host in a network. A segment of 254 hosts could have ping echo-request overhead reduced from 254 to 1. The echo-reply overhead remains the same at one per request per responding host. Though useful, the danger of permitting broadcast pings is well known as a vector assisting DDoS attacks like smurf. Even at the time of their work, the team could not test broadcast ping outside their department network on the wider university system. The likelihood of denying broadcast ping has only increased since then.

Unlike ping, the traceroute tool cannot be deactivated. Traceroute exploits ICMP by purposefully manipulating Time-to-Live (TTL) values. Routers are required to decrement TTL values in the normal course of implementing the Internet Protocol. They are also required to notify senders if the TTL reaches zero through ICMP messages of types 3 (Destination Unreachable) and 11 (Time Exceeded). Traceroute remains a viable and un-impinged tool.

3.5.3 DNS

One of the Siamwalla team's most successful algorithms uses DNS data to seed a candidate list of nodes. The DNS data were obtained by requesting a zone transfer from the DNS server. The candidate list was then validated with ping or traceroute. As has been stated in other parts of this thesis, results relying on DNS are only likely to be as accurate as the original DNS entries. Especially at the time of the Siamwalla team's efforts, DNS tables were populated manually. This practice is an easy vector for introducing errors. It also allows for hostnames not matching the actual configured value on a node to be chosen. Hosts might change IP addresses, yet the hostname mapped to the "before" and "after" IP address in DNS would remain the same. Neither is there a requirement that a host have an entry in DNS. The team recognize these weaknesses when they state that DNS zone transfers "may not be complete, however, since hosts obtaining IP addresses using DHCP are not served by DNS. Moreover, some network managers disable DNS zone transfer due to security concerns" (Siamwalla, et al., 1998, p. 3). Ultimately, DNS is a source of data acting as a proxy for the real data on hosts.

Practices in today's networks simultaneously strengthen and weaken the value of relying upon DNS as a data source. The use of Dynamic DNS (DDNS) should help DNS tables be more accurate. DDNS was defined as a standard in RFC 2136 (Vixie, et al., 1997). Given the date of its debut, it is unlikely DDNS was deployed on the Siamwalla team's target networks. DDNS allows administrators to do away with manually populating DNS tables. Instead, hosts themselves can interact with a DNS server reporting their actual hostname and IP address, or DHCP can integrate with DDNS to provide the same information. In a network with correctly functioning DDNS, a DNS server might be expected to provide a real-time listing of hosts on the network and their actual IP assignments. A growing issue is accessing the DNS data, though. Today's networks must be more and more cautious about leaking information due to security concerns. One of the first activities to help harden a DNS server is to disable anonymous zone transfers. So, although DNS data may be more accurate, it is less available unless special permissions are configured to allow a discovery application to receive a full zone transfer.

3.5.4 SNMP

The team posits that SNMP cannot be universally relied upon mainly because it is not universally deployed. The construction of their algorithm that employs SNMP reflects the deployment issue in two key points. First, only routers are expected to be running the SNMP protocol, and are the only devices with which communication is attempted. Second, the initial method used to communicate with a target router is ping, not SNMP. If the router responds with an echo-reply, then and only then is an SNMP query sent to request information. The information of interest is the IP route table and the ARP table. The IP route table is used to find other router addresses. These routers become new candidates for further testing and discovery of the network. The ARP table contents provide evidence of hosts on the network.

The SNMP algorithm as defined by the team might face several difficulties stemming both from its design as well as modern security concerns. The security concern involves ping. Disabling echo-replies is a common hardening activity today. The algorithm is likely to be highly ineffectual in a network where pings are not answered by routers. The completeness achievable by the algorithm is further likely to be affected by two design choices. First, the algorithm does not attempt to contact hosts (non-router IP devices in this case) directly. In lieu of direct verification, ARP tables are relied upon to represent hosts. This is the second potential flaw. Later, Chapter 4, Section 4.1 sets forth the case that direct querying of nodes allows discovery, information retrieval, and validation to occur all in a single step. Data retrieved directly on a per-node basis requires little interpretation or necessity for assumptions. Not communicating directly with each node means secondary, representative sources must be used instead of the primary source. Like DNS, ARP is effectively a secondary, representative look-up table.

Relying on ARP table data mandates at least recognition of its nature and how that nature might affect a discovery process. The principal detail to recognize is that ARP table entries do timeout and are flushed accordingly. This may occur even if a host is physically on the network, but just inactive for the length of the timeout period. The Siamwalla, et al. (1998) data raises questions around this point, although the affect of ARP entry timeouts is not determinable from it. On the smaller, department network, the SNMP algorithm was very successful since all five routers were SNMP-enabled. The combined ARP tables failed to represent only three hosts (1% of total). Why these three were not represented in the ARP tables is left to conjecture, but they were discovered by another algorithm that elicited an echo-reply from them directly. The data from the larger, university-wide network raises even more questions. Despite 90% of the routers being discovered by the SNMP algorithm, their ARP tables represented only 8% of all hosts. A likely explanation is that a well-connected, SNMP-enabled router supplied the identities of most of the network's routers through its IP route table. Subsequently, that list responded to pings, but failed in the next step to provide further information via SNMP. Had they done so, a clearer picture might emerge concerning ARP. Would a similar 1% of hosts fail to appear?

Essentially, the Siamwalla team's SNMP algorithm only used SNMP as a mechanism to carry information discovered from ARP and protocol routing tables. While it is conceded these sources are highly accurate in representing other network nodes, they are secondary information sources.

Furthermore, the team's commentary lacks asking a deeper question about what is being measured. On the face of it the data directly compare SNMP with couplings of DNS/ping, DNS/traceroute, and ping/traceroute. The resultant differences in completeness, with SNMP suffering the most, are asked to be viewed as equivalent. When considered, though, SNMP completeness is a measure of a protocol state – it is either 'off' or 'on'. On the other hand, ping and traceroute can not help but be in the 'on' state as they are semantic manipulations of the ICMP and IP protocol stacks. With no recognition of the unbalanced nature of the comparison, the team state "that relying entirely on SNMP to discover network topology is a bad idea: though other algorithms take more time and more overhead, for a large topology, they are significantly more complete (Siamwalla, et al., 1998, p. 10). SNMP completeness would likely be buoyed upwards if the time and overhead were spent to enable the protocol on a network's nodes.

3.6 Measuring Discovery

The performance of a discovery system might be measured in various ways. Measurements can be direct or indirect, as well as ones of runtime or results. The importance of each is usually relative to the network environment at hand. Some important measures are:

- Speed (or time) – a direct measure of the runtime of a system's completion of a task.
- Overhead – an indirect value of the additional traffic put upon a network during runtime; redundancy, discussed in Section 3.2.1, is a type of overhead measurement.
- Efficiency – a ratio of at least two other measurements, such as overhead and time, revealing a characteristic per unit value.
- Completeness – a direct measure of results where failure to discover every target network node decreases the value.
- Accuracy – a measure of results expecting returned data to reflect the actual network environment.

These and other measurements are not only relative to a given network, but to each other. For instance, an increase in speed without a decrease in overhead causes the same amount of network load in a shorter timeframe (one type of efficiency). This could be a problem for a 10 Mbps network, but less of one for a 100 Mbps network.

Modern production networks are held to a high standard of uptime. Administrators are familiar with the paradigm of the "Five Nines"[7]. Five Nines refers to the percentage availability of a given system or set of systems. In this case, Five Nines means 99.999%, which, depending on how it is calculated, can mean as little as five minutes out of an entire year. Regardless of the feasibility of achieving such high order availability values, networks today are driven to be highly available. Thusly, when availability is expected, a discovery tool may concern itself more with accuracy and completeness versus quickness.

[7] See the Wikipedia entry Myth of the nines (http://en.wikipedia.org/wiki/Myth_of_the_nines) for a general treatment of the idea.

Since the goal under consideration is to produce documentation, it is reasonable then that speed and overhead might be sacrificed for completeness and accuracy. Resulting documentation will then be accurate too.

3.7 Conclusion

The design of a discovery system should be appropriate to its target network. Target network scopes are LANs, WANs, and the Internet. Discovery systems can be comprised of a single agent, multiple agents, or a distributed set of agents. The scope and the design should be well-matched, as well as work within the bounds of the resources available. Respect of resource constraints is especially important to avoid attack-mimicking and repetition of discovery work. Discovery work itself can be performed through a variety of tools. Common tools are SNMP, ping, traceroute, and DNS. Measurement of a system's performance is a means to confirm it meets design goals and operates within resource boundaries.

Chapter 4 Design and Methodology

4.1 Introduction

This chapter is a response to the issues presented from review of existing literature in the previous chapters. A concept design will be introduced which is high-level, yet incorporates specific choices and technologies. The reasons for electing one methodology or technology over another will be presented. The resultant system is one possible set of answers about effective means to produce documentation of a network.

4.2 Design Motivations

The first motivation was that the design be an effective tool for a system administrator. What defines an effective tool? From Chapter 2 we know that either circumstance or preference causes many system administrators to create their own function-specific scripts. These are considered effective because the sysadmin 1) controls the input; 2) understands the operation mechanics; 3) determines the output; and 4) desires automation resulting in time savings. All of these factors yield a high degree of trust placed in the accurate work of the tool by the sysadmin. So, our design must achieve trustworthiness by satisfying sysadmins' appeal for the same factors.

The second motivation was to create a modular architecture while relying on open standards. Modularity allows any stage's output to be examined, verified, ported, and so on by sysadmins. Simply by employing a modular concept to the process of translating information stored "in" a network into maps on a piece of paper goes a long way to ensuring that sysadmins understands the operational mechanics. This choice recognizes that the information about the network belongs to the network and its proprietors. There is a loss of trust and understanding if work and data is hidden away in code or proprietary file/db formats. Doing work and storing data using open standards permits easy examination, verification, porting, and so on of information. The tool will be flexible enough to allow novel applicability or integration in unforeseen ways. Additionally, in order to be applicable to any network some non-proprietary method must be employed against devices from the myriad device manufacturers. Finally, modularity and open standards allow data gathering and data storage to be separate from the logic about how to present the data.

The third motivation was that the tool help support, or even integrates with, network management systems. This is mostly achieved simply by outputting documentation. Documentation certainly has a role in fault, change, and configuration management as expressed in Chapter 2. The choice to create a modular and open standards-based system itself has an emergent quality that allows pursuit of this third design motivation to a further extent. Because of modularity, other applications might communicate with our system at distinct points. And by using open standards, such communications are simple to hook in the system.

The last design motivation was to query network nodes as directly as possible. Doing so causes the system to obtain actual, real-time information directly from network nodes. What a node reports about itself requires little in the way of verification. Its data *is* its configuration. External sources, on the other hand, do require accuracy checking. Additionally, they may only contain information valuable for a specific view of the network, such as physical versus logical IP views. DNS is an example of a source containing names, but not necessarily *hostnames*, along with IP addresses most valuable for mapping Layer 3. Such a limited source is of little value about how a node represents itself in a Layer 2 STP tree. Querying each node for desired data about itself obviates verification, interpretation, and correlation of multiple external sources.

4.3 Basic Design

Before haphazardly selecting technologies that might serve as appropriate means toward the end goal of creating documentation, it was better considered that a simple framework exist to divide and bound the overall system. A divided framework delineates internal functionality, much the same was the OSI model does. Each functional phase then is a module utilizing its predecessor's outputs and providing its successor's inputs. Given the context of our system being applied against a computer network and its goal to produce documentation, the following five functional phases need to be present:

1. Initialization – an input operation to define a target node list.
2. Communication – a module where work is performed against the node list to retrieve information from or about them.
3. Data Storage – the medium in which the information is stored.
4. Data Processing – the manipulation or filtering of the information in preparation for final delivery/layout.
5. Output – delivery of a desired type and content.

Figure 7 represents this modular framework graphically.

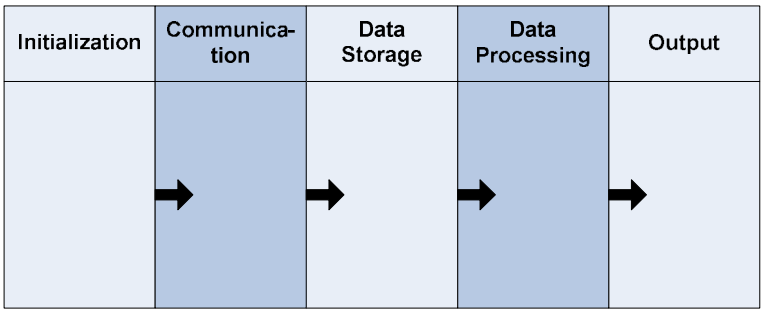


Figure 7 – Framework of a modular application design

The following sections will detail the specific technologies chosen to achieve each stage.

4.4 Initialization

The Initialization module must deliver to the Communication module a list of network nodes. This list could be any set of network characteristics ultimately leading to a node being identified. In order to keep within the modular framework, though, the Initialization module must be responsible for identity resolution work. The output of this work needs to be in an accessible format for sysadmins, and of an expected content for the Communication module. The simplest content type is IP addresses.

This conclusion does not limit how we arrive at a list of IP addresses. At disposal for use as initial sources are all the discovery mechanisms mentioned in Section 3.4, plus any number of unmentioned mechanisms. Whatever source is relied upon to deliver an IP list, processing of the source must deal with the node dynamics of alias resolution and identity volatility, as well as assumptions and limitations deriving from its own nature. For example, DNS may contain more than one record for an IP address, may contain inaccurate IP-to-name data, or may not contain a record at all about a node the Communication module is expected to output information about. Table 2 summarizes some mechanisms for determining a list of nodes grouping them by their type and theorized need for processing in order to arrive at an IP list ready for the system's subsequent module.

Type	Sources/Mechanisms	Expected Degree of Processing & Assumptions
Manually Defined	List of: IP, IP range(s), IP network(s)	Low
Lookup Tables	List of hostnames; DNS; route table; forwarding table; ARP cache	Medium
Semantic	Ping; Traceroute	Medium-High
Custom	Examples from literature: Doubletree; DIMES; CAIDA skitter; Route Views	High

Table 2 – Possible sources for determining a list of target nodes

There is little motivation to select a singular methodology for creating a node list. Instead, the Initialization module can itself be modularized allowing for existing and future node identity sources to be handled as long as each resolves to a consistent format ready for consumption by the Communications module. This decision modifies the basic system framework as it appears in Figure 8.

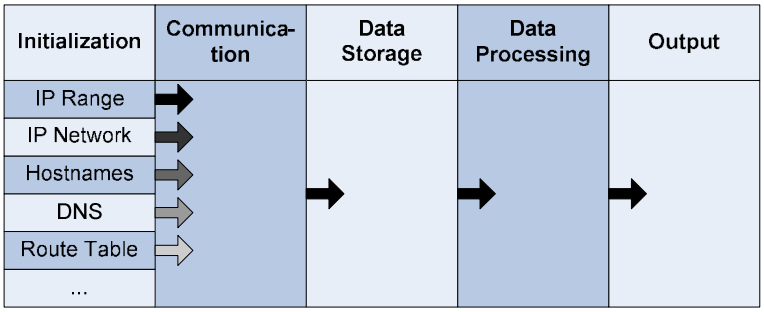


Figure 8 – Initialization module defined on the framework

4.5 Communication

The lines are a bit blurry in the literature about what is a discovery mechanism, communication channel, or data source, as those terms have been utilized so far. Ping, for example, seems to act as all three simultaneously as it "communicates" a message to a potential node; "discovers" the node through its response; and allows data (the node's IP address) to be "sourced" from the process. The modular approach to the design suggests the communication channel should be agnostic as to what data it carries. In order to discover not just nodes, but the topologies of different layers, it will be necessary to capture data pertinent to each layer. Tools like ping, traceroute, and DNS only convey IP information. By their design they are also bound to only being able to carry IP information. Other channels would then need to be utilized for other layers, not to mention even being aware of switches.

The candidate tool best matching the requirement and deliverables is SNMP. As a protocol, SNMP says how to communicate with a node. It leaves open what information it will carry. The node's SNMP agent database acts as the data source. This database contains physical port descriptions, MAC addresses, STP port states, IP addresses, hostnames, route tables, ARP entries, forwarding tables, etc. The SNMP agent acts as the data source – one authoritative about the node – and the protocol provides an agnostic channel to carry any information our discovery system desires.

Not only does our system need node information to identify them, but also means to identify links between nodes. One method by which links can be identified is by correlating returned data sets between connected devices. Such correlation work becomes exponential, though, in even a relatively small network. It might be necessary to examine every port in a network to find which is connected to a given port. The PTOPO MIB alleviates this work and further justifies the use of SNMP.

The Physical Topology Management Information Base (PTOPO MIB) was established in the year 2000 by RFC 2922 (Bierman & Jones, 2000). Its definition requires that each SNMP agent implementing the MIB is responsible for storing:

- a globally unique chassis ID
- a locally unique interface ID for each interface (port)
- a table of remote connections

Links can then be named using the chassis and interface ID's of devices on both ends (Figure 9).

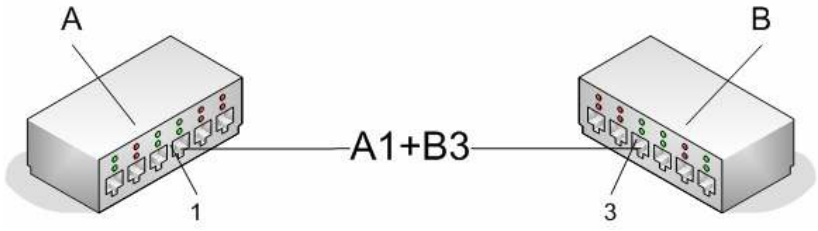


Figure 9 – How the PTOPO MIB defines globally unique identifiers for links

As a result, each node's SNMP agent contains the node's local topology. Interestingly, a network implementing the PTOPO MIB contains a pre-existing, built-in, distributed discovery agent architecture.

The advance of the design is shown in Figure 10. Note the addition of the "Node Pool" module representing the network.

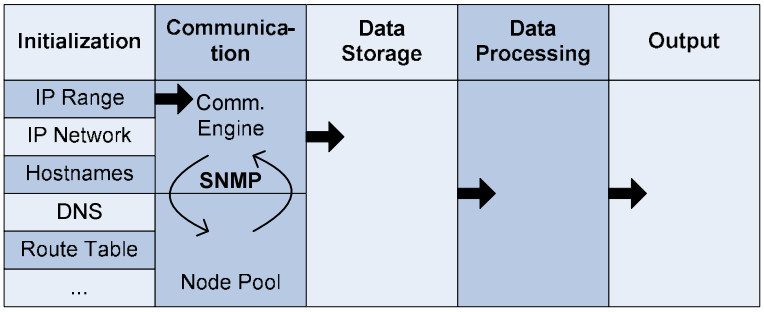


Figure 10 – Communication module defined on the framework

4.6 Data Storage

The design motivation to utilize open standards is really a broader statement having in mind two specific technologies and a desire to investigate their value in simplifying the stages of this system framework. The first was SNMP, which provides a fast communication channel and a wealthy information source. The second is Extensible Markup Language, or XML.

XML's own name is a bit misleading. It is not a language. It is really a specification for building tagging-based languages and applications. It is also intended to serve as a storage medium containing self-describing data (Ray, 2003). Every time an XML document is created and tag names are chosen, a new XML language and data store are being created simultaneously.

Nonetheless, it is still important that XML meets the other design motivations. It does in fact do so. This is as a result of the way XML is stored: as a file. An XML document file is easily accessible and its contents can be inspected by a system administrator. There are no needs for special file systems on which to store an XML file, or files, as there might be for a database. XML files can be sent to and stored at multiple locations either when they are created or at any time in the future. All these factors allow for a high degree of modularity. Applications could easily take advantage of this natural integration point in the design to enhance or extend the system, or be fed copies of the XML data for their own functional proposes.

It will be seen that the choice to rely on XML technologies in this middle data storage module leads to handsome opportunities in the ancestor modules. The evolution of the design is shown in Figure 11.

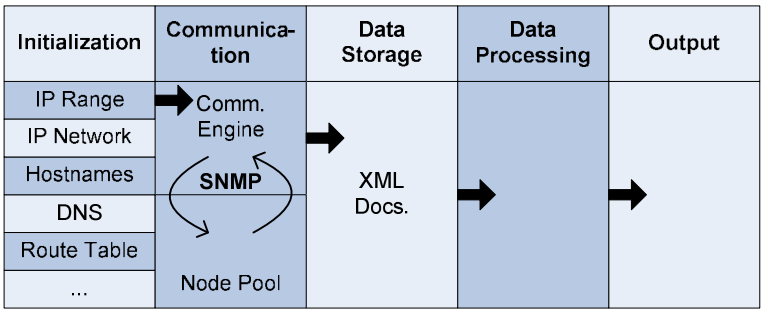


Figure 11 – Data Storage module defined on the framework

4.7 Data Processing & Output

Whatever is chosen to process the network information now stored in XML documents, must take that fact into account along with what output formats are desired. Since the final deliverable of the system is to be documentation, it makes sense to emulate the function and use of documents generated by existing means. Existing documents, like spreadsheets and Visio drawings, are stored as files, printed to paper, published to websites, and emailed as attachments. If our data processing engine can output documents that can be used in the same ways, it should deliver value to system administrators. This does not mean output must be in the same format, although it could be.

One of the ways much information is presented and shared is by publishing to websites. Thinking of our data in an HTML format is an interesting proposition since it is already in a XML format. Recall that XML is a specification for tagging languages. HTML itself is a tagging language. Despite the fact that HTML debuted 10 years before XML, HTML can be employed as *well-formed* XML (Ray, 2003). Any tagging language is considered well-formed if it follows the rules of XML. For HTML this means following the rules of XML while still adhering to HTML's own standards. When this is done, well-formed HTML is called XHTML. Still thinking about our data, it only needs one step – a transformation step – to go from XML tags to XHTML tags. After transformation, it would be its own web page, or content to be inserted into an existing web page.

It certainly would be possible to write an application that would perform such transformations. A significant drawback to such an approach is that different sets of code would likely be needed for each type of output format desired – one for XHTML, one for spreadsheets, etc. Investigation into technologies able to perform transformations reveals the solution is XML itself. Extensible Stylesheet Language (XSL) is another XML language/application specifically designed to take XML input, apply another XML transform document and output any type of content based on the combination of the two. XSL is an open standard consisting of three languages:

- XML Path Language (**XPath**) – a non-XML language for addressing (or navigating) XML document elements.
- XSL Transforms (**XSLT**) – XML language containing rule sets to transform documents.
- XSL Formatting Objects (**XSL-FO**) – XML language to describe paged media document formatting

XPath is used inside an XSLT document to address/navigate/select elements inside another XML document. The two documents can then be passed through an XSLT processor that outputs a new or reformed XML tree. To achieve document formats that are XML languages, like XHTML, this is all that is necessary. To achieve some other document formats, like the especially popular and portable PDF format, the XSLT processor must output an XML Formatting Objects, or .fo, file. This file can next be sent through a XSL-FO processor where a file is output. The scheme of the XSL system appears in Figure 12.

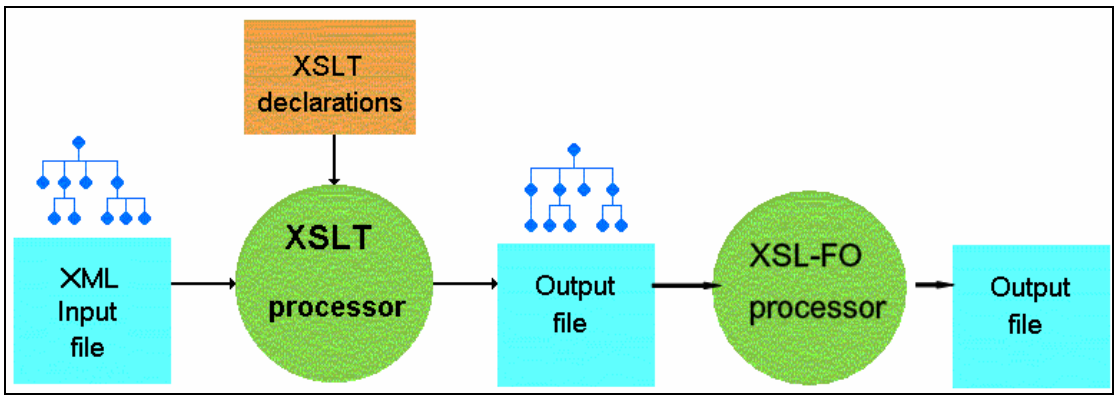


Figure 12 – Scheme of the XSL system

The XSL application is highly modular and agnostic concerning input and output, content, and format. An XSL processing engine will accept XML and XSLT documents as long as each avoids violating its own standards. As such, the processing engine does not contain any intelligence about what the final output document literally looks like. Instead, the final output document layout is determined at the front end of the process in the XSLT file. An XLST file acts in the place of a program's section of code to output different file formats. So, a separate transform file is needed for each file type (.html, .pdf, .doc, .xls, etc.) to be output. The advantage to this is that each transform file can be independently applied to the same XML data source. Also, a transform file can be edited, or a whole new one added, with no affect on the others. The implication is that a library of XSLT transforms needs to be kept. This is less daunting than might be imagined. Transforms aiming for different file types will be quite similar. Their XPath content will be nearly identical if they are intended for use against the same XML source. Their differences will be in how they decide to format the XML tree elements that are discovered in the source.

A transform library also seems preferable to a layout template into which source data is poured. A change in the source data structure might necessitate a modification of the layout template's logic. An XSLT transform, on the other hand, is free to ignore data it is not defined to address. Our system's XML documents can contain many, many elements of information from network nodes. A transform file only needs to deal with the data that is desired to be presented in a webpage or PDF file.

The final high level system design, showing the XSLT Library, a processing engine, and modular output document formats, is shown in Figure 13.

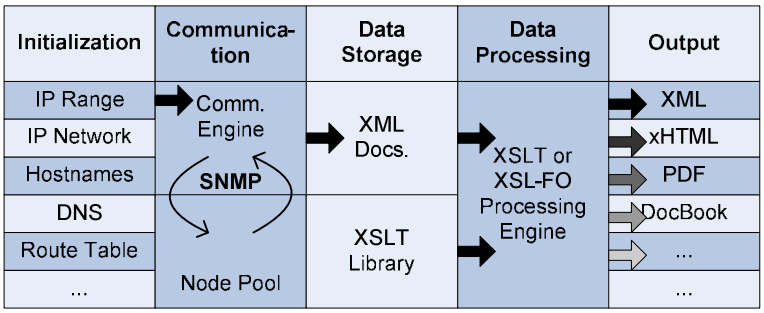


Figure 13 – Final design with all modules defined on the framework

4.8 Hypothetical Enhancements

To this point the design motivations have been sufficiently met through selecting technologies for each module in the framework. The implementation work to be shown in Chapter 5 will not attempt to go beyond demonstrating the elements of the system shown in Figure 13 above. It is easy to conceive, though, that the system can accommodate enhancement and extension. Due to the modular design it is not possible to account for every way this might be done, but a few ideas are discussed here.

A simple way to extend the automation of the system, and therefore its value and effectiveness for system administrators, is by adding job scheduling. Two natural points to implement scheduling are at the Communication and Data Processing modules. At the first, the SNMP-based communications engine would regularly poll nodes for data. It would then regularly output XML data. This is an ideal point at which to discover configuration changes though comparing old and new data. The second engine is also a good candidate for scheduling. As the XML data is updated the processing engine can automatically execute against it to produce up-to-date maps, web pages, etc. Both document sources serve as invaluable resources in attempts to backwardly track change in a network, as is necessary in fault management actions.

The job schedulers are placed at action points just prior to points where a type of document or file is produced. Tangential to document production is the authenticity and security of documents. Data showing details about network and nodes is sensitive and will no doubt be in our system's output documentation. When it is accessed or shared there may be needs to ensure it is controlled and verifiable as authentic. Another sub-module that can be inserted to provide such services is digital signing and/or digital encryption.

Section 4.5 briefly mentioned application integration. Integration can come in the form of a new module, which is inherently highly-coupled to the system. Imagine the Data Storage module is a repository to which multiple data processing modules might attach. Alternatively, an external application might have a more loosely coupled relationship to the system. Instead, imagine documents from the Output module serving as input to them. A Content Management system is a good example of such a relationship.

Integration can come on the front-end, too. The Initialization module's own modularity allows a large range of sources to inform the Communication Engine. An

appealing source is SNMP alarms and alerts from the node pool. Alarms, also called traps, could trigger an alert action by a monitoring system. One of those actions could then serve as input to the Communication Engine causing it to poll specific nodes. Subsequently, comparisons between new and previous data might reveal the root cause of the initial alarm condition. To go a step further, the system could act as the monitoring system itself where alarms are sent to it. Utilizing this example is one method by which network management activities as a whole become more integrated with documenting networks.

Reliance on open standards and modularity allow integrating new functionality in the overall design. Integration of the ideas presented in this section is shown in Figure 14.

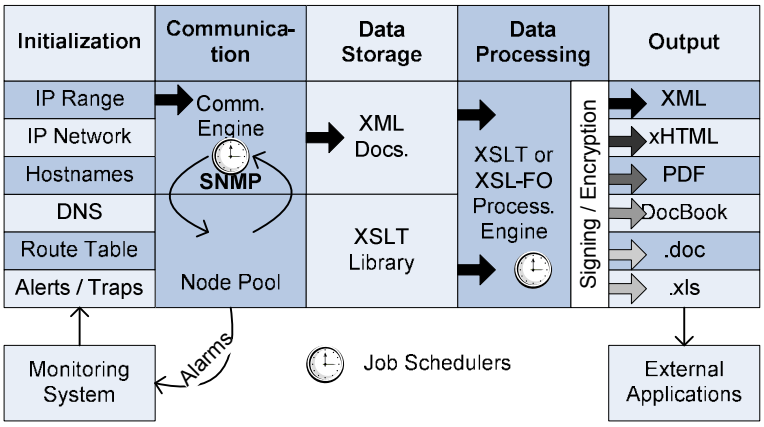


Figure 14 – Application design showing extensions of function and external integration

4.9 Conclusion

A framework whose goal is to deliver documentation was discussed, motivated in its design by open standards, modularity, integration with network management, direct communication with nodes, and value to system administrators. The resulting framework consisted of five modules with distinct missions to initialize the system, communicate with nodes, store data, process data, and output documentation. The first and last modules, where a target node list and document formats are determined, respectively, were themselves modularized allowing the overall system to accept multiple inputs and deliver multiple outputs.

SNMP was selected as the technology upon which communications to nodes would rely. SNMP has been demonstrated to be a fast and relatively less costly channel than semantic manipulations (Siamwalla, et al., 1998). It also provides the option to post changes back to a device. Furthermore, it provides access to data about links between nodes via the PTOPO MIB. This MIB theoretically supplants the need to compute how nodes are linked.

XML technologies provide the design with storage mediums as well as a data agnostic processor capable of outputting many types of formats. Data will itself be stored in XML files. The XSL family of languages provides first XSLT and XPath to define how the source XML will be transformed. It then provides standards about XSL

processors that can deliver well-formed XML content, or proceed through another processing step to deliver visually formatted document types, such as PDF.

Lastly, extensions and enhancements to the design were theorized. The extensibility of the design is due to the initial motivations to utilize open standard technologies along with modularity.

Chapter 5 Implementation

5.1 Introduction

The system designed in Chapter 4 was implemented. This chapter describes the tools used to realize the design and the implementation work. The result of the work is a software application that integrates all five of the modules in the design framework. An overview of the application will be given for orientation, followed by reporting on each module.

The purpose of the application is to explore and understand the methodologies and technologies, both in terms of their feasibility and limitations, by which each of the five design modules can be accomplished. Table 3 in Appendix A gives a list of resources utilized to realize the proof-of-concept application. The table's entries are annotated as either primary or secondary types. Primary types of resources were necessary in implementing the application. Secondary types were used for background research and verification.

5.2 Overview of the User Interface

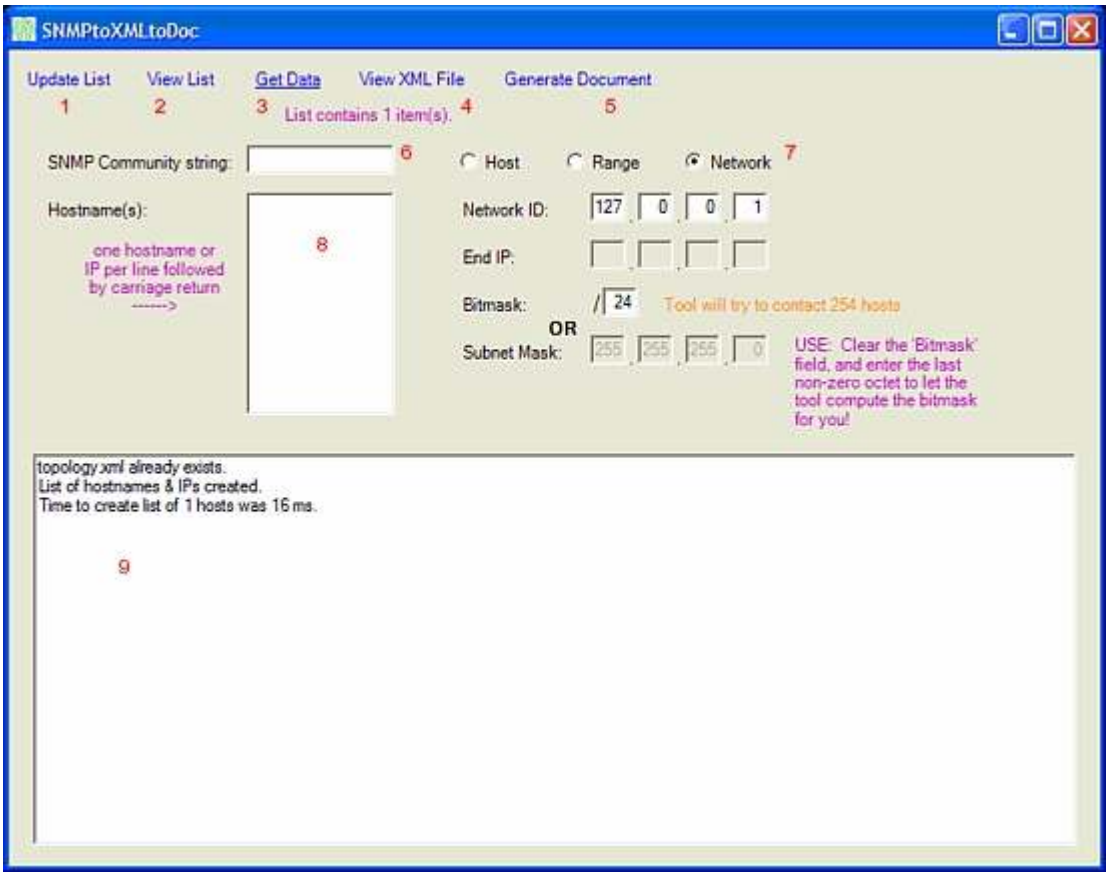


Figure 15 – User interface created for the implementation

REFERENCES FOR FIGURE 15

1. Create/Update List – button to create a list of nodes based on user input values.
2. View List – button to launch list for inspection into a text viewer.
3. Get Data – button to initiate main function of contacting nodes via SNMP, and outputting discovered data to an XML file.
4. View XML File – button to launch XML file for inspection into a text viewer.
5. Generate Document – button to send XML file to XSLT processor for transformation.
6. SNMP Community string – text field to enter the SNMP 'password.'
7. Host/Range/Network – area to define target nodes by IP address, range of IP addresses, or by IP network using CIDR bitmask or subnet mask.
8. Hostname(s) – text list to enter node names to be resolved by DNS.
9. Unlabeled text area to for the tool to show status messages and progress results.

Figure 15 displays the application's user interface. Subsequent sections will expound on the relevant elements of the interface, and then continue to concentrate on how a given module from the design was addressed. Appendix D contains the complete code written to create the application.

5.3 Implementation of the Initialization Module

The Initialization module made no demands of a specific means by which to produce a list of target nodes. Because SNMP is employed, it is only necessary to deliver IP addresses for the Communication module that follows. SNMP can also rely on a network's DNS to resolve hostnames, so they might be permitted, too. The implementation chooses to allow both IP addresses and hostnames. These data are determined manually through user input.

Hostnames are permitted to be entered in a list box control that accepts multiple lines of text. Hostnames must be entered one per line. IP addresses can be defined in one of three ways. Option one is that a single host's IP address may be entered. Option two allows a start and end address defining a continuous range of addresses to be entered. Option three requires a network number to be entered along with a Classless Inter-Domain Routing (CIDR) bitmask or a subnet mask. The initial choice of host, range, or network is controlled by three corresponding radio buttons. The selected radio button controls the availability of the entry fields below, only allowing fields to be accessible pertinent to the selection (Figures 16, 17, & 18).

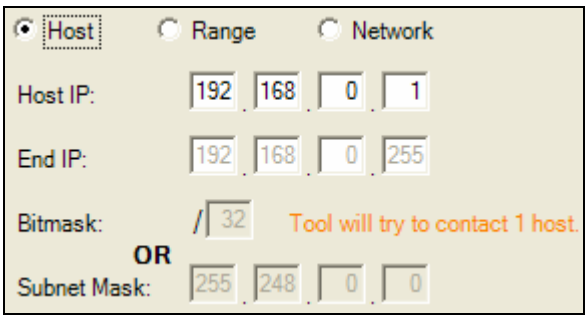


Figure 16 – Defining a single IP address in the GUI

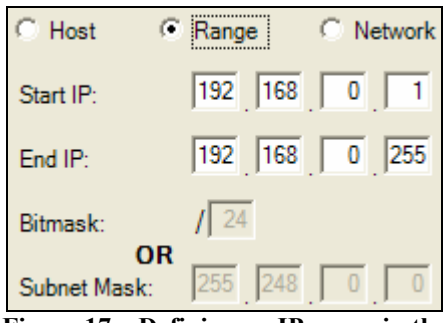


Figure 17 – Defining an IP range in the GUI

Figure 18 – Defining an IP network in the GUI

Regardless of which type of IP address is chosen by the user, the individual fields where IP octets are entered check that the entries are valid (Code Snippet 1) Should an entered value be outside the allowed range for an IP octet (whether numeric, text, or null), the field informs the user by changing to a red colour. If the Bitmask or Subnet Mask field is being used, an orange text alert is displayed (Figure 19). Furthermore, if the user attempts to create or update the host list based on non-valid values, a message is displayed in the status box below.

```
private void txtStartOctet1_TextChanged(object sender,
System.EventArgs e)
{try
  { if ((Convert.ToInt32(txtStartOctet1.Text)<0)^
           (Convert.ToInt32(txtStartOctet1.Text))>255)
      txtStartOctet1.BackColor=Color.LightPink;
    else  txtStartOctet1.BackColor=Color.White;
  }
  catch{txtStartOctet1.BackColor=Color.LightPink;}
}
```

Code Snippet 1 – Example of integer range and non-integer validation

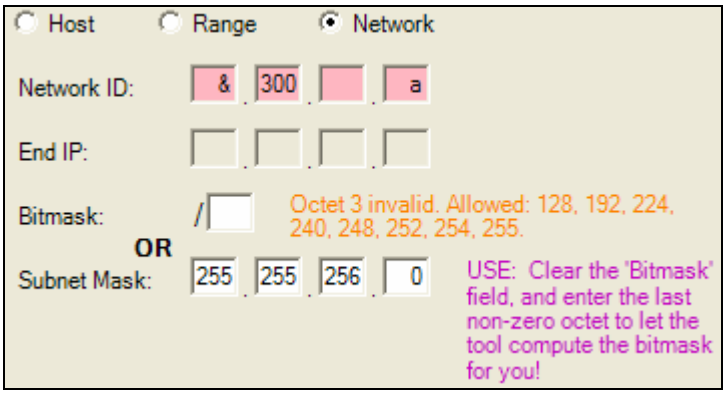


Figure 19 – Alert message displayed when the subnet mask is ill defined

Once hostnames and/or IP addresses are defined, the user must activate the 'Create List' button. The code behind the button completes two main tasks (see Appendix D, Section D.11). The first is to generate any and all addresses when an IP range or network is defined (see Appendix D, Section D.6). The second is to add all the defined hosts to an array in memory. This array is later passed to code dealing with node communication. After the initial creation of a list, the button changes its label to 'Update List,' and makes available the additional buttons 'View List' and 'Get Data.'

Activating the 'Update List' button does little different than before, although it will reinitialize the array in memory and repopulate it with elements based on the actual settings and values of the fields controllable by the user (Figure 20).

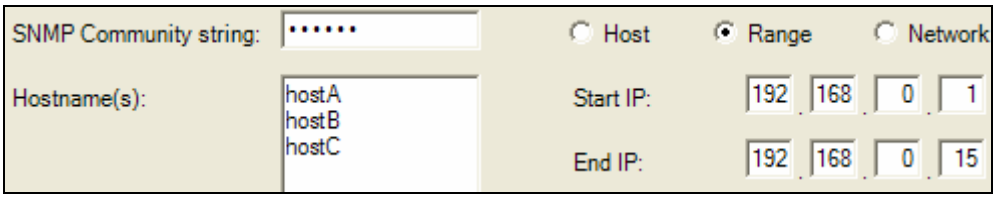


Figure 20 – Three hostnames and a range of 15 IP addresses defined in the GUI

Pressing the 'View List' button writes a text file to the disk containing the elements of the array (Code Snippet 2). Selecting to view the list always destroys any existing text file by creating a new one based on the current contents of the array in memory. After the file is written, it is sent to the system to be handled by whatever external application is responsible for file types of .txt (usually Notepad in Microsoft Windows operating systems) (Figure 21).

```
private void llblViewList_LinkClicked(object sender,
System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
{ if (File.Exists("list.txt"))
    File.Delete("list.txt");
  using (StreamWriter writer = new StreamWriter("list.txt"))
  { for (int i=0;i<IPAR.Count;i++)
      writer.WriteLine(IPAR[i]);
  }
  System.Diagnostics.Process.Start("list.txt");
}
```

Code Snippet 2 – The list of nodes is written to a text file for viewing

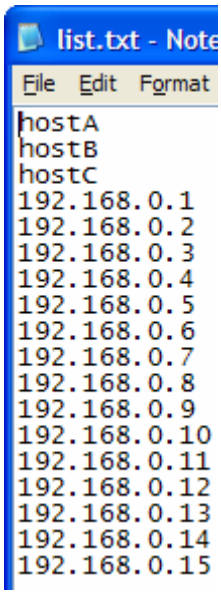


Figure 21 – Target node list of 18 items generated by settings seen in Figure 20

The implementation of the Initialization phase only allows manual user input. This is but one of the methods pointed out in Section 4.4. This method of input was chosen for its relative simplicity. Explicit and actual device discovery is not performed. The tool takes user-provided "seed" values to generate a host and IP list. This manual

method still permits the implementation to demonstrate value to the user, modularity, and the ability of the user to verify the work of the tool. The value to the user comes in permitting the tool to do the work of computing IP ranges. This is a time-consuming task, otherwise. The tool also controls for invalid IP octets, bitmasks, and subnet values. Modularity is shown by the option to utilize the very simply formatted text file, as it is available as a separate file on the system's hard disk. These same results are easily accessible and verifiable by viewing the host list prior to initiating the communications phase.

5.4 Implementation of the Communication Module

The design of the Communication module is more specific than its predecessor. It calls for the use of SNMP to gather information directly from nodes. It also asks that information about links between nodes be gathered through the PTOPO MIB.

The only way this module is represented in the user interface is through the 'Get Data' button. Behind this button lies all the code that implements SNMP communication to nodes supplied in the node list. As a final feedback mechanism to the user, hovering the mouse pointer over this button reports the number of items in the node list array (Figure 22).

Figure 22 – Hovering over the 'Get Data' button displays the count of target nodes

The last unacceptable condition that may exist is that of the user not supplying a password. If the 'SNMP Community string' field is blank, clicking the 'Get Data' button results in a message stating so in the status area below. If the value of the string field is not null, communication to each of the list elements in initiated.

Microsoft's set of libraries provided with the Visual Studio suite of products and the .NET Framework does not include a library for the SNMP protocol. As indicated in Table 3 of Appendix A, a library written in 2003 by Malcolm Crowe of the University of Paisley was employed (Code Snippet 3).

```
using Snmp;

uint[] ciscotype = new uint[] {1,3,6,1,4,1,9,2,1,3,0};
ManagerSession sess = new ManagerSession(IPAdd,CommString);
ManagerItem mi3 = new ManagerItem(sess,ciscotype);
```

Code Snippet 3 – Reference to the snmp.dll and usage of its classes

Because SNMP is a UDP protocol, it is not possible to establish an end-to-end connection. SNMP is only capable of sending a request for information. It must then wait for responses as they come. The proof-of-concept application parses the node list serially, contacting each element one at a time. Should a node not exist, or not send a SNMP response, it is undesirable that the application wait forever. To deal with this situation the initial SNMP request is implemented through an asynchronous call (Code Snippet 4).

```
AsyncMethodCaller caller = new AsyncMethodCaller(CallHost);
IAsyncResult myresult = caller.BeginInvoke(IPAdd,
CommString,null,null);

public delegate Universal[] AsyncMethodCaller(string IPAdd, string
CommString);

private Universal[] CallHost(string IPAdd, string CommString)
{ uint[] sysName = new uint[] { 1, 3, 6, 1, 2, 1, 1, 5, 0 };
  uint[] ifNumber = new uint[] { 1, 3, 6, 1, 2, 1, 2, 1, 0 };
  uint[] sysDescr = new uint[] { 1, 3, 6, 1, 2, 1, 1, 1, 0 };
  ManagerSession sess = new ManagerSession(IPAdd, CommString);
  Universal[] oids = sess.Get(sess.VarBind(sysName),
sess.VarBind(ifNumber), sess.VarBind(sysDescr));
  sess.Close();
  return oids;
}
```

Code Snippet 4 – Implementation of the asynchronous call

After the call is initiated, another section of code keeps a timer running while checking the completion status of the call. The call either will receive a response, thereby completing successfully, or will timeout after 10,000 milliseconds (10 seconds) (Figure 23). If successful, the host will have returned values for the system name, the number of interfaces on the system, and the system description. Later, additional requests retrieve eight more values describing each individual interface on the system. These subsequent calls are not implemented asynchronously. It is assumed the node will continue to respond as it already did.

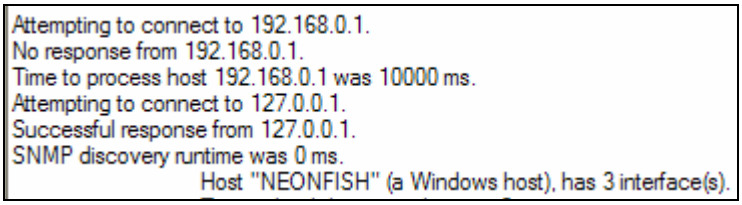


Figure 23 – Status output showing no response from a target node after 10 seconds (lines 1 -3)

In line with the design, information is retrieved via SNMP. Indeed, any information stored in the node's SNMP agent database is available. The application code simply needs to be tasked with the proper Object Identifiers (OIDs) to request, and then deal with the response message and content. The basic information gathered about each node's system and interface attributes is the limit of what was actually implemented. The output of this data will be seen in the next section dealing with XML and data storage.

Another aim of the design was to gather link information by retrieving it from the PTOPO MIB data on each node's SNMP agent. This was not achieved. As might be guessed from the previous paragraph, this was simply because it was not available; it did not exist. Further investigation revealed that Cisco Systems did not implement the PTOPO MIB on the platforms available for testing (see Table 3, Appendix A). And, despite the PTOPO standard being 6 years old, Cisco have only implemented it on

two of their products. Both are high-end, optical networking platforms intended for use by organizations such as large Internet Service Providers[8].

Overall, the implementation still demonstrates the core principals of the design. It takes a list of nodes then gathers information directly from responding nodes using SNMP, an open standard protocol.

5.5 Implementation of the Data Storage Module

The choice of technology in the design of the Data Storage module was XML. Information retrieved by the application's code is so far still in memory. Now it must be written to a file external to the application.

Upon launch, the application checks if the target XML file already exists. If it does not, it is created (Code Snippet 5). When created for the first time, the XML file only contains the minimum elements necessary to build an XML tree. These elements are an XML header identifying the content as XML, and a document root. Respectively, each is analogous to an HTML head section, and a body, albeit null of content. See Appendix C, Section C.1 for further information about XML.

```
public void DocTest ()
{ if (!(File.Exists(xmlFile)))            //test for/creates XML file
  { lstOutputStatus.Items.Add("XML file does not exist! Creating..");
    XmlTextWriter docmaker = null;
    docmaker = new XmlTextWriter (xmlFile, Encoding.UTF8);
    try
    { docmaker.Formatting = Formatting.Indented;
      docmaker.Indentation= 6;      docmaker.Namespaces = false;
      docmaker.WriteStartDocument();
      docmaker.WriteProcessingInstruction("xml-stylesheet",
"type='text/xsl' href='web-device-report.xslt'");
      docmaker.WriteStartElement("topology", "");
      docmaker.WriteComment("Start of device listing");
      docmaker.WriteEndElement();      docmaker.Flush();
      lstOutputStatus.Items.Add("Created topology.xml.");
      llblViewXML.Visible=true;
      llblGenDoc.Visible=true;
    }
    catch(Exception e)    {lstOutputStatus.Items.Add("Error creating
XML file: "+e.ToString()+".");}
    finally    { if (docmaker != null)      {docmaker.Close();}
    }
  }
  else
  { lstOutputStatus.Items.Add("topology.xml already exists.");
    llblViewXML.Visible=true;    llblGenDoc.Visible=true;
  }
}
```

Code Snippet 5 – Creating the initial XML file

[8] Cisco's MIB Locator tool was used to confirm support of the PTOPO-MIB on their product range. The tool is located at http://tools.cisco.com/ITDIT/MIBS/.

Figure 24 shows the actual initialized text of the XML file the application creates. The first line is the document header. The opening and closing tags with the name "topology" comprise the document root with only a single commentary line as a body.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type='text/xsl' href='web-device-report.xslt'?>
<topology>
      <!--Start of device listing-->
</topology>
```

Figure 24 – Content of the initialized XML data file upon application launch
N.B. The second line is not absolutely necessary for well-formed XML; it is present to prepare the file for transformation into a final output document.

Because the application ensures an XML file exists, the 'View XML File' button is always available to the user. Activating it sends the XML file to Windows' Notepad text editor. Otherwise, the user interface offers little to indicate a separation of the data retrieval process and the creation of the XML store for that data. Indeed, to the user the 'Get Data' button activates and completes both processes. The code behind the control is contained in distinct programming objects. Once information has been gathered from a node, it is stored in an array in memory. The last action of the main code block responsible for handling SNMP communication is to call another code block (see Section D.7 of Appendix D). This second section of code writes the given node's information into the XML file.

The code block which writes to the XML file is only concerned with adding the current network host information and the information for each of its interfaces (see Section D.10 of Appendix D). It first checks for any existing device in the XML file that has an identical system name. The system name is expected to be a unique value. If a duplicate is found, no writing occurs and a message is output to the user that the specific device already exists. Else, the information about the network node is appended to the list of devices in the XML file. Figure 25 show the content of the file after a single device with two interfaces has been added. Appendix E, Section E.1 contains an example XML file containing two devices, each with multiple interfaces.

Figure 25 allows the grammar of this particular XML language to be seen. Recall that any implementation of XML is its own XML language, standardized or not. This non-standard, unnamed language has a root element named "topology." The "topology" element contains a single child element of "device." It could contain any number of child elements and any number of "device" child elements. "Device" has an attribute named "type," along with three children: "sysName," "ifNumber," and "interface." Like its parent element, "interface" has an attribute named "ifIndex" and seven child elements.

Due to the SNMP and XML actions being performed in customized code, the implementation achieves only a low degree of modularity between the Communication and Data Storage modules. The modules are implemented in distinct code blocks, but remain coupled to each other. If a new OID value is to be retrieved from nodes, additional code must be added to do so. Arrays must be expanded and re-ordered. Likewise, the code responsible for writing data to the XML file must be modified in accordance with modification to the source data array, and new XML element tags must be chosen. The application does output an XML file which puts a distinct line between it and the successor module of Data Processing.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type='text/xsl' href='web-device-report.xslt'?>
<topology>
  <!--Start of device listing-->
  <device type="Windows host">
    <sysName>NEONFISH</sysName>
    <ifNumber>2</ifNumber>
    <interface ifIndex="1">
      <ifDescr>MS TCP Loopback interface</ifDescr>
      <ifType>softwareLoopback</ifType>
      <ifSpeed>10000000</ifSpeed>
      <ifAdminStatus>1</ifAdminStatus>
      <ifOperStatus>1</ifOperStatus>
      <IPAddress>127.0.0.1</IPAddress>
      <NetMask>255.0.0.0</NetMask>
    </interface>
    <interface ifIndex="2">
      <ifDescr>Intel(R) PRO/Wireless 2200BG</ifDescr>
      <ifType>ethernetCsmacd</ifType>
      <ifSpeed>54000000</ifSpeed>
      <ifAdminStatus>1</ifAdminStatus>
      <ifOperStatus>2</ifOperStatus>
      <IPAddress>0.0.0.0</IPAddress>
      <NetMask>0.0.0.0</NetMask>
    </interface>
  </device>
</topology>
```

Figure 25 – XML data file containing a single device with two interfaces

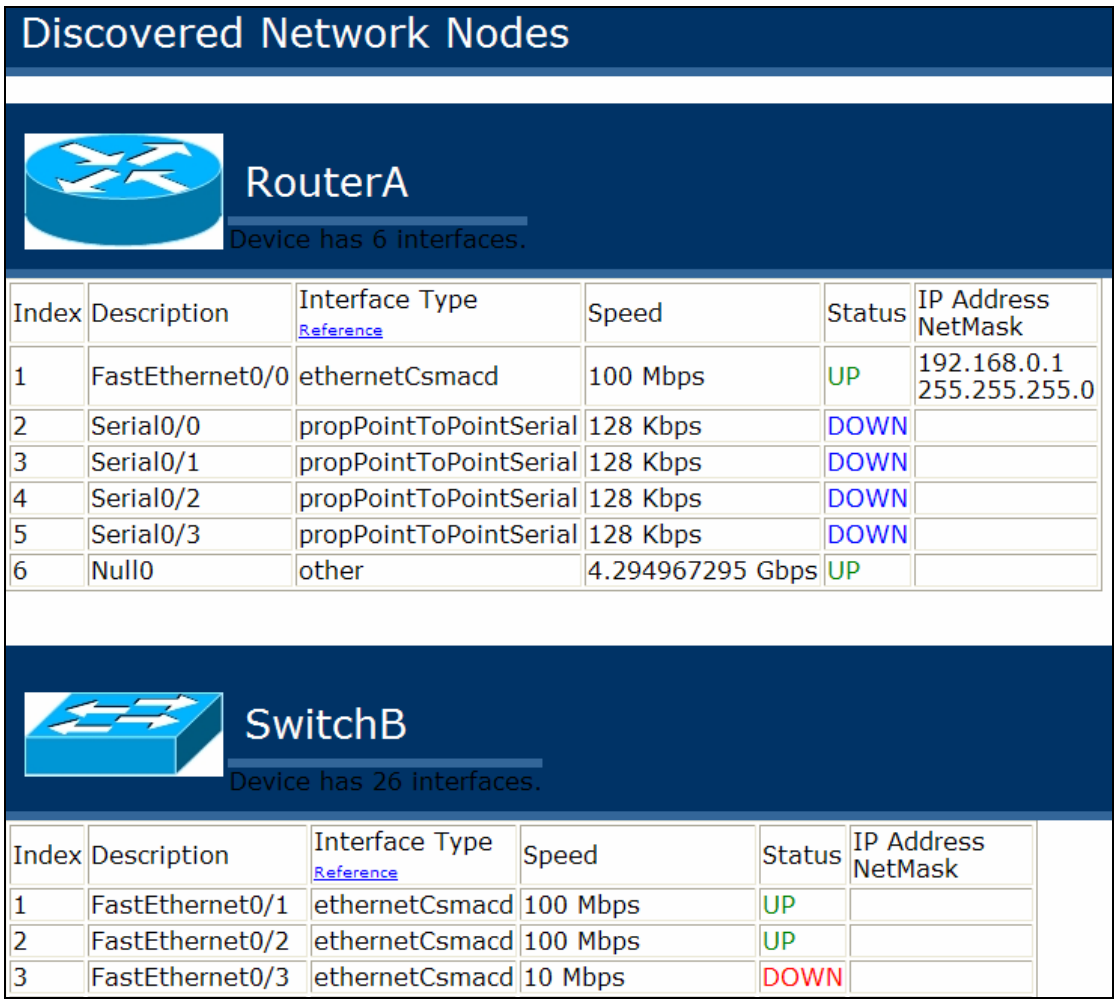
5.6 Implementation of the Data Processing & Output Modules

To move from the Data Storage module through Data Processing into the Output module, the design decided to use the XSL family of technologies. In practice this first requires XSLT files for defining transformations. Each XSLT file maps to a single file type and visual layout combination. Next, the original XML-stored data and transforms are passed to XSLT and XSL-FO processors for final output. The implementation of these stages did achieve transformation from XML to both HTML and PDF.

Transformation to HTML only requires the fist half of the XSL framework (see Figure 12 in Chapter 4, Section 4.6). That is, only an XML file, a transform file, and a XSLT processor are used to produce an HTML document. There is no need to pass the HTML through the second half's XSL-FO processor. Neither is there usually a need to produce the resultant HTML as an actual file. Instead, the HTML is only produced in memory for presentation in an HTML interpreter – almost always a web browser. Fortunately, modern web browsers have their own built-in XML and XSLT processors.

When an XML file is loaded by a web browser, it notifies the browser that its content should be transformed and presented according to a specific .xslt transform file. This is accomplished through a special XML Standard-defined tag, 'xml-stylesheet.' The tag contains attributes telling the browser that the stylesheet will be of an XSL type, and the path to the transform file. This is shown in the second line of Figure 24, above.

Two .xslt files were written that each transform the same XML source to HTML, but produce different layouts. The first is shown in Appendix E, Section E.2. Its product is a table format containing information about each device and their interfaces (Figure 26).



Discovered Network Nodes

RouterA

Device has 6 interfaces.

Index	Description	Interface Type Reference	Speed	Status	IP Address NetMask
1	FastEthernet0/0	ethernetCsmacd	100 Mbps	UP	192.168.0.1 255.255.255.0
2	Serial0/0	propPointToPointSerial	128 Kbps	DOWN	
3	Serial0/1	propPointToPointSerial	128 Kbps	DOWN	
4	Serial0/2	propPointToPointSerial	128 Kbps	DOWN	
5	Serial0/3	propPointToPointSerial	128 Kbps	DOWN	
6	Null0	other	4.294967295 Gbps	UP	

SwitchB

Device has 26 interfaces.

Index	Description	Interface Type Reference	Speed	Status	IP Address NetMask
1	FastEthernet0/1	ethernetCsmacd	100 Mbps	UP	
2	FastEthernet0/2	ethernetCsmacd	100 Mbps	UP	
3	FastEthernet0/3	ethernetCsmacd	10 Mbps	DOWN	

Figure 26 – HTML output of the XML data file transformed by the first .xslt file

The second .xslt file (Appendix E, Section E.3) presents the same information from the same XML source file. Its presentation is graphical instead of tabular. Hovering the mouse pointer over an interface shows information about it (Figure 27).

Figure 27 – HTML output of the XML data file transformed by the second .xslt file

The application's user interface presents a button named 'Generate Document.' The limit of the functionality of this button is to send the XML data file to the system's default handler for .xml file types (Code Snippet 6). This is usually a web browser. The XML file is already seeded with the 'xml-stylesheet' tag pointing to the first (table layout) transform file. The second transform result (graphic layout) can be produced by manually editing the XML data file and changing the value of the file name referenced by the 'xml-stylesheet' tag.

```
private void llblGenDoc_LinkClicked(object sender,
System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
{
  System.Diagnostics.Process.Start(xmlFile);
}
```

Code Snippet 6 – Clicking the 'Generate Document' label sends the XML file to an external handler

Production of a PDF file is not a dissimilar process. Again, a transform file must be written. The XML data file and transform file are passed through a XSLT processor. Instead of HTML, the result is a Formatting Objects XML tree. Formatting Objects is itself a standardized XML language for describing how text and pictures (objects) are to be laid out (formatted) in a defined space. Most commonly these defined spaces are dimensions of a sheet of paper. A FO processor takes the FO tree and outputs a file format suitable for printing. A preferred file format is PDF.

The .xslt transform written to create a FO tree is presented in Appendix E, Section E.4. Normally, the .xml and .xslt files are passed into a processor and a PDF is output directly. The intermediate step of creating the FO tree is performed only in memory. This is the case with the processor employed, fop 0.92 beta. However, it did allow for the procedure to be halted midway through, and the FO tree to be written to a .fo file. Such .fo files can be quite lengthy due to the many attributes that must be described in forming even a single page layout. Therefore the .fo file appears, but truncated, in Appendix E, Section E.5.

This third .xslt file, written to produce the FO tree and PDF document, was purposefully engineered to mimic the tabular layout achieved by the first HTML-

generating .xslt file. The first page of the PDF file result is shown in Figure 28 (one page on).

The application's user interface does not provide any access to, nor does the application code integrate with, the FO processor. Transformation and PDF document production were executed solely through fop 0.92 beta's command line interface.

At its conclusion the implementation successfully realized the design's elements for data processing and output. It employed XSL as the operational framework. A small XSLT library of three transforms was built. These transforms were independently combined with a single XML data file to produce three different documents with their own distinct layouts. The documents produced were of differing types; one PDF and two HTML. Web browsers and a FO engine were used as processors. This demonstrated modularity in that multiple applications could act as the Data Processing module depending on the type of output required. Doing so did not demand a change to the source XML data. The final output documents produced were in formats that are easy to store, print, publish to a website, or attach to email messages.

5.7 Conclusion

The realized application is modelled within the design framework in Figure 29.

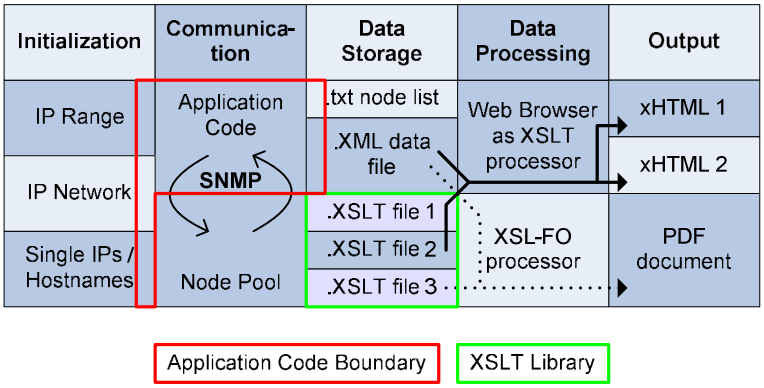


Figure 28 – The realized implementation mapped onto the design framework

Each module of the design was implemented successfully enough to act as a proof-of-concept. The implementation adhered in most places to the motivating concepts of being an effective and transparent tool employing open standard technologies in a modular way. It also directly queried network devices versus relying on secondary sources from which only inferences can be made about a device's attributes and configuration values. The implementation keeps in mind, but does not directly address, integration with network management systems.

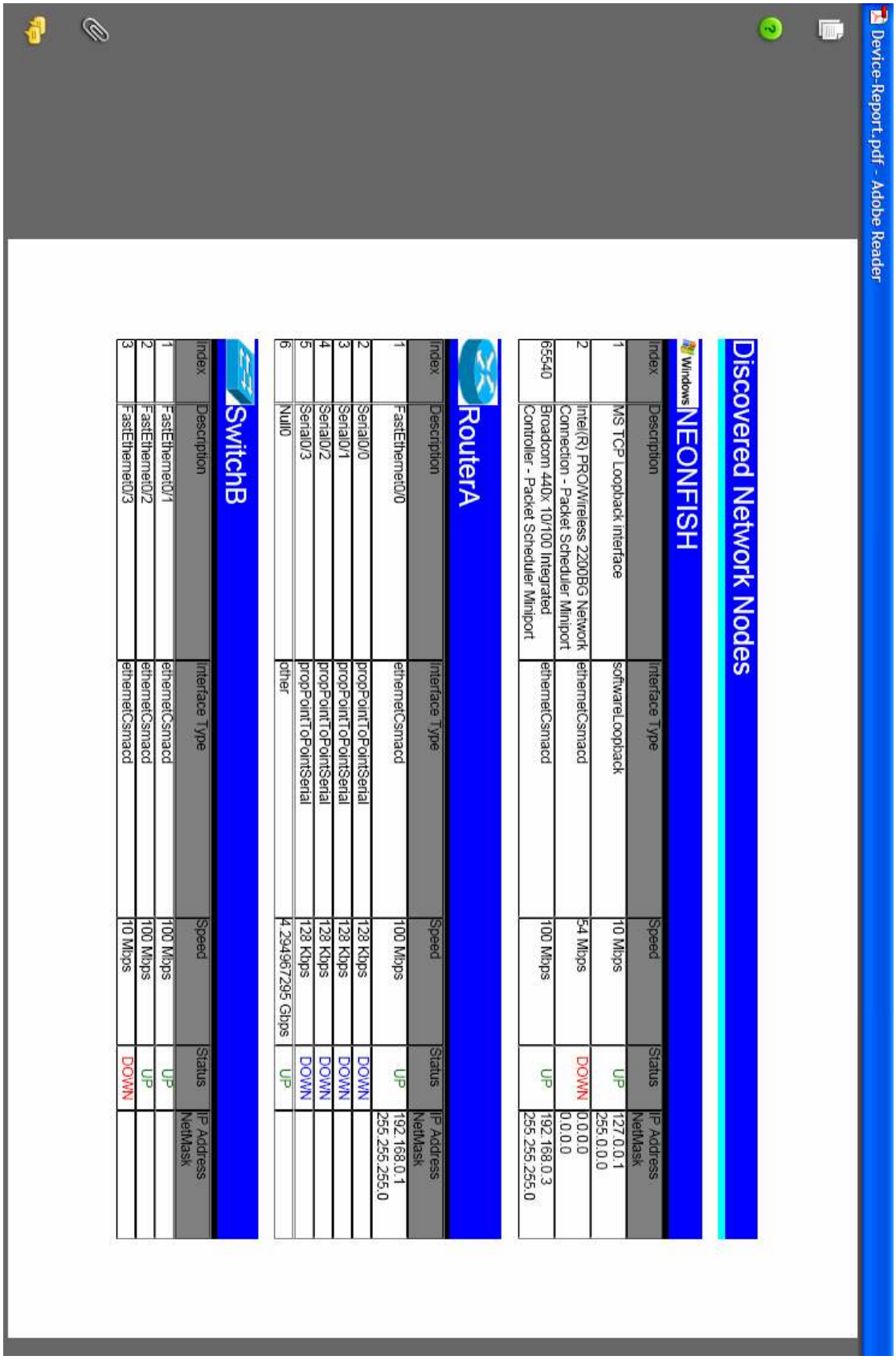


Figure 29 – PDF output of the XML data file transformed by the third .xslt file

Chapter 6 Evaluation

6.1 Introduction

This chapter gives in Section 6.2 an analysis of the implementation just concluded. Section 6.3 critiques the implementation on several points, leading to a look into ideas either not explored or discovered in the course of research in Section 6.4. The last section, 6.5, appraises the overall work against the stated thesis objectives and offers final conclusions.

6.2 Analysis of Implementation

The application realized from the implementation of the design framework was tested and measured against a topology illustrated in Figure 30. The equipment is the same as reported on in Table 3 from Appendix A. The switch was configured with its management VLAN interface on the same IP network as the devices attached to it. All routers were configured with RIPv2 as the routing protocol. All activities were conducted with no load induced on the network. The population of ARP caches was not controlled for, thereby possibly introducing additional time should any device have the need to initiate an ARP query. All interfaces were running at 100 Mbps/Full Duplex.

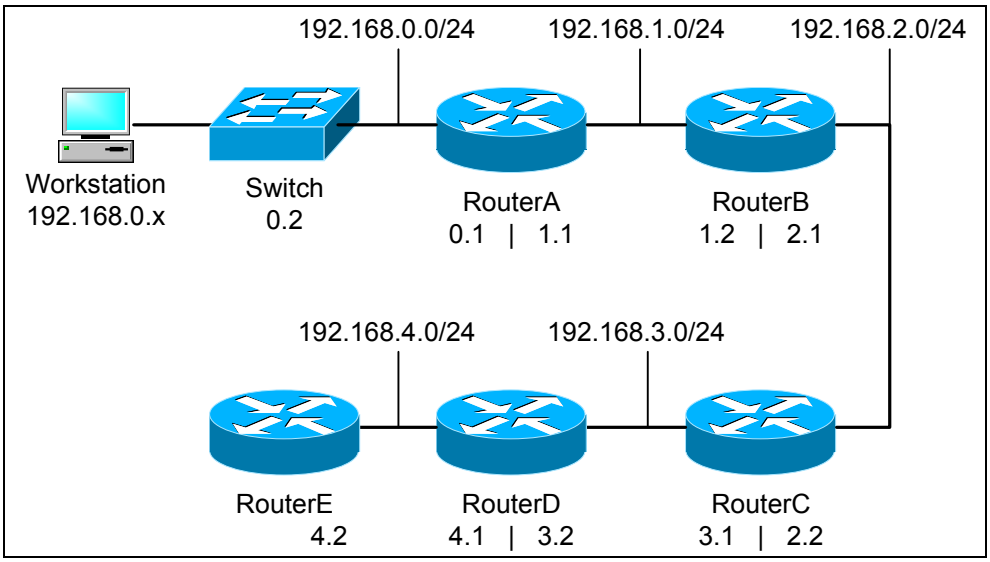


Figure 30 – Configuration of test environment

The following sub-sections report on the testing and measurement, discussing each module, in turn, which is touched by the application code (see Figure 29 at the conclusion of Chapter 5).

6.2.1 Initialization Phase

The job of the Initialization module is to provide a list of candidate nodes to be contacted by the Communication module. The selected method to generate this list in the implementation takes user input in the form of hostnames and IP addresses. Entry

by the user of hostnames or solitary IP addresses are simply added to the list. The GUI interface further allows for defining IP ranges and networks. When the user defines IP spaces, the application computes all the IP addresses in the space.

The code accomplishes computing IP addresses by setting the starting and ending values of the IP space, adding 1 to the start value, and testing if the new value is equal to the ending value. As each new value is computed it is added to an array in memory. The durations were recorded to compute the number of usable IP addresses in networks with decreasing subnet mask values. The count of usable IP addresses is then represented by $2^{(32-N)}$-2, where N is the bitmask value. The measured times are shown along two scales in Figure 31.

The left-hand decimal scale in seconds corresponds to the curve exhibiting an exponential growth pattern. This curve plots the average number of seconds to compute a range of IP addresses. Minimum and maximum values are shown with variance bars extending below and above each plot point, although at lower values they are not apparent on this scale. The hardware (an Intel Centrino 1.6 GHz processor paired with 512 MB RAM) easily handled about 250,000 values before breaking the 1 second mark. After this point, the doubling of time along with the doubling of IP addresses with each increment of N, becomes visually apparent in the graph. The plot points roughly go up to 2, 4, 8, 16, and 32 seconds. The curve stops at this final point where almost 8.4 million addresses were computed in about 30 seconds on average. The hardware's memory limitations did not allow an array much larger than 8.4 million elements to exist before the system had to resort to page-file swapping on the hard disk. Doing so dramatically affected completion times. Where the expected duration would be about 60 seconds for 16.7 million values with $N = 8$, the system was allowed to run for more than an hour without finishing.

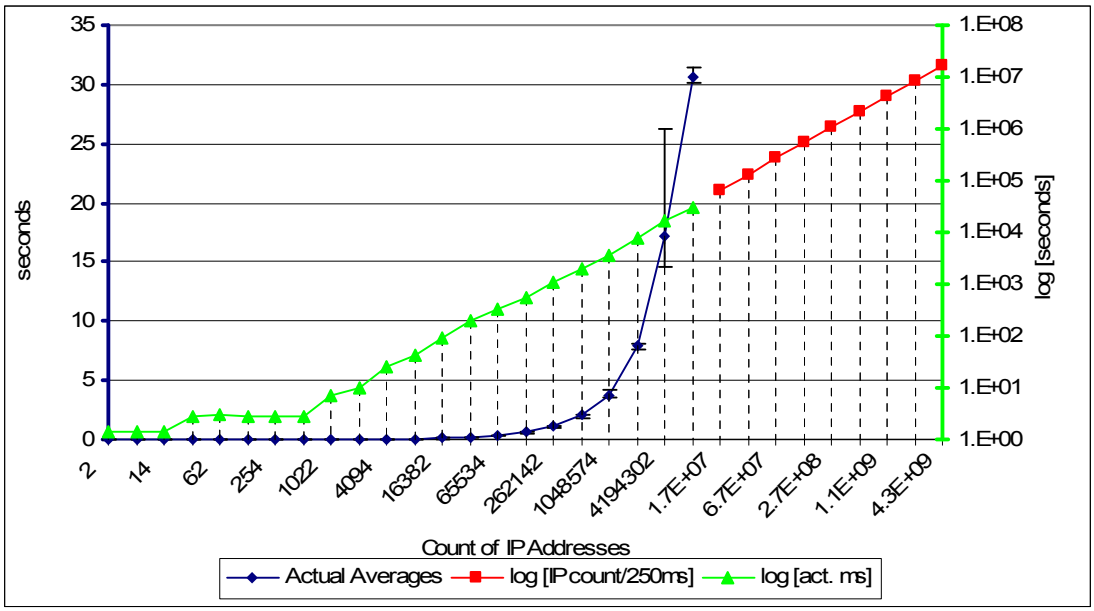


Figure 31 – Actual and estimated time to generate IP addresses

To demonstrate the full expected growth pattern for the largest possible set of values, an actual and extrapolated curve is plotted against a logarithmic scale on the right-hand side of the graph. This curve shows a linear growth pattern expected of an

exponential one mapped onto a logarithmic scale. The actual average time values are plotted first against the logarithmic scale. Where the actual measurements stopped at ~8.4 million, with $N = 9$, a break in the line is present. The remaining values to the right of the break are estimated values. The estimated values were arrived at by calculating the average processing rate from the actual measurements. Already noted above was the point that approximately 250,000 values were seen to be handled in the 1 second range. This suggests a processing rate of 250 values per millisecond. When the processor was under a sustained load (observed above ~130,000 values), the average processing rate indeed began to converge around the 250 ms range. Thusly, 250 ms was divided into the IP count to return estimated completion times. The last value on the graph represents the time to compute every possible IPv4 address at ~4.7 hours.

Although the hardware showed its own limits, it did demonstrate that a slightly aged laptop could output half the IP addresses of a traditional Class A IP space in only 30 seconds. Clearly, the majority of networks do not approach such sizes. Therefore, in its ability to define a target list, the application seems validly applicable to a wide population of networks. Additionally, this computational method imposes zero load on a network, whereas obtaining target information from DNS, route tables, and ARP caches does induce traffic to query and receive data.

6.2.2 Communication Phase

Implementation of the Communication module used only SNMP to attempt to communicate with every candidate node in the list delivered by the Initialization phase. A timer was wrapped around the section of code responsible for the initial SNMP communication. This initial SNMP Get message requested only three OID values from the target node. Once the node responded, the timer was stopped and the duration between messages recorded. A separate counter acted as a timeout by quitting listening for a response after 10 seconds. In this condition, no further attempts were made to contact the node, and the code proceeded to the next candidate node from the list of targets. If the node did respond, several more SNMP Get/Response cycles were executed to retrieve specific OID values. These were not timed. The duration values recorded only reflect the initial SNMP communication attempt. This attempt is seen as comparable to a ping in that it validates the presence of a network node while providing identifying information about that node. The average duration of the SNMP Get/Response cycle is graphed in Figure 32, as a function of the physical distance of the node from the querying workstation.

The SNMP Get/Response cycle completed very quickly with a statistical average of 21 ms; a statistical median value of 16 ms; and a statistical mode value of 0 ms (realistically in the sub-millisecond range). The timers relied on the system clock which returned values in nothing finer than milliseconds.

To compare these results with the ping tool a similar test was run using it. A simple batch script was executed sending two echo-requests, each with a timeout of 5 seconds. Failure of both would approximate the 10 second timeout used in the application code. Ping, too, only returns values as fine as one millisecond. It is able to report when lesser values are detected, but only outputs a value of "<1 ms."

Instances reported as such were assigned a value of 0.5 ms. The results of the ping tests are in Figure 33.

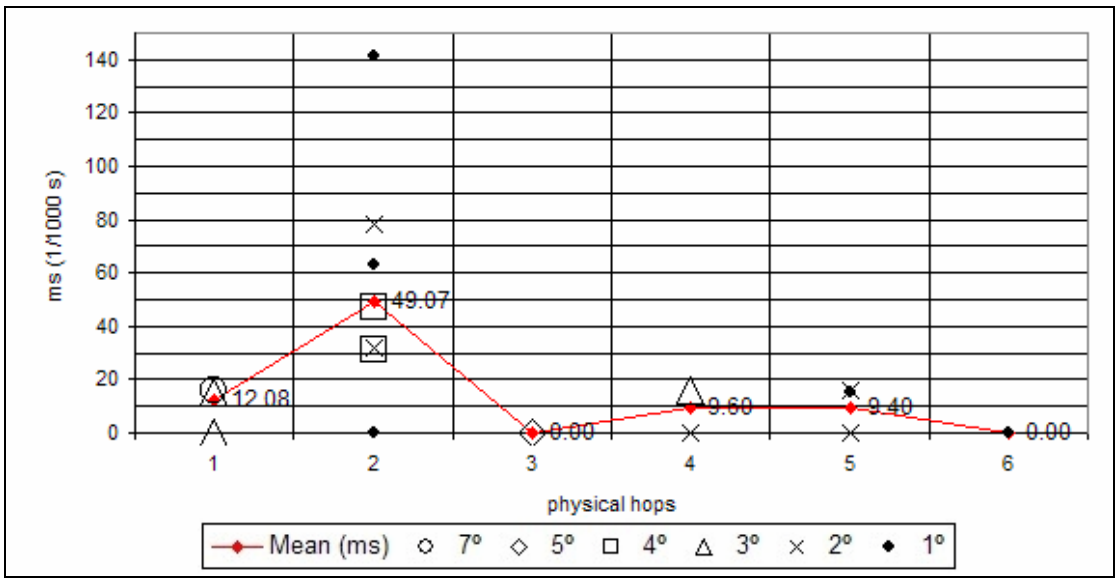


Figure 32 – Average time (ms) for the initial SNMP get/response to complete by hop distance
Values in the graph are the mean values, seen connected by a line. Distribution of values is also shown according to hop distance and degree of relative occurrence. For example, at 2 hops away, the SNMP get/response completed in 78 ms twice as often (represented by a cross) as it did in 63 ms (represented by a dot).

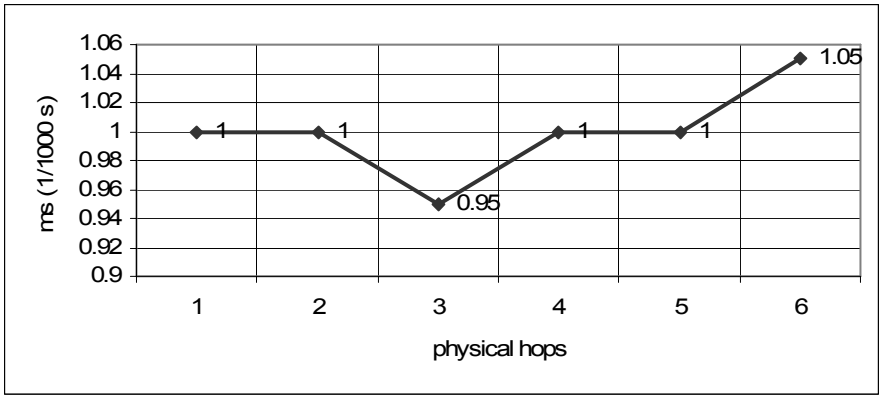


Figure 33 – Average time (ms) for round-trip pings to complete by hop distance

Of the values recorded concerning ping, several were reported as being less than 1 ms, and a few also as 2 ms. Otherwise, the value of 1 ms. was unwavering in its occurrence. Comparison of the two graphs seems to confirm ping being the quicker of the two methods. The contrary, however, will be shown in Section 6.2.4 reporting overall runtime measurements.

All of the tests conducted were positive in terms of the target node responding to either a SNMP or ping message. However, the negative case is important to consider. Failure of a node to respond manifests the worst-case scenario of having to wait a designated period of time before being able to process the next target. The implementation selected a value of 10 seconds as the timeout. Its maximum approximate runtime is therefore $10 \times N$ seconds, N being the number of list items. The work by Siamwalla, et al. (1998) states explicitly that two pings were sent to each

host, and also refers to an interval of 20 seconds per non-answering hosts. The inference is that the timeout value was 10 seconds per ping. Their estimated maximum runtime would be $20 \times N$ seconds, twice as long as our implementation.

Aside from the timeout value, the manner in which the target node list is determined has a direct affect on the full runtime. The count of items is a linear multiplier of the timeout, after all. So, there is a possibility that using complete lists of all IP values in an IP space (range or network) may result in a high rate of non-answering hosts. On the other hand, the Siamwalla team's algorithms avoid attempts to poll non-existent hosts. This is so because they rely on IP data from network sources, principally DNS and route tables, which are highly indicative of IP assignments.

Apart from awareness of the measurements of speed and overhead imposed on the network, awareness must exist about the information SNMP has to offer versus what is wished to be gathered from each node. Certain limitations were observed in the course of the implementation. As mentioned in Chapter 5, absence of the PTOPO MIB made identification of links between devices difficult compared to the ease it would provide if present. Identifying links remains the same problem it has always been of correlating data from different devices after the data has been collected. It is still not possible to collect link identity data directly from a non-proprietary source.

Likewise, the type of device, whether router, switch, or PC, is somewhat difficult to determine. Although a definitive SNMP OID may contain such data, it was not observed in the course of this work. Where this data was seen to be more obvious, but still not definitive, was in the manufacturer-specific OID's of the hierarchical SNMP tree. But knowing which manufacturer OID to query already tells a great deal about the type of chassis one is likely to confirm. Looking in the Cisco OID branch of 1.3.6.1.4.1.9 creates a high likelihood of finding the device to be some type of network gear (router or switch). Looking in Microsoft's (1.3.6.1.4.1.311) is equally as likely to be a network host. The implementation got around this stumbling block simply by examining the text contained in the system description OID (1.3.6.1.2.1.1.1). This OID is outside the manufacturer-specific branch, but such a solution is hardly ideal as the field is free-form text determined by the vendor anyway.

Difficulties in knowing a device's type point to larger issues about interpretations by manufacturers when they implement SNMP agents. Even in critically important standard OID's Microsoft and Cisco were seen to behave differently. Of critical importance to this thesis was collecting information about each interface on a device. The MIBs dealing with interfaces call for each to be assigned an index number. The index number is then used as a reference by other OID's to store nearly every piece of information about the interface, such as IP address, MAC address, etc. For external viewers, like our system, it is critical that the index numbers remain static. Indeed, this is required by the SNMP standards from the time a device boots up until it powers off or restarts. Moreover, even disabled interfaces should be indexed since they are still physically present in the chassis.

The Microsoft workstation revealed SNMP agent behaviour that failed to conform to the standard. First, interface indexes were only assigned when an interface was enabled. Disabling an interface caused the overall interface count to decrement.

Neither were index numbers assigned incrementally. Reading the OID which reports the count of interfaces did not allow knowledge of what the index values would be. An interface count of '3' resulted in index values of '1,' '2,' and '65540.' Gathering data on this third interface required reading in the index value so that additional OID's which use this value in their addressing could be read, too. These issues become a problem across time should a discovery system want to monitor information about a device's interfaces.

Behaviour on Cisco hardware revealed other interpretations of the SNMP agent. Thankfully, disabled interfaces still remained present in the interface table and the indexing occurred sequentially. A different difficulty encountered dealt with finding an interface's IP address. A table apart from the interface table stores IP information in the SNMP database. It is the IP Address Table. This table contains data organized with the IP address itself acting as the key value. One of the sub-values contains the interface index allowing data from the two tables to be correlated. This correlation is not one-to-one, though. Interfaces without IP assignments do not appear in the IP Address Table. Thus, the two main tables identifying interfaces are not of the same dimensions unless every interface is configured for IP operation.

6.2.3 Data Storage Phase

Regarding the Data Storage module, measurements were taken examining two areas. First, how long write and search operations performed by the application code could be performed. Secondly, how the size of the XML file grew as device elements were added.

The XML file structure, recall, contained three main elements. The first was the XML root called 'topology' which contains any number of child 'device' elements. These in turn contain any number of 'interface' elements. Measurements were recorded about how long it took the application code to write new interfaces and devices into the XML file. Writing a new interface was implemented using 17 lines of code to write 9 values into 9 lines of text. Every measurement of this action resulted in an output value of 0 ms. Again, the system's timer could not return fine enough times to capture the shorter duration of the event.

Measuring the time to write all interfaces of a single device, however, did produce events spanning longer than 1 ms. Figure 34 graphs this data in three manners. First, the solid diamonds plot the duration to write a device as a function of its number of interfaces. A general trend is seen that the time is proportional to the interface count. This feature is accented by the second graph, a heavy-marked curve denoting the trendline for the same data set. The trendline illustrates a linear growth pattern expected from a proportional relationship. The third graph is marked with triangles. It plots the same data as a function of distance from the polling workstation. This graph shows a spike, then dip close to the workstation, then flatter area moving further away. This is mostly explained by the fact that the switch, with its relatively high interface count of 26, was positioned adjacent to the workstation. Understanding these measurements better would require normalizing the data somehow, or using devices all with an equal number of interfaces at each hop.

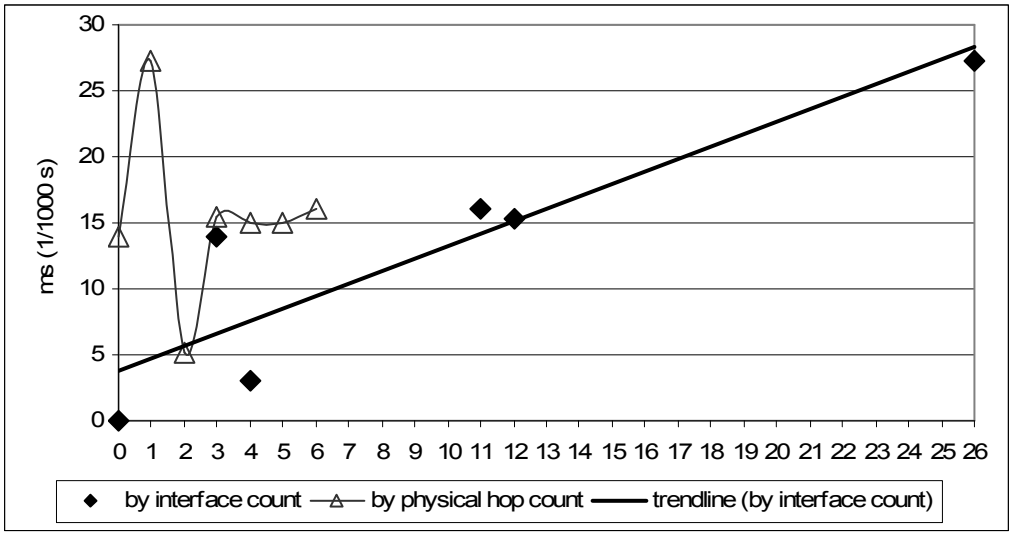


Figure 34 – Time (ms) to write a XML "device" element according to its hop distance and number of interfaces

While Figure 34 deals with creating content of the XML file, Figure 35 examines the case when stored data already exists. The application code was written such that when a device is discovered it is appended to the list of 'device' elements in the XML file. The list of 'device' elements is not ordered in any way. To qualify for being appended, the XML file is searched for any existing 'device' element with the same value for its system name. The system name value is meant to be unique in the XML file. To measure the duration of the search algorithm, the XML file was seeded with devices containing on average 13 interfaces and 104 values per device. A device entry with a known value for its system name was then positioned in the XML with a known number of preceding devices. Measurements were taken beginning at 100 preceding devices, incrementing by 100, up to 2,000. From 2,000, the increment is 1,000 stopping at 10,000 preceding device entries. Please note, then, that this range on the right end of the x-axis of Figure 35 is compressed by a factor of 10. While the left-hand side demonstrates a slowly growing, linear curve, the compression of the right-hand side allows us to view a larger range along that same curve.

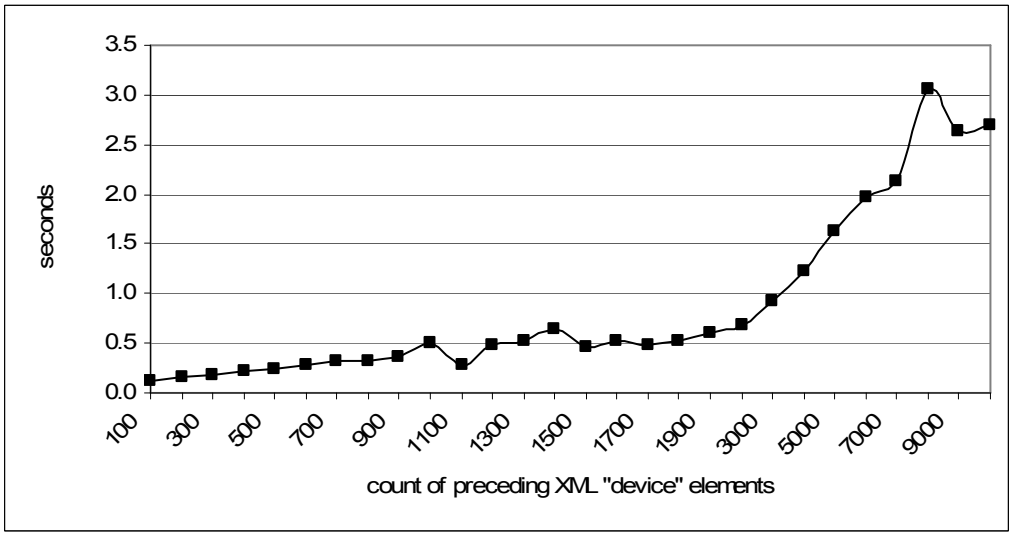


Figure 35 – Time (s) to find an existing duplicate XML "device" element

The ability to find pre-existing nodes in our XML-stored data set is a vital one. Continued polling of devices, monitoring of their status, and detection of changes could be realized through finding the original entry in the XML for comparison. That duplicate searching is achievable for large sets in less than 3 seconds per device seems a good result, but the time can add up quickly. Given a set of 1,000 nodes already present in an XML file, the time to find every node's existing entry would be the arithmetical sum from 1 to 1,000 of the search function. Represented in mathematical notation, this would be: $\sum_{1}^{1000} f(s)$. What existing function might approximate $f(s)$ is unknown, but is undoubtedly linear in nature. It would also need to account for the speed of the processor on which it is run.

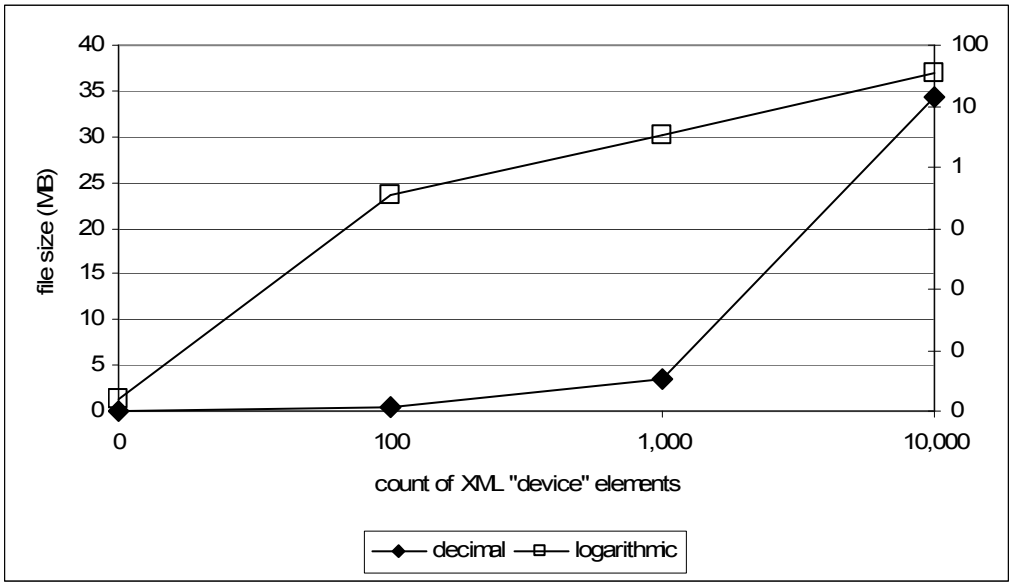


Figure 36 – Growth of the size of an XML file on a megabyte and logarithmic scale

The size of the XML file is another factor to examine. An initial file of 100 devices is about 350 KB, or .34 MB. Increasing the device count by a factor of 10 also increases the file size by a factor of 10 to 3.4 MB. A 10,000 device-count XML file was measured to be 34.3 MB. These plot points are shown in Figure 36 along both a decimal and logarithmic scale. Simple arithmetic produces a file of just over 14 TB to contain data on every one of the possible IPv4 addresses (almost 4.3 billion). The same average interface-to-device ratio of 13 was used in the sample files to obtain these values. Still, if most networks contain less than 10,000 devices, the ability to store an XML file poses no threat to physical storage mediums. The more information that is stored per device could easily grow the file, though.

6.2.4 Cumulative Measurements

The previous three sections each addressed the work performed by the application code relative to a single design module. This section adds all measurements together to illustrate total runtimes per device and per application run.

The graphs in Figure 37 continue to show a data set on a per device basis. Now the solid diamonds are plotting the total time for the application to address a single device including all SNMP communications and the information being written into the XML

file as a function of the device's interface count. A trendline is present to show the estimated curve around which durations would likely cluster. Even for a device with 26 interfaces, all communications and data storage is accomplished in just over 7/10 of a second.

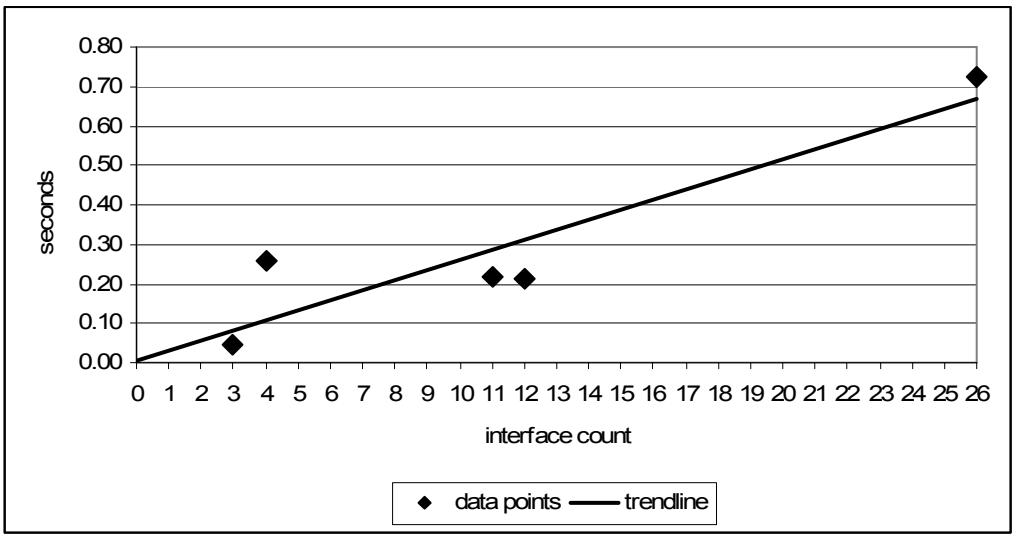


Figure 37 – Total time (s) to perform SNMP & XML operations per host by its number of interfaces

The next graphs in Figure 38 compare the SNMP-based application and the ping script referred to in Section 6.2.2. Each plot point represents the total time to execute against seven devices configured per Figure 30. For the application, this is from the point the user clicks the 'Get Data' button. From that point the application attempts communication to each node in the target list using SNMP, receives back information, processes the information in preparation for storage, and writes the information into the XML file.

For the ping script the time represents sending two pings to each of the seven hosts. Although Figure 33 demonstrated that each individual ping was answered in 1 ms, there is enough delay in the ping tool itself, and/or processing time of the batch file that the full runtime clustered around 7.5 seconds. And, although the values from Figure 32 show an individual SNMP communication completing in 21 ms on average, the rest of the application code is sufficiently fast enough to process all seven hosts in about 1.5 seconds.

The data presented in this section, especially this last graph, should bolster the argument that comparing SNMP's protocol state to a tool like ping using completeness as the measuring stick does not tell the whole story. Here, the evidence begins to suggest that the difference in speed is more significant than the difference in completeness when both the tools being measured are deployed and "on" in the network. Also, due to such speed, SNMP and XML appear to be scalable to moderately large networks, whereas manipulations of the IP stack continue to be closed off in the face of security concerns.

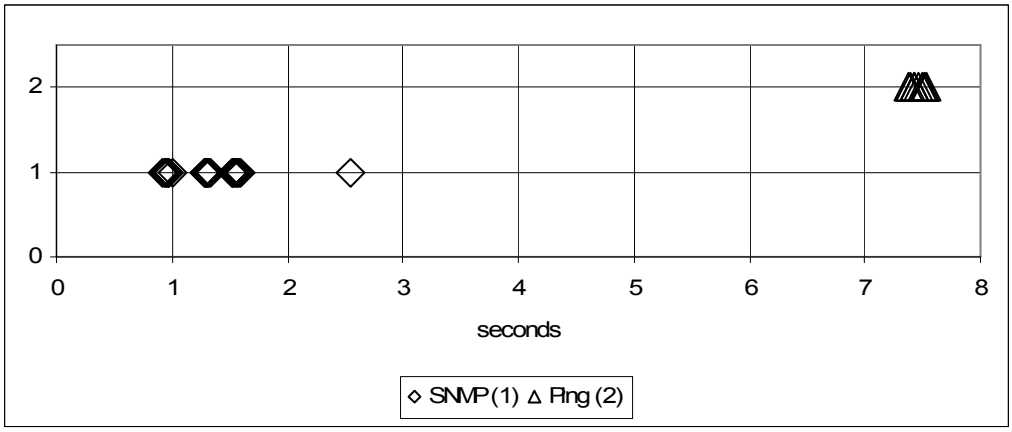


Figure 38 – Comparative distribution of total runtime of the SNMP-based application and a ping script

6.3 Critique of Implementation

There was an original aspiration to capture snapshots through time of devices' configurations. Then, series of data sets could be compared to reveal changes. The idea was introduced in Section 4.8 using hypothetical schedulers to automate periodic data collection or polling activities. The implementation did not achieve this idea. Questions arise, though, about how to store and when to compare the data. Data might be compared at collection time as was done with searching for existing system names in the XML file. Any value or attribute might be compared in the same way. Were a comparison to reveal a variance between a past and present value, the application must somehow preserve both. One option would be to add the new value to the existing XML. This choice seems to ask that the data be time-stamped. Alternatively, comparison of data could occur after collection. This choice requires the newly acquired data to be stored, either in a new XML file or in the existing XML file. Again, it seems necessary to look to implementing document versioning or time-stamping so that later comparisons will be understood in a chronological context.

Although the ideas of this thesis propose a heavy shift towards automating documentation production, this is not meant to wholly remove the humans, that is the system administrators, which are involved. Essentially there is an effort to try to remain mindful of "Automation Irony," described thusly:

> Automation usually addresses the easy tasks for humans, leaving to the operator the complex, rare tasks that were not successfully automated… The irony is that automation reduces the chance for operators to get hands-on control experience… The challenge is to design systems that are synergistic with human operators (Patterson, et al., 2002, p. 3-4).

The modular framework design attempted to recognize the continuing importance of human involvement by allowing degrees of interaction or control around each module, as well as viewing of the output and work of the modules.

Indeed, the human factor is of an utmost importance. After all, the overarching motivation is to produce visual representations of network topologies. The question

of how to visualize data moves easily into areas of HCI where some less technical questions need addressing. Mainly, we refer to questions of aesthetics. It is simple enough to produce visualized documents, but effective ones will have to be mindful of their aesthetic appeal. While doing so, documents should deliver sufficient information regarding their purpose, but not cause information overload for the viewers. Some work related to spatial layout of topological information is available in Au, et al. (2004), and Buchanan and Zellweger (2005). The work of this thesis did achieve some visualized outputs, but did not go so far as tackling spatial layout or aesthetic challenges.

There was a special interest in using SNMP due to the possibilities of the PTOPO MIB delivering valuable, pre-computed link information. The version of SNMP employed was v2c. This version still contains flaws and remains susceptible to several types of attacks. So, despite SNMP v2c being the *de facto* version used in today's networks, it is looked upon with a certain amount of suspicion by the same administrators who deploy it. The framework introduced in Chapter 4 does not call for a specific version of SNMP. The implementation was forced to use SNMP v2c by the capabilities of the Cisco IOS on the available switches and routers, and by the workstation's Windows OS. Further work would be very much interested in an SNMP architecture employing v3 agents and security.

6.4 Suggestions for Future Work

Reproduce work by Siamwalla Team
The reading and research undertaken while exploring network discovery, network management systems, open standard tools, and document production point to several areas worthy of further investigation. The first of these would be a reproduction of the work by Siamwalla, et al. (1998). As previously discussed, changes in technology and management systems are likely to reveal modern networks that behave differently when examined by the Siamwalla team's algorithms. Their conclusions that SNMP was an ineffective tool for discovery might lose ground since the better performing tools are now subject to more security risks. SNMP is also more common in modern operating systems along with the uses of management systems. Hurdles to implement SNMP across a network, and not just on routers, are much lower than a decade ago.

Link Discovery
While device discovery might be considered fairly mature, link discovery is still hobbled by the need to re-compute relationships between devices. The PTOPO MIB theoretically solved this issue when it became defined in 2000. It faltered, though, without a standard protocol defined for populating the MIB space. Industry has largely ignored implementing PTOPO, preferring their own proprietary protocols and MIB spaces (Cisco's CDP is a prime example). So, although little work is to be had in exploiting easily available data from PTOPO, continued observation of link discovery technology is warranted. This is mainly due to the adoption of the Link Layer Discovery Protocol in 2005. LLDP was ratified as IEEE standard 802.1AB (IEEE, 2005). LLDP is supposed to finally provide a standard protocol for link discovery. Perhaps as exciting, LLDP is also supposed to populate its data into its own SNMP MIB space while simultaneously updating the PTOPO MIB. If LLDP takes hold, PTOPO might get widespread implementation too. A result such as this

would provide SNMP-accessible, pre-computed data on both physical links and logical, Layer 2 links.

Visualisation

Visualising a data set is perhaps more important than the data set itself. Spreadsheets certainly tell less compelling stories than coloured, iconographic maps when it comes to networks. But, producing a well laid-out map is its own challenge. An individual person drafting a map manually is able to choose object positioning to maximize aesthetic design, as well as clearly relate the object's connections, context, and relationships. How can the same output be accomplished automatically? Work by Au, et al. (2004), and Buchanan and Zellweger (2005) provides a technical starting point. Presentations by Jeff Han[9] and Hans Rosling[10] provide inspiration.

GIS & Visualised Temporal Sequencing

Visualisation is a building block to further contextualization of data. A map of a network's physical topology carries even more information when presented in the context of its physical environment, such as a building floor plan. A WAN map might be more effective placed against a geographical map of cities and countries. The area of Geographic Information Systems seems it might offer synergies with presenting data about a network. GIS mainly deals with layering multiple data sets on top of the same context. Imagine a building floor plan with subsets of the network layered on by the choice of the end-user. All switches could be selected for to see the STP topology. Or, all routers selected to show the IP layout.

Contextualization can take other forms. Important to network management (especially fault management) is understanding changes through time. If we have data on a topology captured at intervals, it would seem we have the frames of a movie showing an evolving network. Since XML technologies were of special interest for data storage and document production purposes, they might offer a way to produce time-contextualized presentations. Indeed there is an XML language called SMIL (an acronym for Synchronized Multimedia Integration Language, and pronounced "smile") that "describes multimedia presentations. It defines timing mark-up, layout mark-up, animations, visual transitions, and media embedding, among other things" (Wikipedia, 2007b).

XML Language for Network Topologies

The most interesting idea generated from this research is that of an XML language for describing network topologies. This concept was largely inspired by a cursory review of CML. CML is the Chemical Markup Language, a standard language to describe molecules. CML interpreters take only CML text and produce three-dimensional renderings of the molecules described. When considered, molecules are substantially similar to a network topology. Atoms are like network nodes. Chemical bonds are like network links. There are rules about how atoms can bond just like there are rules about how routers, switches, and hosts can interconnect. The vision of a language for describing networks coupled with a visualization platform would seem to be a

[9] See Han (2005) for his paper, or his TED 2006 presentation at http://www.ted.com/index.php/talks/view/id/65.

[10] Hans Rosling relied upon Trendalyzer, software now owned by Google, but developed by Gapminder. His presentation can be seen at http://www.ted.com/index.php/talks/view/id/92, while the software can be found at http://tools.google.com/gapminder/.

powerful one. Different scales of a network could be viewed without a need to switch between sources from a library of documents. Instead, one could zoom in or out to see small-scale LAN segments or large scale WAN connections. This approach is already common in presenting geographical data (think Google Maps). It seems a small leap to apply this presentation method to other kinds of maps, and specifically to maps of network topologies.

6.5 Conclusion

This chapter put forth measurements of the proof-of-concept application. These measurements showed the application to be faster compared to the ping tool while returning significantly more information about each network node. Also, the application is projected to be applicable to networks ranging from the very small to moderately large. This is concluded based on several factors. The application has the ability to generate an initial target list of nodes numbering in the thousands, or even millions, in under 15 seconds, and usually much less. It is also predicted to be able to store large amounts of information well inside the limits of today's storage mediums. Aside from these technical data, several critiques of the implementation and suggestions on future work were offered.

Like the review of the implementation's success, the overall work of the thesis must be considered according to its scope and goals. The objectives serve as the agenda for this revision. Here they are as stated in Chapter 1:

1. To conduct a critical review of existing literature relating to network management and topology discovery systems.
2. To design a novel system framework for producing network documentation.
3. To use a non-proprietary protocol to retrieve information from network devices.
4. To store network information in an accessible, portable format.
5. To automatically produce network documentation as both paged and non-paged media.

Objective 1 was accomplished and reported on in both Chapters 2 and 3. Exploration of background issues in system administration and network management paradigms, along with an understanding of previous research into topology discovery systems, was key to accomplishing the remaining objectives. The review of literature informed the ideas to be challenged and validated. It helped form the questions that the design and implementation stages attempted to answer and investigate.

Chapter 4 dealt with Objective 2. Therein was demonstrated the evolution of a design built first from motivations that were responses to the issues discovered in the literature review. This simple framework was then gradually completed with specific technological answers to the questions of operations. The method of presenting the framework by building up its parts is deliberate to show it is arrived at though synthesis and not by reliance on existing paradigms. In this way it is a novel solution to the problem set at hand.

Meeting objective 3 required first a choice of protocol followed by actually using that protocol for communication with network nodes to retrieve data. The protocol chosen

was SNMP, as justified in Section 4.5. Its main strength is that it is a well-established, non-proprietary, open standard. It is also a widely implemented protocol by many hardware and software vendors. Through a borrowed programming library, the implementation successfully sent requests to network devices for specific data stored by the devices' SNMP agents. Those devices appropriately returned information carried by SNMP protocol packets to the implemented application. By employing SNMP the application is not constrained to information about a single OSI layer. Instead, all the information in a device's agent database is available. These databases' information will cover the primary layers of interest (Layers 1 to 3), if not all OSI layers. Analysis of the implementation found SNMP to be fast at information retrieval. Its continuing challenge is to be deployed widely across an enterprise, and to have its secure versions adopted over legacy ones.

Objective 4 was achieved through using XML documents as data storage mediums. XML files are both accessible and portable. They are accessible in the same way that any stand-alone file is. They can be shared through file systems, published to web sites, subjected to document control and versioning, and emailed as attachments. Furthermore, XML files are readable by a variety of tools from simple text editors to web browsers to specialized integrated development environments. XML content is also intended to be machine and human readable. The fact that it is both makes XML highly portable, too. Content is self-describing through names of elements and attributes. It is easily parsed by machines by recognition of elements, tags, and the hierarchical structure which emerges from their organization. These properties make XML portable because it is simple to read and in a ready state for transformation to other formats.

The use of XML to store network node information found it to be an easy fit to the nature of the data. XML naturally has a hierarchical, tree-like structure which is well-suited to represent network devices. Any device can be seen as a single container of one or more interfaces. These interfaces are then containers of their own descriptive information, or possibly even virtual interfaces. XML elements, too, act as containers of both data and other elements. They naturally nest inside one another with no limitations as to quantity of elements. The grammar chosen in our XML document demonstrates this concept. It is further likely that XML's nature would be equally appropriate for representing the tree-like structure of network topologies. The analysis and projections of the actual XML data stores realized suggests their size and parsing time might be appropriate for even moderately large networks. Certainly, there is a potential for efficiency gains in smaller networks where XML data storage could supplant the need for deploying, running, and supporting a complex database solution.

Objective 5 was of the utmost importance to the aim of the thesis. Three example documents were produced in the implementation phase. Two were non-paged, HTML documents suitable primarily for consumption through web browsers. The third was a PDF document. It was output as its own file apart from the source data XML document, unlike the HTML outputs. As its own file, it could be shared, emailed, etc. Since it is a paged media, the PDF document could be viewed with a PDF reader or printed, with no discrepancy of layout between the soft and hard mediums. All of these outputs were constructed automatically by transforming the XML data store using XSL.

XSL is extremely powerful for reorganizing data and/or transforming it into alternate languages. Due to its initial design goals, XSL is very adept at producing paged media, like PDF. And, owing to the co-opting of XSLT by the web development community, it is exceptional at outputting XHTML, among other XML languages.

By meeting each of the objectives, the aim is largely accomplished. Several open standard technologies were explored to discover ones both capable and appropriate to fulfil a functional role in the design of a system. Those selected were SNMP, XML, and the XSL family. The system into which they were inserted was one designed to automatically produce documentation about a network. The marrying of each of these technologies was then demonstrated through a proof-of-concept application. Automation was accomplished using two programming vectors. First, custom code was written to employ SNMP for data gathering, and to store that data in XML documents. Second, customized XSLT transforms were written to convert the network data into both screen-based and paper-based formats (HTML and PDF).

The aim of producing such a system as the one described is but one answer to the problem of documenting IT systems. Investigation of producing documentation reveals it to be a necessary activity in both theoretical and practical management systems. In practice, though, it is traditionally a manual process, meaning producing documentation is time-consuming, expensive, or ignored altogether. Looking to do so in a fast, inexpensive manner led this solution to rely upon open standard tools. This results in several strengths. The solution is not affiliated to a specific vendor making it applicable to any network. It should also remain adaptable, even in the face of evolving standards or networking environments. Finally, it produces network documentation – a highly valuable resource for administration of modern, advanced computer networks.

Appendix A – Implementation Resources

Resource	Sourced from	Purpose	Type
Microsoft Visual C# 2005 Express Edition	Microsoft Corporation	Software Development Environment	Primary
C#.NET	Microsoft Corporation	Programming Language	Primary
.NET Framework 2.0	Microsoft Corporation	Runtime Environment	Primary
snmp.dll v0.3.3f	Malcolm Crowe, University of Paisley, 2003 (http://cis.paisley.ac.uk/crow-ci0/)	Runtime for SNMP Protocol	Primary
Apache FOP 0.92 beta	http://xmlgraphics.apache.org/fop/	XSL-FO processor	Primary
Cisco Catalyst 2950 switches & Cisco 2600-series routers	Napier University School of Computing Networking laboratory (C27)	Hardware for SNMP polling and mock network	Primary
Computer (Intel Centrino 1.6 GHz, 512 MB RAM, Windows XP SP2)	Self provided	Development and testing platform	Primary
MIB Navigator	http://sourceforge.net/projects/mibnavigator/	Utility for browsing SNMP agent trees	Secondary
MIBToXML Converter	http://sourceforge.net/projects/mibnavigator/	Utility to convert standard MIB files for use in MIB Navigator	Secondary
FreeSNMP	NsaSoft (http://www.nsauditor.com/network_tools/snmp_mib_browser.html)	Utility for browsing SNMP agent trees	Secondary
Treebeard v0.9.0	http://treebeard.sourceforge.net/	XSLT Integrated Development Environment	Secondary

Table 3 – Resources utilized in the implementation

Appendix B – SNMP Basics

There are many good texts providing a treatment of SNMP. Black (1995) was consulted to verify the details offered here. The main components of the Simple Network Management Protocol architecture are:

- Managed Systems – any device running a management agent.
- Management Agents – software for handling SNMP communications and the local information store.
- Management Station – a central system for initiating management activities via SNMP, and for receiving alarms from managed systems.

A management agent runs as a service, or daemon, on each managed system. It maintains information about the managed system in a local Management Information Base (MIB) (Figure 39).

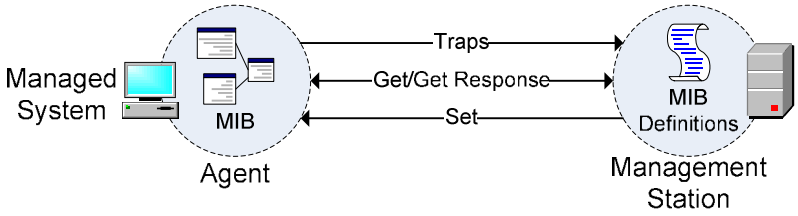


Figure 39 – Architecture of SNMP

MIBs are hierarchically organized. Elements in a MIB can be referenced by a canonical name, or by their Object Identifier (OID) (Figure 40). The full name or OID of a given element is built by naming each branch in the hierarchy starting from the root. In order to translate between canonical names and OIDs, MIB definition files are stored on the Management Station. The station's management software consults the appropriate MIB definition file corresponding to a given OID.

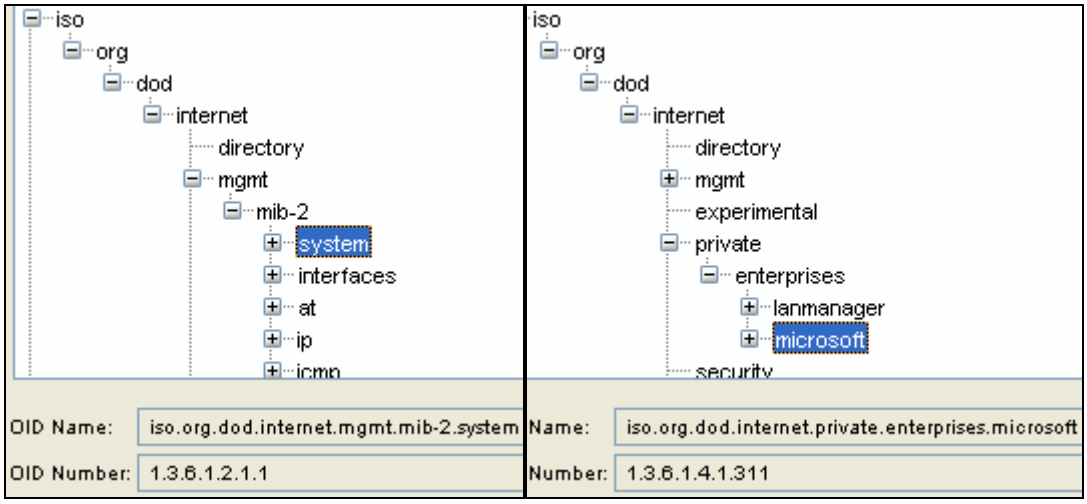


Figure 40 – MIB Hierarchy showing OID names and numbers

The left pane shows the MIB branch "system," which contains general information such as "sysDescr" and "sysName." The right pane shows the vendor-specific branch, highlighting "microsoft." Vendors have the authority to define MIBs inside their designated branch. Microsoft's OID number is 311, while Cisco's is 9.

Appendix C – XML Basics

The Extensible Markup Language is a generic markup language standard recommended by the World Wide Web Consortium (W3C). It is the basis for all other XML languages, or applications. Examples include XHTML, SVG, CSS, CML, MathML, SOAP, WSDL, UDDI, and many more. XML is designed for the storage, retrieval, and validation of data, and document transforming and formatting. This appendix was informed by Ray (2003), Pawson (2002), and Tidwell (2001), and is cited where appropriate.

C.1 XML

The basic component of XML is the tag. Tags must open and close, may contain data, and may have attributes. Tag and attribute names can be almost any string of alphanumeric characters, although there are some restrictions. Here are some basic examples of tag syntax:

```
<tag1>data</tag1>
<tag2 attribute-A=”value” attribute-B=”value”>data</tag2>
<tag3 attribute-C=’value’/>
```

An XML document is split into two sections. They are the document prologue and the document element. Their functions are analogous to an HTML document’s header and body, respectively. The document prologue consists of special tags called declarations. Declarations are denoted through the use of special characters as opening delimiters (`<?`, `<!`), along with some cases of special syntax. The most common declaration is an XML declaration which informs an XML processor about the document.

Analogous to the body, the document element (also called the root element) of the XML document begins with the first tag element lacking the use of special opening delimiters. The document element ends when this element is closed. Elements can contain other elements in a hierarchy. The hierarchical organization produces data, described through their tags and attributes, organized in a tree-like structure.

Important to the use of XSLT and XSL-FO are XML namespaces. Namespaces allow elements to belong to different vocabularies in the same document. Were a namespace given the name “`napier`,” elements become members by prefixing them with the name. So, a “`napier:student`” element is different from a regular “`student`” element. Namespaces are defined like an attribute, but must adhere to the syntax of: `Xmlns:`*`name`*`=”`*`URI`*`”`.

Here is a very basic, yet valid, XML document showing the document prologue, definition and use of namespaces, the document element, and hierarchical structure:

```
<?xml version="1.0" encoding="utf-8"?>
<address-book xmlns:oc=”http://occupant.com”
xmlns:ad=”http://address.com”>
  <house descr=”Holyrood Palace”>
    <oc:person>Queen Elizabeth II</oc:person>
    <ad:number> Canongate, The Royal Mile, EH8 8DX</ad:number>
```

```
    <number>5 Horse Wynd</number>
  </house>
</address-book>
```

C.2 XSLT

The Extensible Stylesheet Language for Transformations is another W3C recommendation. XSLT is a language for transforming XML documents into other formats, usually still XML. XSLT is itself XML. An XSLT document contains iterative and recursive rules that pattern-match data in source documents. An XSLT processor interprets the source document according to the rules and outputs a new document according to them.

One way to apply the XSLT document to transform an XML document, the XML document itself can contain a declaration as to what, if any, stylesheets should be applied to it. Like most declarations, this occurs in the document prologue. The declaration is a processing instruction with the name `xml-stylesheet`. It is able to declare the location of the stylesheet and the stylesheet's MIME[11] type. The `xml-stylesheet` element can equally be used to call Cascading Style Sheets (CSS). Here is an example of a declaration to an XSLT transform located in a directory relative to the XML document's location named 'web':

```
<?xml-stylesheet type='text/xsl' href='web/tform.xslt'?>
```

Now, the XSLT document must be built. The root element of an XSLT tree, at its most basic, opens a `stylesheet` element in the `xsl` namespace, states the XSLT version, and declares the `xsl` namespace. This is vitally important as an XSLT processor interprets the `xsl` namespace elements. These "elements control the process. Any elements or attributes not in that namespace… will be interpreted as data to be output in the result tree" (Ray, 2003, p. 232).

Next, pattern-matching rules are defined using instances of the `template` element from the `xsl` namespace. An attribute called `match` is then employed to define the pattern to be searched for in the source XML document. The values of this and other attributes conform to XPath rules and syntax. (See either Ray (2003) or Tidwell (2001) for more on XPath.) Everything that follows between the opening and closing `template` element tags comprises the rule, or set of actions, that the XSLT processor will perform. Here is a simple stylesheet that builds a web page based on our earlier address book example XML document:

```
<xsl:stylesheet
      version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
   <xsl:template match="address-book">
      <html>
         <head>
            <title>My Address Book</title>
         </head>
         <body>
            <h1>Entries in the address book</h1>
            <xsl:apply-templates/>
         </body>
```

[11] Multipurpose Internet Mail Extensions

```
        </html>
      </xsl:template>
    </xsl:stylesheet>
```

Note that the resulting HTML document contains only one line of text in its body. To add more content, more `xsl:template` rules must exist to handle other types of elements from the source XML document. Additional templates would also be child nodes to the root element, but sibling nodes of other `xsl:template` elements in the XSLT document's hierarchy. To reference the sibling templates, the element `xsl:apply-templates` is used.

The `xsl:apply-templates` element is a good example of an XSLT processor's iterative nature. At the point this element occurs, the instruction tells the processor to proceed to interpret all further defined `xsl:template` elements. It iterates each rule. Each rule defines its own pattern to be matched along with actions. Each instance of a found pattern is itself iterated until the given `xsl:template` element is exhausted. So, to handle the `house` elements in the XML address book, an additional XSLT rule might output the name and address of entry like this:

```
<xsl:template match="house">
   <xsl:value-of select="oc:person"></xsl:value-of>
   <xsl:value-of select="ad:number"></xsl:value-of>
   <xsl:value-of select="number"></xsl:value-of>
</xsl:template>
```

C.3 XSL-FO

Formatting Objects is the third component of the original W3C XSL recommended standard. An FO document is an XML syntax arrived at through XSLT+XPath transformation of an XML source document. The FO hierarchical tree is a series of containers for discreet portions of content, each then associated directly with page layout, formatting, and presentational characteristics. FO documents are rarely needed except as they are passed to an FO processor. The processor is driven by the content to produce paged media. This is as opposed to paged media that is layout-driven, such as a newspaper. Generally, there is the most interest in outputs of PDF and PostScript as the paged media types. PDF is easily printable, or shareable in its electronic form. PostScript output can be sent directly to print devices for hard copy production.

FO follows a model built around areas. The first area defined is the page itself. Further sub-areas will then be defined within the page's dimensions for placing content (text, images, etc.). The geometry of pages and their sequences are setup in the required `fo:layout-master-set` element (note the use of the `fo` namespace). The layout master has children elements (hence, a set) of page masters and sequence masters. Page masters prescribe the height, width, margins, orientation, text direction, flow direction, and so on. Sequence masters setup the order in which pages can occur, like when odd and even pages in a book require distinct layouts. Page and sequence masters are templates that are later invoked through a globally unique naming reference.

After the stylistic parameters have been established, a page is instantiated with a `fo:page-sequence` element. This element calls the unique name of a page/sequence

master combination thereby setting the geometrical environment into which a flow of content will be "poured." A flow is started with the `fo:flow` element. The flow must declare and attach itself, via the `flow-name` attribute, to one of the pre-defined regions of a page. Page regions would be the body, or left margin, for example. Child elements' content will then appear in the page region defined.

In Western layouts, text flows from top to bottom and left to right. An established FO flow will fill a page region in the same fashion using blocks and inlines. A block is a rectangular area containing content. The simplest usage would be to use a block for each paragraph of text. An inline element can be used to modify content, as its name implies, without having to split the content among multiple areas. This is the same concept of wrapping a single word in HTML inside `<a href="">` and `</a>` elements to create a hyperlink. An inline element can be used in the same way to, say, modify the font, colour, etc. of the text it encloses.

XSL-FO is very much more complicated and powerful than the very basic concepts presented here. However, if we imagine an XSLT transform were applied to our simple XML address book from above, it would be simple to construct the following FO tree for delivery to an FO processor:

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

 <fo:layout-master-set>
  <fo:simple-page-master margin-right="1in" margin-left="1in"
      margin-bottom="1in" margin-top="1in" page-width="11in"
      page-height="8.5in" master-name="simple">
   <fo:region-body margin-top="0in"/>
  </fo:simple-page-master>
 </fo:layout-master-set>

 <fo:page-sequence master-reference="simple">
  <fo:flow flow-name="xsl-region-body">
   <fo:block space-after="12pt" background-color="black"
      color="white" font-family="sans-serif" font-size="12pt">
    <fo:inline font-style="italic">Holyrood Palace</fo:inline>
     <nl/>
      Queen Elizabeth II, Canongate, The Royal Mile, EH8 8DX
   </fo:block>
  </fo:flow>
 </fo:page-sequence>
</fo:root>
```

This FO tree demonstrates most of the basic concepts presented above. An FO processor would output an 8½ by 11 inch, landscape oriented page with text very much like this:

Holyrood Palace
Queen Elizabeth II, Canongate, The Royal Mile, EH8 8DX

Appendix D – Application Code

D.1 System Generated Code

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using Snmp;
using X690;
using System.Xml;
using System.Text;
using System.IO;
using System.Threading;

namespace WinSNMPtoXML
{
  /// <summary> ...
  public class Form1 : System.Windows.Forms.Form
  {
    private System.Windows.Forms.LinkLabel llblViewXML;
    private System.Windows.Forms.ListBox lstOutputStatus;
    private System.Windows.Forms.LinkLabel llblGenDoc;
    private System.Windows.Forms.LinkLabel llblGetData;
    private System.Windows.Forms.TextBox txtStartOctet1;
    private System.Windows.Forms.Label lblCommString;
    private System.Windows.Forms.TextBox txtPassword;
    private System.Windows.Forms.Label lblHostname;
    private System.Windows.Forms.TextBox txtHostname;
    private System.Windows.Forms.Label lblStartIP;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.TextBox txtStartOctet2;
    private System.Windows.Forms.Label label2;
    private System.Windows.Forms.TextBox txtStartOctet3;
    private System.Windows.Forms.Label label3;
    private System.Windows.Forms.TextBox txtStartOctet4;
    private System.Windows.Forms.TextBox txtEndOctet4;
    private System.Windows.Forms.Label label4;
    private System.Windows.Forms.TextBox txtEndOctet3;
    private System.Windows.Forms.Label label5;
    private System.Windows.Forms.TextBox txtEndOctet2;
    private System.Windows.Forms.Label label6;
    private System.Windows.Forms.Label lblEndIP;
    private System.Windows.Forms.TextBox txtEndOctet1;
    private System.Windows.Forms.Label label7;
    private System.Windows.Forms.Label label8;
    private System.Windows.Forms.Label label9;
    private System.Windows.Forms.Label label10;
    private System.Windows.Forms.Label lblBitmask;
    private System.Windows.Forms.TextBox txtBitmask;
    private System.Windows.Forms.Label label11;
    private System.Windows.Forms.Label label12;
    private System.Windows.Forms.TextBox txtMask4;
    private System.Windows.Forms.TextBox txtMask3;
    private System.Windows.Forms.TextBox txtMask2;
    private System.Windows.Forms.TextBox txtMask1;
```

```
    private System.Windows.Forms.Label lblTip;
    private System.Windows.Forms.Label lblNote;
    private System.Windows.Forms.Label lblHostnameDir;
    private System.Windows.Forms.RadioButton rbtnHost;
    private System.Windows.Forms.RadioButton rbtnRange;
    private System.Windows.Forms.RadioButton rbtnNetwork;
    private System.Windows.Forms.Label lblPassword;
    private System.Windows.Forms.LinkLabel llblCreateList;
    private System.Windows.Forms.LinkLabel llblViewList;
    private System.Windows.Forms.Label lblListCount;
    private Label lblArrow;
    /// <summary> ...
    private System.ComponentModel.Container components = null;

    public Form1()...

    protected override void Dispose( bool disposing )...

    #region Windows Form Designer generated code
    /// <summary> ...
    private void InitializeComponent()...
    #endregion

    [STAThread]
    static void Main()
    {
      Application.Run(new Form1());
    }
```

D.2 Global Variables

```
    private const string xmlFile = "topology.xml";
    private const string ifTypeFile="ifTypes.csv";
    ArrayList IPAR;
    string[,] ifTypesAR;
    double runtime;
```

D.3 XML File Initialization

```
    public void DocTest ()
    {
      if (!(File.Exists(xmlFile)))        //test for/creates XML file
      {
        lstOutputStatus.Items.Add("XML file does not exist!
Creating...");
        XmlTextWriter docmaker = null;
        docmaker = new XmlTextWriter (xmlFile, Encoding.UTF8);
        try
        {
          docmaker.Formatting = Formatting.Indented;
          docmaker.Indentation= 6;
          docmaker.Namespaces = false;
          docmaker.WriteStartDocument();
          docmaker.WriteProcessingInstruction("xml-stylesheet",
"type='text/xsl' href='web-device-report.xslt'");
          docmaker.WriteStartElement("topology", "");
          docmaker.WriteComment("Start of device listing");
          docmaker.WriteEndElement();
          docmaker.Flush();
          lstOutputStatus.Items.Add("Created topology.xml.");
          llblViewXML.Visible=true;
```

```
          llblGenDoc.Visible=true;
        }
        catch(Exception e)
        {lstOutputStatus.Items.Add("Error creating XML file:
"+e.ToString()+".");}
        finally
        {
          if (docmaker != null)
          {docmaker.Close();}
        }
      }
      else
      {
        lstOutputStatus.Items.Add("topology.xml already exists.");
        llblViewXML.Visible=true;
        llblGenDoc.Visible=true;
      }
    }
```

D.4 Loader for IANA Interface Type Lookup

```
    public string[,] LoadifTypes()
    {
      //loads .csv file with interface type codes & descriptions
      //per the IANA ifType-MIB, revision September 25, 2006
      string[] flatstrAR = File.ReadAllLines(ifTypeFile);
      string[,] dimstrAR = new string[flatstrAR.Length,2];
      int i = 0;
      foreach (string s in flatstrAR)
      {
        string[] tempS=s.Split(',');
        dimstrAR[i, 0] = tempS[0];
        dimstrAR[i, 1] = tempS[1];
        i++;
      }
      return dimstrAR;
    }
```

D.5 Class to Create IP and Hostname List

```
    public ArrayList CreateHostList ()
    {
      //populates array with all hosts & IPs indicated by user
      ArrayList v = new ArrayList();
      string text = null;
      string dot = ".";
      int a= new int();
      int b= new int();
      int c= new int();
      int d= new int();
      //check for non-integer condition
      try
      {
        a=Convert.ToInt32(txtStartOctet1.Text);
        b=Convert.ToInt32(txtStartOctet2.Text);
        c=Convert.ToInt32(txtStartOctet3.Text);
        d=Convert.ToInt32(txtStartOctet4.Text);
      }
      catch
      {
        lstOutputStatus.Items.Add("Host/Start/Network IP: check for
non-integer condition.");
```

```
        goto End;
      }
      //check for blank hostname field
      if (txtHostname.Text==null)
        goto Skip1;
      else
      //add hostnames to list
      {
        StringReader reader = new StringReader(txtHostname.Text);
        do
        {
          text=reader.ReadLine();
          if (text==null) break;
          v.Add(text);
        }
        while (text != null);
        reader.Close();
      }
      //section to handle all IP validation, generation, & addition
to list
      Skip1:
      {
        //test start IP octets for non-int & out of range
        bool ok = CheckIP(a,b,c,d);
        if (ok==false)
        {
          lstOutputStatus.Items.Add("Host/Start/Network IP: check for
out of range issue.");
          goto End;
        }
        //add single host IP
        if (rbtnHost.Checked)
        {

text=txtStartOctet1.Text+dot+txtStartOctet2.Text+dot+txtStartOctet3.T
ext+dot+txtStartOctet4.Text;
          v.Add(text);
          goto End;
        }
        //for IP ranges, validate end IP, create range, add to list
        if (rbtnRange.Checked)
        {
          //test end IP octets for non-int & out of range
          try
          {
            ok =
CheckIP(Convert.ToInt32(txtEndOctet1.Text),Convert.ToInt32(txtEndOcte
t2.Text),Convert.ToInt32(txtEndOctet3.Text),Convert.ToInt32(txtEndOct
et4.Text));
            if (ok==false)
            {
              lstOutputStatus.Items.Add("End IP: check for out of
range issue.");
              goto End;
            }
          }
          catch
          {
            lstOutputStatus.Items.Add("End IP: check for non-integer
condition.");
            goto End;
```

```
            }
            //build IPs in range & add to list
            v=IPGenerator(a,b,c,d,

txtEndOctet1.Text,txtEndOctet2.Text,txtEndOctet3.Text,txtEndOctet4.Te
xt,v);
            goto End;
          }
          //for networks, validate bitmask, compute end IP, create
range, add to list
          if (rbtnNetwork.Checked)
          {
            //test bitmask for non-int & out of range
            try
            {
              if
((Convert.ToInt32(txtBitmask.Text)<0)^(Convert.ToInt32(txtBitmask.Tex
t)>32))
              {
                lstOutputStatus.Items.Add("Bimask: check for non-
integer or out of range issue.");
                goto End;
              }
            }
            catch
            {
              lstOutputStatus.Items.Add("Bitmask: check for non-integer
or out of range issue.");
              goto End;
            }
            //validate start IP is valid network ID
            int bm=Convert.ToInt32(txtBitmask.Text);
            double Z=System.Math.Pow(2,32-bm);
            if (bm>=24)
            {
              if (d%Z!=0)
              {
                lstOutputStatus.Items.Add("Invalid Network ID for given
Bimtask.");
                goto End;
              }
              else
              {
                txtEndOctet1.Text=txtStartOctet1.Text;
                txtEndOctet2.Text=txtStartOctet2.Text;
                txtEndOctet3.Text=txtStartOctet3.Text;
                txtEndOctet4.Text=Convert.ToString(d+(Z-2));
                v =
IPGenerator(a,b,c,d+1,a.ToString(),b.ToString(),c.ToString(),Convert.
ToString(d+(Z-2)),v);
                goto End;
              }
            }
            if (bm>=16)
            {
              if (c%Z/256!=0)
              {
                lstOutputStatus.Items.Add("Invalid Network ID for given
Bimtask.");
                goto End;
              }
```

```
            else
            {
              d=0;
              txtStartOctet4.Text="0";
              txtEndOctet1.Text=txtStartOctet1.Text;
              txtEndOctet2.Text=txtStartOctet2.Text;
              txtEndOctet3.Text=Convert.ToString(c+((Z/256)-1));
              txtEndOctet4.Text="254";
              v =
IPGenerator(a,b,c,d+1,a.ToString(),b.ToString(),Convert.ToString(c+((
Z/256)-1)),"254",v);
              goto End;
            }
          }
          if (bm>=8)
          {
            if (b%Z/65536!=0)
            {
              lstOutputStatus.Items.Add("Invalid Network ID for given
Bimtask.");
              goto End;
            }
            else
            {
              c=0;d=0;
              txtStartOctet3.Text="0";
              txtStartOctet4.Text="0";
              txtEndOctet1.Text=txtStartOctet1.Text;
              txtEndOctet2.Text=Convert.ToString(b+((Z/65536)-1));
              txtEndOctet3.Text="255";
              txtEndOctet4.Text="254";

v=IPGenerator(a,b,c,d+1,a.ToString(),Convert.ToString(b+((Z/65536)-
1)),"255","254",v);
              goto End;
            }
          }
          if (bm>=0)
          {
            if (a%Z/16777216!=0)
            {
              lstOutputStatus.Items.Add("Invalid Network ID for given
Bimtask.");
              goto End;
            }
            else
            {
              b=0;c=0;d=0;
              txtStartOctet2.Text="0";
              txtStartOctet3.Text="0";
              txtStartOctet4.Text="0";
              txtEndOctet1.Text= Convert.ToString(a+((Z/16777216)-
1));
              txtEndOctet2.Text="255";
              txtEndOctet3.Text="255";
              txtEndOctet4.Text="254";
              v= IPGenerator
(a,b,c,d+1,Convert.ToString(a+((Z/16777216)-1)),"255","255","254",v);
              goto End;
            }
          }
```

```
        }
      }
      End:
      return v;
    }
```

D.6 IP Address Generator

```
    public ArrayList IPGenerator (int a,int b,int c,int d,string
e1,string e2,string e3,string e4,ArrayList w)
    {
      //generates all IPs in a defined range or network
      string txt=null;
      string dot=".";
      //build IPs in range & add to list
      do
      {
        if (d>255)
        {
            c++;d=0;
          if (c>255)
          {
              b++;c=0;
            if (b>255)
            {a++;b=0;}
          }
        }

txt=a.ToString()+dot+b.ToString()+dot+c.ToString()+dot+d.ToString();
        w.Add(txt);
        d++;
      }
      while (txt != e1+dot+e2+dot+e3+dot+e4);
      return w;
    }
D.7
    public bool CheckIP (int a,int b,int c,int d)
    {
      //checks IP octets for numerical range validity
      bool oklocal = true;
      if ((a<0)^(a>255))
        oklocal=false;
      if ((b<0)^(b>255))
        oklocal=false;
      if ((c<0)^(c>255))
        oklocal=false;
      if ((d<0)^(d>255))
        oklocal=false;
      return oklocal;
    }
```

D.7 Main Class for SNMP Communication

```
    public void GetData (string IPAdd, string CommString)
    {
      //main function for SNMP communication with list items
      lstOutputStatus.Items.Add("Attempting to connect to " + IPAdd +
".");
      lstOutputStatus.Refresh();
      try
      {
        //array to store device name & # of if's.
```

```
        ArrayList DeviceAL = new ArrayList(2);
        int rowcount = 0;
        //set oid of cisco description
        uint[] ciscotype = new uint[] {1,3,6,1,4,1,9,2,1,3,0};
        //create snmp session with agent
        ManagerSession sess = new ManagerSession(IPAdd,CommString);
        AsyncMethodCaller caller = new AsyncMethodCaller(CallHost);
        bool contact = false;
        IAsyncResult myresult = caller.BeginInvoke(IPAdd,
CommString,null,null);
        double starttime = System.Environment.TickCount;
        do
        {
          if (myresult.IsCompleted == false)
            Thread.Sleep(10);
          else
          {
            Universal[] Oids = caller.EndInvoke(myresult);
            for (int i = 0; i < Oids.Length; i++)
            {
              Universal result = Oids[i];
              DeviceAL.Add(result[1].ToString());
            }
            lstOutputStatus.Items.Add("Successful response from " +
IPAdd + ".");
            lstOutputStatus.Items.Add("SNMP discovery runtime was " +
runtime + " ms.");
            contact = true;
            break;
          }
        }
        while ((System.Environment.TickCount - starttime) < 10000);
        if (contact == false)
        {
          lstOutputStatus.Items.Add("No response from " + IPAdd +
".");
          goto End;
        }
        if (DeviceAL[2].ToString().ToLower().Contains("windows"))
          DeviceAL[2]="Windows host";
        else
        {
          ManagerItem mi3 = new ManagerItem(sess,ciscotype);
          DeviceAL[2]=mi3.Value.ToString();
        }
        lstOutputStatus.Items.Add("\t\tHost
"+DeviceAL[0].ToString()+" (a "+DeviceAL[2].ToString()+"), has "+
DeviceAL[1].ToString()+" interface(s).");
        //retrieve IP to Index mapping
        string[,] IndxToIp = GetIPtoIndex(sess);
        //array to store if. characteristics
        string[,] InterfaceAR = new
string[Convert.ToInt16(DeviceAL[1].ToString()),8];
        //initialize variable names to store info on each interface
        uint[] ifIndex = new uint[] {1,3,6,1,2,1,2,2,1,1};
        uint[] ifDescr;
        uint[] ifType;
        uint[] ifSpeed;
        //uint[] ifPhysAddress;
        uint[] ifAdminStatus;
        uint[] ifOperStatus;
```

```
        //get index #'s of interfaces
        ManagerSubTree mst = new ManagerSubTree(sess,ifIndex);
        if (mst.Length==0)
          lstOutputStatus.Items.Add("\t\tHost "+IPAdd+" has no
interfaces.");
        else
          foreach(ManagerItem mi in mst)
          {
            //get info for each interface
            InterfaceAR[rowcount,0]=mi.Value.ToString();
            ifDescr = new uint[]
{1,3,6,1,2,1,2,2,1,2,Convert.ToUInt32(mi.Value.ToString())};
            ifType = new uint[]
{1,3,6,1,2,1,2,2,1,3,Convert.ToUInt32(mi.Value.ToString())};
            ifSpeed = new uint[]
{1,3,6,1,2,1,2,2,1,5,Convert.ToUInt32(mi.Value.ToString())};
            ifAdminStatus = new uint[]
{1,3,6,1,2,1,2,2,1,7,Convert.ToUInt32(mi.Value.ToString())};
            ifOperStatus = new uint[]
{1,3,6,1,2,1,2,2,1,8,Convert.ToUInt32(mi.Value.ToString())};
            Universal[] Oids = sess.Get(sess.VarBind(ifDescr),
sess.VarBind(ifType), sess.VarBind(ifSpeed),
              sess.VarBind(ifAdminStatus),
sess.VarBind(ifOperStatus));
            //write out interface values
            for (int i=0; i<Oids.Length; i++)
            {
              Universal result = Oids[i];
              InterfaceAR[rowcount,i+1]=result[1].ToString();
            }
            rowcount++;
          }
        for (int b = 0; b < (IndxToIp.Length / 3); b++)
        {
          for (int c=0; c<Convert.ToInt16(DeviceAL[1]); c++)
            if (IndxToIp[b, 0] == InterfaceAR[c, 0].ToString())
            {
              InterfaceAR[c,6]=IndxToIp[b,1];
              InterfaceAR[c,7]=IndxToIp[b,2];
              break;
            }
        }
        for (int n = 0; n < Convert.ToInt16(DeviceAL[1].ToString());
n++)
        {
          int m=0;
          bool found=false;
          while (found==false)
            if (InterfaceAR[n, 2] == ifTypesAR[m, 0])
            {
              InterfaceAR[n, 2] = ifTypesAR[m, 1];
              found = true;
            }
            else
              m++;
        }
        //call function to write data to XML file
        CreateDevice(DeviceAL[0].ToString().Replace("\"", ""),
DeviceAL[1].ToString(), DeviceAL[2].ToString(), InterfaceAR,
rowcount);
        sess.Close();
```

```
      }
      catch
      {
        lstOutputStatus.Items.Add("Error while processing host:
"+IPAdd+".");
      }
    End: ;
    }
```

D.8 Asynchronous SNMP Call

```
    public delegate Universal[] AsyncMethodCaller(string IPAdd,
string CommString);

    private Universal[] CallHost(string IPAdd, string CommString)
    {
      //initial 'SNMP GET' function called asynchronously
      //set oid of system name
      uint[] sysName = new uint[] { 1, 3, 6, 1, 2, 1, 1, 5, 0 };
      //set oid of # of interfaces
      uint[] ifNumber = new uint[] { 1, 3, 6, 1, 2, 1, 2, 1, 0 };
      //set oid of system description
      uint[] sysDescr = new uint[] { 1, 3, 6, 1, 2, 1, 1, 1, 0 };
      ManagerSession sess = new ManagerSession(IPAdd, CommString);
      double starttime = new double();
      double stoptime = new double();
      starttime = System.Environment.TickCount;
      //get name & # of interfaces to store in array
      Universal[] oids = sess.Get(sess.VarBind(sysName),
sess.VarBind(ifNumber), sess.VarBind(sysDescr));
      stoptime = System.Environment.TickCount;
      sess.Close();
      runtime = stoptime - starttime;
      return oids;
    }
```

D.9 Class for Correlating Interfaces' IP and Index Values

```
    public string[,] GetIPtoIndex (ManagerSession s)
    {
      //function correlates interface indexes and IP addresses
      uint[] ipAdEntAddr = new uint[] {1,3,6,1,2,1,4,20,1,1};
      ManagerSubTree mst = new ManagerSubTree(s,ipAdEntAddr);
      int ni=mst.Length;
      string[,] iptondxAR = new string[ni,3];
      int rowcount = 0;
      uint[] ipAdEntIfIndex;
      uint[] ipAdEntNetMask;
      if (ni==0)
        lstOutputStatus.Items.Add("\t\tHost has no interfaces!");
      else
        foreach(ManagerItem mi in mst)
        {
          iptondxAR[rowcount,1]=mi.Value.ToString();
          //convert IP to csv integers here.
          char[] ipcommas = mi.Value.ToString().Replace('.',
',').ToCharArray();
          int j = 0;
          int k = 0;
          int[,] octetAR = new int[4,4];
          octetAR[0,3]=0;
          foreach (char c in ipcommas)
```

```
            {
              if (c==',')
              {
                j++;
                k=0;
                octetAR[j,3]=0;
              }
              else
              {
                octetAR[j,k]=Convert.ToInt32(c.ToString());
                k++;
                octetAR[j,3]++;
              }
            }
            uint[] ipoctets = new uint[4];
            for (int y=0;y<4;y++)
            {
              switch (octetAR[y,3])
              {
                case 3:

ipoctets[y]=Convert.ToUInt32((octetAR[y,0]*100+octetAR[y,1]*10+octetA
R[y,2]));
                  break;
                case 2:

ipoctets[y]=Convert.ToUInt32((octetAR[y,0]*10+octetAR[y,1]));
                  break;
                case 1:
                  ipoctets[y]=Convert.ToUInt32(octetAR[y,0]);
                  break;
              }
            }
            ipAdEntIfIndex = new uint[]
{1,3,6,1,2,1,4,20,1,2,ipoctets[0],ipoctets[1],ipoctets[2],ipoctets[3]
};
            ipAdEntNetMask = new uint[]
{1,3,6,1,2,1,4,20,1,3,ipoctets[0],ipoctets[1],ipoctets[2],ipoctets[3]
};
            ManagerItem mi2 = new ManagerItem(s,ipAdEntIfIndex);
            ManagerItem mi3 = new ManagerItem(s,ipAdEntNetMask);
            iptondxAR[rowcount,0]=mi2.Value.ToString();
            iptondxAR[rowcount,2]=mi3.Value.ToString();
            rowcount++;
            j=0;
            k=0;
            ipcommas.Initialize();
            octetAR.Initialize();
            ipoctets.Initialize();
          }
        //sort the iptondxAR before returning
        int[] tempAR = new int[ni];
        for (int w=0;w<ni;w++)
          tempAR[w]=Convert.ToInt32(iptondxAR[w,0]);
        Array.Sort(tempAR);
        string[,] FinalAR = new string[ni,3];
        int v=0;
        for (int z=0;z<ni;z++)
        {
          do
          {
```

```
          if (tempAR[z].ToString()==iptondxAR[v,0])
          {
            FinalAR[z,0]=iptondxAR[v,0];
            FinalAR[z,1]=iptondxAR[v,1];
            FinalAR[z,2]=iptondxAR[v,2];
            v=0;
          }
          else
            v++;
        }
        while (FinalAR[z,0]==null);
      }
      return FinalAR;
    }
```

D.10 Main Class for Writing Device/Interfaces to XML File

```
    public void CreateDevice(string s1, string s2, string s3,
string[,] oa, int rc)
    {
      //main function to write data into XML file
      XmlDocument toposource = new XmlDocument();
      toposource.Load(xmlFile);
      double loadnodeend = new double();
      double loadnodebeg = System.Environment.TickCount;
      XmlNodeList nodelist = toposource.SelectNodes("//device");
      loadnodeend = System.Environment.TickCount;
      lstOutputStatus.Items.Add("\t\tTime to load device nodes was "
+ (loadnodeend - loadnodebeg) + " ms.");
      switch (nodelist.Count)
      {
        case 0:
          goto MakeDevNode;
        default:
          goto TestDevNode;
      }
    TestDevNode:
      {
        double findnodeend = new double();
        double findnodebeg = System.Environment.TickCount;
        for (int i = 0; i < nodelist.Count; i++)
          if
(nodelist[i].ChildNodes[0].InnerText.ToLower().Equals(s1.ToLower()))
          {
            findnodeend = System.Environment.TickCount;
            lstOutputStatus.Items.Add("\t\tXML device node " + s1 + "
already exists!");
            lstOutputStatus.Items.Add("\t\tTime to find existing node
was " + (findnodeend-findnodebeg) + " ms.");
            goto End;
          }
      }
      MakeDevNode:
      {
        double newdevnodebeg = System.Environment.TickCount;
        lstOutputStatus.Items.Add("\t\tCreating XML device node:
"+s1+" ...");
        XmlElement newDevice = toposource.CreateElement("device");
        XmlAttribute newAttr = toposource.CreateAttribute("type");
        newAttr.Value = s3.Replace("\"", "");
        newDevice.Attributes.Append(newAttr);
```

```
        newDevice.InnerXml =
"<sysName></sysName><ifNumber></ifNumber>";
        newDevice["sysName"].InnerText=s1.Replace("\"","");
        newDevice["ifNumber"].InnerText = s2.Replace("\"", "");
        for (int i=0;i<rc;i++)
        {
          lstOutputStatus.Items.Add("\t\tCreating XML interface node:
"+(oa[i,1].Replace("\"", "")).Replace("\0", "")+".");
          double newinfnodebeg = System.Environment.TickCount;
          XmlElement newInterface =
toposource.CreateElement("interface");
          newAttr = toposource.CreateAttribute("ifIndex");
          newAttr.Value = oa[i,0];
          newInterface.Attributes.Append(newAttr);
          newInterface.InnerXml =
"<ifDescr></ifDescr><ifType></ifType><ifSpeed></ifSpeed><ifAdminStatu
s></ifAdminStatus><ifOperStatus></ifOperStatus><IPAddress></IPAddress
><NetMask></NetMask>";
          newInterface["ifDescr"].InnerText = (oa[i, 1].Replace("\"",
"")).Replace("\0", "");
          newInterface["ifType"].InnerText=oa[i,2];
          newInterface["ifSpeed"].InnerText=oa[i,3];
          newInterface["ifAdminStatus"].InnerText=oa[i,4];
          newInterface["ifOperStatus"].InnerText=oa[i,5];
          newInterface["IPAddress"].InnerText=oa[i,6];
          newInterface["NetMask"].InnerText=oa[i,7];
          newDevice.AppendChild(newInterface);
          double newinfnodeend = System.Environment.TickCount;
          lstOutputStatus.Items.Add("\t\t XML interface " + (oa[i,
1].Replace("\"", "")).Replace("\0", "") + " created in " +
(newinfnodeend - newinfnodebeg) + " ms.");
        }
        toposource.DocumentElement.AppendChild(newDevice);
        toposource.Save(xmlFile);
        double newdevnodeend = System.Environment.TickCount;
        lstOutputStatus.Items.Add("\t\t XML device node " + s1 + "
creation time was "+(newdevnodeend-newdevnodebeg)+" ms.");
      }
      End:{}

    }
```

D.11 User Interface Controls

```
    private void Form1_Load(object sender, System.EventArgs e)
    {
      DocTest();
      ifTypesAR=LoadifTypes();
      txtBitmask.Text="32";
    }
    private void llblCreateList_LinkClicked(object sender,
System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
    {
      double createlistbeg = System.Environment.TickCount;
      IPAR = CreateHostList();
      double createlistend = System.Environment.TickCount;
      if (IPAR.Count==0)
        lstOutputStatus.Items.Add("No hostnames or valid IP Addresses
provided.");
      else
      {
```

```
        lstOutputStatus.Items.Add("List of hostnames & IPs
created.");
        lstOutputStatus.Items.Add("Time to create list of
"+IPAR.Count.ToString()+" hosts was "+(createlistend-createlistbeg)+"
ms.");
        llblViewList.Visible=true;
        llblGetData.Visible=true;
        lblListCount.Text="List contains "+IPAR.Count.ToString()+"
item(s).";
        llblCreateList.Text="Update List";
      }
      lstOutputStatus.TopIndex = (lstOutputStatus.Items.Count -
(lstOutputStatus.Height / lstOutputStatus.ItemHeight));
    }
    private void llblViewList_LinkClicked(object sender,
System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
    {
      if (File.Exists("list.txt"))
        File.Delete("list.txt");
      using (StreamWriter writer = new StreamWriter("list.txt"))
      {
        for (int i=0;i<IPAR.Count;i++)
          writer.WriteLine(IPAR[i]);
      }
      System.Diagnostics.Process.Start("list.txt");
    }
    private void llblGetData_MouseEnter(object sender,
System.EventArgs e)
    {
      lblListCount.Visible=true;
    }
    private void llblGetData_MouseLeave(object sender,
System.EventArgs e)
    {
      lblListCount.Visible=false;
    }
    private void llblGetData_LinkClicked(object sender,
System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
    {
      if (txtPassword.Text.Length==0)
      {
        lstOutputStatus.Items.Add("Community String cannot be
blank.");
        goto End;
      }
      double fulltimebeg = System.Environment.TickCount;
      foreach (string j in IPAR)
      {
        double hosttimebeg = System.Environment.TickCount;
        GetData(j, txtPassword.Text);
        double hosttimeend = System.Environment.TickCount;
        lstOutputStatus.Items.Add("Time to process host "+j+" was " +
(hosttimeend - hosttimebeg) + " ms.");
      }
      double fulltimeend = System.Environment.TickCount;
      lstOutputStatus.Items.Add("Total runtime was "+ (fulltimeend-
fulltimebeg)+" ms.");
      lstOutputStatus.Items.Add("DONE.");
      lstOutputStatus.Items.Add("*******************************");
      lstOutputStatus.Items.Add("");
```

```
      lstOutputStatus.TopIndex = (lstOutputStatus.Items.Count -
(lstOutputStatus.Height / lstOutputStatus.ItemHeight));
      End:;
    }
    private void llblViewXML_LinkClicked(object sender,
System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
    {
      System.Diagnostics.Process.Start("notepad.exe", xmlFile);
    }
    private void llblGenDoc_LinkClicked(object sender,
System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
    {
      System.Diagnostics.Process.Start(xmlFile);
    }
    private void txtPassword_Enter(object sender, System.EventArgs e)
    {
      lblPassword.Visible=true;
      lblArrow.Visible = true;
    }
    private void txtPassword_Leave(object sender, System.EventArgs e)
    {
      lblPassword.Visible=false;
      lblArrow.Visible = false;
    }
    private void txtHostname_Enter(object sender, EventArgs e)
    {
      lblHostnameDir.Visible = true;
    }
    private void txtHostname_Leave(object sender, EventArgs e)
    {
      lblHostnameDir.Visible = false;
    }
    private void rbtnHost_Click(object sender, System.EventArgs e)
    {
      rbtnHost.Checked=true;
      rbtnRange.Checked=false;
      rbtnNetwork.Checked=false;
      lblStartIP.Text="Host IP:";
      txtEndOctet1.Enabled=false;
      txtEndOctet2.Enabled=false;
      txtEndOctet3.Enabled=false;
      txtEndOctet4.Enabled=false;
      txtBitmask.Enabled=false;
      txtBitmask.Text="32";
      lblTip.Visible=false;
    }
    private void rbtnRange_Click(object sender, System.EventArgs e)
    {
      rbtnHost.Checked=false;
      rbtnRange.Checked=true;
      rbtnNetwork.Checked=false;
      lblStartIP.Text="Start IP:";
      txtEndOctet1.Enabled=true;
      txtEndOctet2.Enabled=true;
      txtEndOctet3.Enabled=true;
      txtEndOctet4.Enabled=true;
      txtBitmask.Enabled=false;
      txtBitmask.Text="24";
      lblNote.Text=null;
      lblTip.Visible=false;
    }
```

```
    private void rbtnNetwork_Click(object sender, System.EventArgs e)
    {
      rbtnHost.Checked=false;
      rbtnRange.Checked=false;
      rbtnNetwork.Checked=true;
      lblStartIP.Text="Network ID:";
      txtEndOctet1.Enabled=false;
      txtEndOctet2.Enabled=false;
      txtEndOctet3.Enabled=false;
      txtEndOctet4.Enabled=false;
      txtBitmask.Enabled=true;
      txtBitmask.Text=null;
      txtBitmask.Text="24";
      lblTip.Visible=true;
    }
    private void txtStartOctet1_TextChanged(object sender,
System.EventArgs e)
    {
      try
      {
        if
((Convert.ToInt32(txtStartOctet1.Text)<0)^(Convert.ToInt32(txtStartOc
tet1.Text))>255)
          txtStartOctet1.BackColor=Color.LightPink;
        else
          txtStartOctet1.BackColor=Color.White;
      }
      catch{txtStartOctet1.BackColor=Color.LightPink;}
    }
    private void txtStartOctet2_TextChanged(object sender,
System.EventArgs e)
    {
      try
      {
        if
((Convert.ToInt32(txtStartOctet2.Text)<0)^(Convert.ToInt32(txtStartOc
tet2.Text))>255)
          txtStartOctet2.BackColor=Color.LightPink;
        else
          txtStartOctet2.BackColor=Color.White;
      }
      catch{txtStartOctet2.BackColor=Color.LightPink;}
    }
    private void txtStartOctet3_TextChanged(object sender,
System.EventArgs e)
    {
      try
      {
        if
((Convert.ToInt32(txtStartOctet3.Text)<0)^(Convert.ToInt32(txtStartOc
tet3.Text))>255)
          txtStartOctet3.BackColor=Color.LightPink;
        else
          txtStartOctet3.BackColor=Color.White;
      }
      catch{txtStartOctet3.BackColor=Color.LightPink;}
    }
    private void txtStartOctet4_TextChanged(object sender,
System.EventArgs e)
    {
      try
```

```
      {
        if
((Convert.ToInt32(txtStartOctet4.Text)<0)^(Convert.ToInt32(txtStartOc
tet4.Text))>255)
          txtStartOctet4.BackColor=Color.LightPink;
        else
          txtStartOctet4.BackColor=Color.White;
      }
      catch{txtStartOctet4.BackColor=Color.LightPink;}
    }
    private void txtEndOctet1_TextChanged(object sender,
System.EventArgs e)
    {
      try
      {
        if
((Convert.ToInt32(txtEndOctet1.Text)<0)^(Convert.ToInt32(txtEndOctet1
.Text))>255)
          txtEndOctet1.BackColor=Color.LightPink;
        else
          txtEndOctet1.BackColor=Color.White;
      }
      catch{txtEndOctet1.BackColor=Color.LightPink;}
    }
    private void txtEndOctet2_TextChanged(object sender,
System.EventArgs e)
    {
      try
      {
        if
((Convert.ToInt32(txtEndOctet2.Text)<0)^(Convert.ToInt32(txtEndOctet2
.Text))>255)
          txtEndOctet2.BackColor=Color.LightPink;
        else
          txtEndOctet2.BackColor=Color.White;
      }
      catch{txtEndOctet2.BackColor=Color.LightPink;}
    }
    private void txtEndOctet3_TextChanged(object sender,
System.EventArgs e)
    {
      try
      {
        if
((Convert.ToInt32(txtEndOctet3.Text)<0)^(Convert.ToInt32(txtEndOctet3
.Text))>255)
          txtEndOctet3.BackColor=Color.LightPink;
        else
          txtEndOctet3.BackColor=Color.White;
      }
      catch{txtEndOctet3.BackColor=Color.LightPink;}
    }
    private void txtEndOctet4_TextChanged(object sender,
System.EventArgs e)
    {
      try
      {
        if
((Convert.ToInt32(txtEndOctet4.Text)<0)^(Convert.ToInt32(txtEndOctet4
.Text))>255)
          txtEndOctet4.BackColor=Color.LightPink;
```

```
        else
          txtEndOctet4.BackColor=Color.White;
      }
      catch{txtEndOctet4.BackColor=Color.LightPink;}
    }
    private void txtBitmask_TextChanged(object sender,
System.EventArgs e)
    {
      try
      {
        switch (txtBitmask.Text.Length.Equals(0))
        {
          case true:
            txtMask1.Enabled=true;
            txtMask2.Enabled=true;
            txtMask3.Enabled=true;
            txtMask4.Enabled=true;
            lblNote.Text="";
            break;
          case false:
            txtMask1.Enabled=false;
            txtMask2.Enabled=false;
            txtMask3.Enabled=false;
            txtMask4.Enabled=false;
            if
(Convert.ToInt32(txtBitmask.Text)<0^Convert.ToInt32(txtBitmask.Text)>
32)
            {
              lblNote.Text="Invalid mask; value must be in range: 0 -
32";
              goto End;
            }
            if
(txtBitmask.Text.Equals("32")^txtBitmask.Text.Equals("31"))
            {
              txtEndOctet1.Enabled=false;
              txtEndOctet2.Enabled=false;
              txtEndOctet3.Enabled=false;
              txtEndOctet4.Enabled=false;
              txtBitmask.Text="32";
              lblNote.Text="Tool will try to contact 1 host.";
              goto End;
            }
            if (Convert.ToInt32(txtBitmask.Text)>=0)
            {
              lblNote.Text="Tool will try to contact
"+(System.Math.Pow(2,32-Convert.ToDouble(txtBitmask.Text))-2)+"
hosts.";
              goto End;
            }
            break;
        }
      }
      catch
      {lblNote.Text="Bitmask must be integer.";}
      End:;
    }
    private void txtMask1_TextChanged(object sender, System.EventArgs
e)
    {
      try
```

```
      {Convert.ToInt32(txtMask1.Text);}
      catch
      {
        lblNote.Text="Octet 1 invalid. Allowed: 128, 192, 224, 240,
248, 252, 254, 255.";
        goto End;
      }
      if (!(txtMask1.Text.Length==0)&(txtMask1.Enabled==true))
      {
        switch (Convert.ToInt32(txtMask1.Text))
        {
          case 255:
            txtBitmask.Text="8";
            goto Set;
          case 254:
            txtBitmask.Text="7";
            goto Set;
          case 252:
            txtBitmask.Text="6";
            goto Set;
          case 248:
            txtBitmask.Text="5";
            goto Set;
          case 240:
            txtBitmask.Text="4";
            goto Set;
          case 224:
            txtBitmask.Text="3";
            goto Set;
          case 192:
            txtBitmask.Text="2";
            goto Set;
          case 128:
            txtBitmask.Text="1";
            goto Set;
          default:
            lblNote.Text="Octet 1 invalid. Allowed: 128, 192, 224,
240, 248, 252, 254, 255.";
            goto End;
        }
      }
      else
        goto End;
      Set:
        txtMask2.Text="0";
      txtMask3.Text="0";
      txtMask4.Text="0";
      End:;
    }
    private void txtMask2_TextChanged(object sender, System.EventArgs
e)
    {
      try
      {Convert.ToInt32(txtMask2.Text);}
      catch
      {
        lblNote.Text="Octet 2 invalid. Allowed: 128, 192, 224, 240,
248, 252, 254, 255.";
        goto End;
      }
      if (!(txtMask2.Text.Length==0)&(txtMask2.Enabled==true))
```

```
      {
        switch (Convert.ToInt32(txtMask2.Text))
        {
          case 255:
            txtBitmask.Text="16";
            goto Set;
          case 254:
            txtBitmask.Text="15";
            goto Set;
          case 252:
            txtBitmask.Text="14";
            goto Set;
          case 248:
            txtBitmask.Text="13";
            goto Set;
          case 240:
            txtBitmask.Text="12";
            goto Set;
          case 224:
            txtBitmask.Text="11";
            goto Set;
          case 192:
            txtBitmask.Text="10";
            goto Set;
          case 128:
            txtBitmask.Text="9";
            goto Set;
          default:
            lblNote.Text="Octet 2 invalid. Allowed: 128, 192, 224,
240, 248, 252, 254, 255.";
            goto End;
        }
      }
      else
        goto End;
      Set:
        txtMask1.Text="255";
      txtMask3.Text="0";
      txtMask4.Text="0";
      End:;
    }
    private void txtMask3_TextChanged(object sender, System.EventArgs
e)
    {
      if (!(txtMask3.Text.Length==0)&(txtMask1.Enabled==true))
      {
        try
        {Convert.ToInt32(txtMask3.Text);}
        catch
        {
          lblNote.Text="Octet 3 invalid. Allowed: 128, 192, 224, 240,
248, 252, 254, 255.";
          goto End;
        }
        switch (Convert.ToInt32(txtMask3.Text))
        {
          case 255:
            txtBitmask.Text="24";
            goto Set;
          case 254:
            txtBitmask.Text="23";
```

```
            goto Set;
          case 252:
            txtBitmask.Text="22";
            goto Set;
          case 248:
            txtBitmask.Text="21";
            goto Set;
          case 240:
            txtBitmask.Text="20";
            goto Set;
          case 224:
            txtBitmask.Text="19";
            goto Set;
          case 192:
            txtBitmask.Text="18";
            goto Set;
          case 128:
            txtBitmask.Text="17";
            goto Set;
          default:
            lblNote.Text="Octet 3 invalid. Allowed: 128, 192, 224,
240, 248, 252, 254, 255.";
            goto End;
        }
      }
      else
        goto End;
      Set:
        txtMask1.Text="255";
      txtMask2.Text="255";
      txtMask4.Text="0";
      End:;
    }
    private void txtMask4_TextChanged(object sender, System.EventArgs
e)
    {
      if (!(txtMask4.Text.Length==0)&(txtMask1.Enabled==true))
      {
        try
        {Convert.ToInt32(txtMask4.Text);}
        catch
        {
          lblNote.Text="Octet 4 invalid. Allowed: 128, 192, 224, 240,
248, 252, 254, 255.";
          goto End;
        }
        switch (Convert.ToInt32(txtMask4.Text))
        {
          case 255:
            txtBitmask.Text="32";
            goto Set;
          case 254:
            txtBitmask.Text="31";
            goto Set;
          case 252:
            txtBitmask.Text="30";
            goto Set;
          case 248:
            txtBitmask.Text="29";
            goto Set;
          case 240:
```

```
            txtBitmask.Text="28";
            goto Set;
          case 224:
            txtBitmask.Text="27";
            goto Set;
          case 192:
            txtBitmask.Text="26";
            goto Set;
          case 128:
            txtBitmask.Text="25";
            goto Set;
          default:
            lblNote.Text="Octet 4 invalid. Allowed: 128, 192, 224,
240, 248, 252, 254, 255.";
            goto End;
        }
      }
      else
        goto End;
      Set:
        txtMask1.Text="255";
        txtMask2.Text="255";
        txtMask3.Text="255";
      End:;
    }
  }
}
```

Appendix E – XML Files

E.1 XML Source File (Example)

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type='text/xsl' href='web-device-report.xslt'?>
<topology>
  <!--Start of device listing-->
  <device type="Windows host">
    <sysName>NEONFISH</sysName>
    <ifNumber>3</ifNumber>
    <interface ifIndex="1">
      <ifDescr>MS TCP Loopback interface</ifDescr>
      <ifType>softwareLoopback</ifType>
      <ifSpeed>10000000</ifSpeed>
      <ifAdminStatus>1</ifAdminStatus>
      <ifOperStatus>1</ifOperStatus>
      <IPAddress>127.0.0.1</IPAddress>
      <NetMask>255.0.0.0</NetMask>
    </interface>
    <interface ifIndex="2">
      <ifDescr>Intel(R) PRO/Wireless 2200BG Network Connection -
Packet Scheduler Miniport</ifDescr>
      <ifType>ethernetCsmacd</ifType>
      <ifSpeed>54000000</ifSpeed>
      <ifAdminStatus>1</ifAdminStatus>
      <ifOperStatus>2</ifOperStatus>
      <IPAddress>0.0.0.0</IPAddress>
      <NetMask>0.0.0.0</NetMask>
    </interface>
    <interface ifIndex="65540">
      <ifDescr>Broadcom 440x 10/100 Integrated Controller - Packet
Scheduler Miniport</ifDescr>
      <ifType>ethernetCsmacd</ifType>
      <ifSpeed>100000000</ifSpeed>
      <ifAdminStatus>1</ifAdminStatus>
      <ifOperStatus>1</ifOperStatus>
      <IPAddress>192.168.0.3</IPAddress>
      <NetMask>255.255.255.0</NetMask>
    </interface>
  </device>
  <device type="Router">
    <sysName>RouterA</sysName>
    <ifNumber>6</ifNumber>
    <interface ifIndex="1">
      <ifDescr>FastEthernet0/0</ifDescr>
      <ifType>ethernetCsmacd</ifType>
      <ifSpeed>100000000</ifSpeed>
      <ifAdminStatus>1</ifAdminStatus>
      <ifOperStatus>1</ifOperStatus>
      <IPAddress>192.168.0.1</IPAddress>
      <NetMask>255.255.255.0</NetMask>
    </interface>
    <interface ifIndex="2">
      <ifDescr>Serial0/0</ifDescr>
      <ifType>propPointToPointSerial</ifType>
      <ifSpeed>128000</ifSpeed>
      <ifAdminStatus>2</ifAdminStatus>
```

```
      <ifOperStatus>2</ifOperStatus>
      <IPAddress>
      </IPAddress>
      <NetMask>
      </NetMask>
    </interface>
    <interface ifIndex="3">
      <ifDescr>Serial0/1</ifDescr>
      <ifType>propPointToPointSerial</ifType>
      <ifSpeed>128000</ifSpeed>
      <ifAdminStatus>2</ifAdminStatus>
      <ifOperStatus>2</ifOperStatus>
      <IPAddress>
      </IPAddress>
      <NetMask>
      </NetMask>
    </interface>
    <interface ifIndex="4">
      <ifDescr>Serial0/2</ifDescr>
      <ifType>propPointToPointSerial</ifType>
      <ifSpeed>128000</ifSpeed>
      <ifAdminStatus>2</ifAdminStatus>
      <ifOperStatus>2</ifOperStatus>
      <IPAddress>
      </IPAddress>
      <NetMask>
      </NetMask>
    </interface>
    <interface ifIndex="5">
      <ifDescr>Serial0/3</ifDescr>
      <ifType>propPointToPointSerial</ifType>
      <ifSpeed>128000</ifSpeed>
      <ifAdminStatus>2</ifAdminStatus>
      <ifOperStatus>2</ifOperStatus>
      <IPAddress>
      </IPAddress>
      <NetMask>
      </NetMask>
    </interface>
    <interface ifIndex="6">
      <ifDescr>Null0</ifDescr>
      <ifType>other</ifType>
      <ifSpeed>4294967295</ifSpeed>
      <ifAdminStatus>1</ifAdminStatus>
      <ifOperStatus>1</ifOperStatus>
      <IPAddress>
      </IPAddress>
      <NetMask>
      </NetMask>
    </interface>
  </device>
</topology>
```

E.2 XSLT Transform File 1 to HTML

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
```

```
  <!--use document element to create html page-->
  <xsl:template match="topology">
    <html>
      <head>
        <link rel="stylesheet" href="topology.css" />
        <title>My Network Nodes</title>
      </head>
      <body>
        <h1>Discovered Network Nodes</h1>
        <br></br>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <!--output device & icon-->
  <xsl:template match="device">
    <h1>
      <table>
        <tr>
          <td>
            <xsl:choose>
              <xsl:when test="@type='Router'">
                <img src="router.jpg"></img>
              </xsl:when>
              <xsl:when test="@type='Switch'">
                <img src="switch.jpg"></img>
              </xsl:when>
              <xsl:when test="@type='Windows host'">
                <img src="windows.jpg"></img>
              </xsl:when>
            </xsl:choose>
          </td>
          <td>
            <h1>
              <xsl:value-of select="sysName"></xsl:value-of>
            </h1>
            Device has <xsl:value-of select="ifNumber"></xsl:value-
of> interfaces.
          </td>
        </tr>
      </table>
    </h1>
    <table border="1">
      <tr>
        <td>Index</td>
        <td>Description</td>
        <td>Interface Type<br>
        <font size="1">
          <a href="http://www.iana.org/assignments/ianaiftype-
mib">Reference</a>
        </font>
        </br>
      </td>
        <td>Speed</td>
        <td>Status</td>
        <td>IP Address<br>NetMask</br></td>
      </tr>
      <xsl:apply-templates select="interface"/>
    </table>
```

```
    <br></br>
    <br></br>
  </xsl:template>
  <xsl:template match="interface">
    <tr>
      <td>
        <xsl:value-of select="@ifIndex"></xsl:value-of>
      </td>
      <td>
        <xsl:value-of select="ifDescr"></xsl:value-of>
      </td>
      <td>
        <xsl:value-of select="ifType"></xsl:value-of>
      </td>
      <xsl:choose>
        <xsl:when test="((ifSpeed)div(1000000000))>=1">
          <td>
            <xsl:value-of
select="(ifSpeed)div(1000000000)"></xsl:value-of>
            <xsl:text> Gbps</xsl:text>
          </td>
        </xsl:when>
        <xsl:when test="((ifSpeed)div(1000000))>=1">
          <td>
            <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-
of>
            <xsl:text> Mbps</xsl:text>
          </td>
        </xsl:when>
        <xsl:when test="((ifSpeed)div(1000))>=1">
          <td>
            <xsl:value-of select="(ifSpeed)div(1000)"></xsl:value-of>
            <xsl:text> Kbps</xsl:text>
          </td>
        </xsl:when>
      </xsl:choose>
      <xsl:choose>
        <xsl:when test="ifOperStatus=2 and ifAdminStatus=2">
          <td>
            <font color="blue">DOWN</font>
          </td>
        </xsl:when>
        <xsl:when test="ifOperStatus=2">
          <td>
            <font color="red">DOWN</font>
          </td>
        </xsl:when>
        <xsl:when test="ifOperStatus=1">
          <td>
            <font color="green">UP</font>
          </td>
        </xsl:when>
      </xsl:choose>
      <td>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <br></br>
        <xsl:value-of select="NetMask"></xsl:value-of>
      </td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

E.3 XSLT Transform File 2 to HTML

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <!--use document element to create html page-->
  <xsl:template match="topology">
    <html>
      <head>
        <link rel="stylesheet" href="topology.css" />
        <title>My Network Devices</title>
      </head>
      <body>
        <h1>Discovered Network Devices</h1>
        <br></br>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <!--output device & icon-->
  <xsl:template match="device">
    <h2>
      <xsl:value-of select="sysName"></xsl:value-of>
    </h2>
    <p></p>
    <xsl:choose>
      <xsl:when test="@type='Router'">
        <img src="r2620.jpg" usemap="#r2650_map" style="border-
style:none"/>
        <map id="r2650_map" name="r2650_map">
          <xsl:apply-templates mode="router" select="interface"/>
        </map>
      </xsl:when>
      <xsl:when test="@type='Switch'">
        <img src="cat2950.jpg" usemap="#cat2950_map" style="border-
style:none"></img>
        <map id="cat2950_map" name="cat2950_map">
          <xsl:apply-templates mode="switch" select="interface"/>
        </map>
      </xsl:when>
      <xsl:when test="@type='Windows host'">
        <img src="laptop.jpg"></img>
      </xsl:when>
    </xsl:choose>
    <br></br>
    <br></br>
    <br></br>
    <hr></hr>
  </xsl:template>
  <xsl:template mode="router" match="interface">
    <xsl:choose>
      <xsl:when test="ifDescr='FastEthernet0/0'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="702,60,744,92" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
```

```
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='Serial0/0'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="755,7,841,23" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000)"></xsl:value-of>
        <xsl:text> Kbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
      </xsl:when>
      <xsl:when test="ifDescr='Serial0/1'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="755,25,841,41" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000)"></xsl:value-of>
        <xsl:text> Kbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
      </xsl:when>
      <xsl:when test="ifDescr='Serial0/2'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="544,7,631,22" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000)"></xsl:value-of>
        <xsl:text> Kbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
      </xsl:when>
      <xsl:when test="ifDescr='Serial0/3'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="544,24,631,39" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000)"></xsl:value-of>
        <xsl:text> Kbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
      </xsl:when>
    </xsl:choose>
```

```
  </xsl:template>
  <xsl:template mode="switch" match="interface">
    <xsl:choose>
      <xsl:when test="ifDescr='FastEthernet0/1'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="126,25,157,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/2'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="160,25,189,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/3'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="192,25,225,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/4'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="227,25,257,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/5'">
```

```
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="258,25,291,83" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/6'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="292,24,324,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/7'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="326,24,357,83" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/8'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="359,24,391,84" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/9'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="404,23,436,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
```

```
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/10'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="437,24,468,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/11'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="471,24,502,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/12'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="505,24,535,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/13'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="538,25,568,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
```

```
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/14'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="571,24,602,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/15'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="604,24,636,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/16'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="637,25,668,83" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/17'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="683,24,714,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/18'">
```

```
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="717,25,748,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/19'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="750,26,781,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/20'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="784,26,815,84" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/21'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="818,26,848,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/22'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="850,27,880,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
```

```
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/23'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="883,27,913,81" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
      <xsl:when test="ifDescr='FastEthernet0/24'">
        <xsl:text disable-output-escaping="yes">&lt;area shape="rect"
href=# coords="917,27,947,82" title="</xsl:text>
        <xsl:value-of select="ifDescr"/>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="(ifSpeed)div(1000000)"></xsl:value-of>
        <xsl:text> Mbps</xsl:text>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="IPAddress"></xsl:value-of>
        <xsl:text> -- </xsl:text>
        <xsl:value-of select="NetMask"></xsl:value-of>
        <xsl:text disable-output-escaping="yes">"/&gt;</xsl:text>
        <xsl:text/>
      </xsl:when>
    </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

E.4 XSLT Transform File 3 to PDF

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  Author:
  File:
  Date:
  Purpose:
 -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <!--<xsl:output method="xml"/>-->
  <xsl:template match="topology">
    <fo:root>
      <fo:layout-master-set>
        <fo:simple-page-master master-name="simple"
           page-height="8.5in" page-width="11in"
```

```
            margin-top="1in" margin-bottom="1in"
            margin-left="1in" margin-right="1in">
          <fo:region-body margin-top="0in"/>
        </fo:simple-page-master>
      </fo:layout-master-set>

      <fo:page-sequence master-reference="simple">
        <fo:flow flow-name="xsl-region-body">
          <fo:block font-size="20pt"
            font-family="sans-serif"
            color="white"
            background-color="blue"
            space-after="12pt"
            border-after-style="solid"
            border-after-width="4pt"
            border-after-color="cyan">
            Discovered Network Nodes
          </fo:block>
          <xsl:apply-templates select="device"/>
        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>
  <xsl:template match="device">
    <xsl:choose>
      <xsl:when test="@type='Router'">
        <fo:block font-size="20pt"
          font-family="sans-serif"
          color="white"
          background-color="blue"
          border-after-style="solid"
          border-after-width="4pt"
          border-after-color="black">
          <fo:external-graphic src="router.jpg"/>
          <xsl:value-of select="sysName"/>
        </fo:block>
      </xsl:when>
      <xsl:when test="@type='Switch'">
        <fo:block font-size="20pt"
          font-family="sans-serif"
          color="white"
          background-color="blue"
          border-after-style="solid"
          border-after-width="4pt"
          border-after-color="black">
          <fo:external-graphic src="switch.jpg"/>
          <xsl:value-of select="sysName"/>
        </fo:block>
      </xsl:when>
      <xsl:when test="@type='Windows host'">
        <fo:block font-size="20pt"
          font-family="sans-serif"
          color="white"
          background-color="blue"
          border-after-style="solid"
          border-after-width="4pt"
          border-after-color="black">
          <fo:external-graphic src="windows.jpg"/>
          <xsl:value-of select="sysName"/>
        </fo:block>
      </xsl:when>
```

```
    </xsl:choose>
    <fo:table space-after="12pt" table-layout="fixed" font-
size="10pt">
      <fo:table-column column-number="1" column-width="proportional-
column-width(.5)"></fo:table-column>
      <fo:table-column column-number="2" column-width="proportional-
column-width(2)"></fo:table-column>
      <fo:table-column column-number="3" column-width="proportional-
column-width(2)"></fo:table-column>
      <fo:table-column column-number="4" column-width="proportional-
column-width(1)"></fo:table-column>
      <fo:table-column column-number="5" column-width="proportional-
column-width(.5)"></fo:table-column>
      <fo:table-column column-number="6" column-width="proportional-
column-width(1)"></fo:table-column>

      <fo:table-body>
        <fo:table-row background-color="gray">
          <fo:table-cell border-collapse="collapse" border="solid
black 1px">
            <fo:block>Index</fo:block>
          </fo:table-cell>
          <fo:table-cell border-collapse="collapse" border="solid
black 1px">
            <fo:block>Description</fo:block>
          </fo:table-cell>
          <fo:table-cell border-collapse="collapse" border="solid
black 1px">
            <fo:block>Interface Type</fo:block>
          </fo:table-cell>
          <fo:table-cell border-collapse="collapse" border="solid
black 1px">
            <fo:block>Speed</fo:block>
          </fo:table-cell>
          <fo:table-cell border-collapse="collapse" border="solid
black 1px">
            <fo:block>Status</fo:block>
          </fo:table-cell>
          <fo:table-cell border-collapse="collapse" border="solid
black 1px">
            <fo:block>IP Address</fo:block>
            <fo:block>NetMask</fo:block>
          </fo:table-cell>
        </fo:table-row>
        <xsl:apply-templates select="interface"/>
      </fo:table-body>
    </fo:table>
  </xsl:template>
  <xsl:template match="interface">
    <fo:table-row>
      <fo:table-cell border-collapse="collapse" border="solid black
1px">
        <fo:block>
          <xsl:value-of select="@ifIndex"/>
        </fo:block>
      </fo:table-cell>
      <fo:table-cell border-collapse="collapse" border="solid black
1px">
        <fo:block>
          <xsl:value-of select="ifDescr"/>
        </fo:block>
```

```
      </fo:table-cell>
      <fo:table-cell border-collapse="collapse" border="solid black
1px">
        <fo:block>
          <xsl:value-of select="ifType"/>
        </fo:block>
      </fo:table-cell>
      <fo:table-cell border-collapse="collapse" border="solid black
1px">
        <fo:block>
          <xsl:choose>
            <xsl:when test="((ifSpeed)div(1000000000))>=1">
              <xsl:value-of
select="(ifSpeed)div(1000000000)"></xsl:value-of> Gbps
            </xsl:when>
            <xsl:when test="((ifSpeed)div(1000000))>=1">
              <xsl:value-of
select="(ifSpeed)div(1000000)"></xsl:value-of> Mbps
            </xsl:when>
            <xsl:when test="((ifSpeed)div(1000))>=1">
              <xsl:value-of select="(ifSpeed)div(1000)"></xsl:value-
of> Kbps
            </xsl:when>
          </xsl:choose>
        </fo:block>
      </fo:table-cell>
      <fo:table-cell border-collapse="collapse" border="solid black
1px" text-align="center">
        <fo:block>
          <xsl:choose>
            <xsl:when test="ifOperStatus=2 and ifAdminStatus=2">
              <fo:inline color="blue">DOWN</fo:inline>
            </xsl:when>
            <xsl:when test="ifOperStatus=2">
              <fo:inline color="red">DOWN</fo:inline>
            </xsl:when>
            <xsl:when test="ifOperStatus=1">
              <fo:inline color="green">UP</fo:inline>
            </xsl:when>
          </xsl:choose>
        </fo:block>
      </fo:table-cell>
      <fo:table-cell border-collapse="collapse" border="solid black
1px">
        <fo:block>
          <xsl:value-of select="IPAddress"/>
        </fo:block>
        <fo:block>
          <xsl:value-of select="NetMask"/>
        </fo:block>
      </fo:table-cell>
    </fo:table-row>
  </xsl:template>
</xsl:stylesheet>
```

E.5 FO File (Truncated)

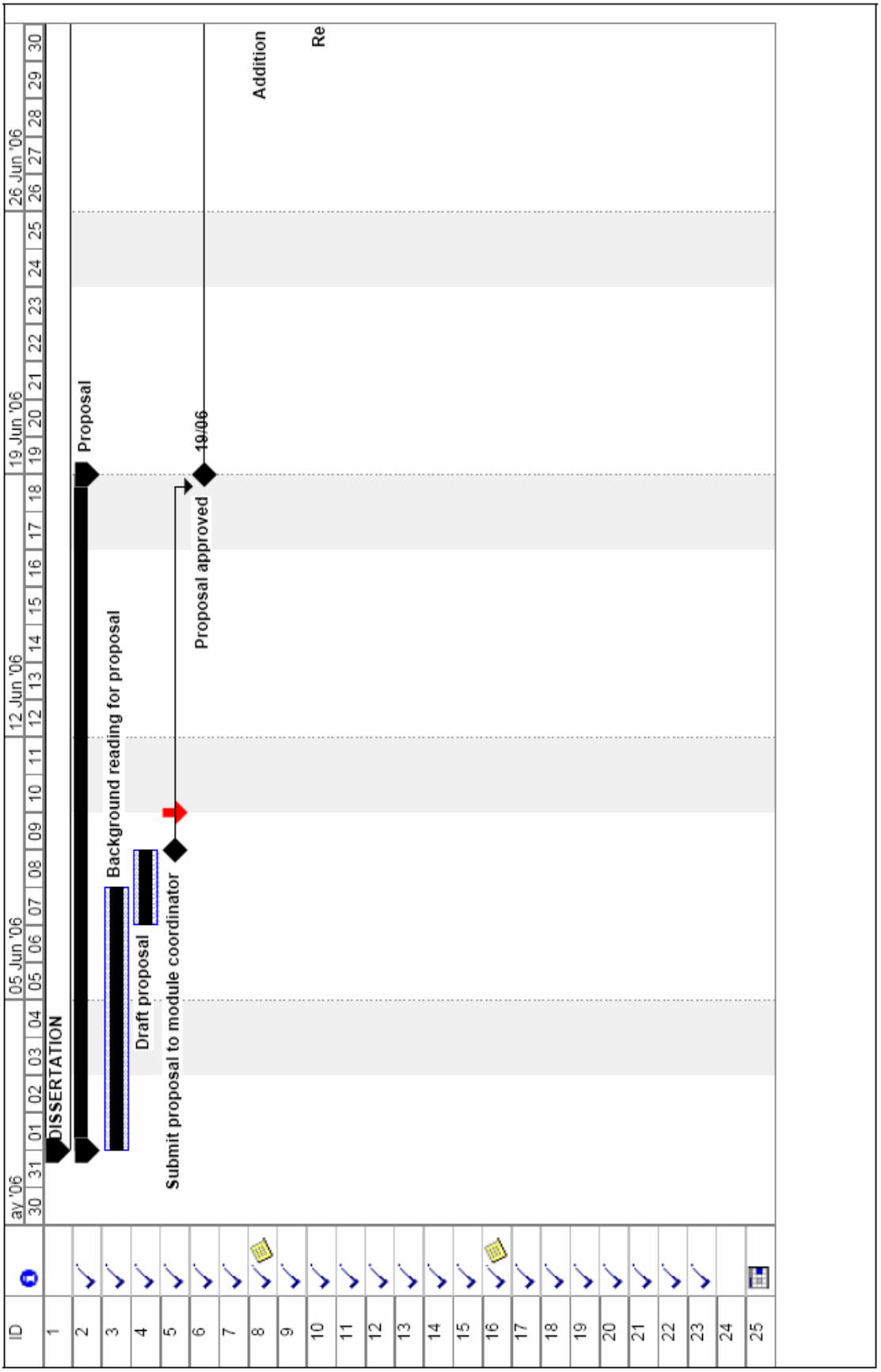
```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

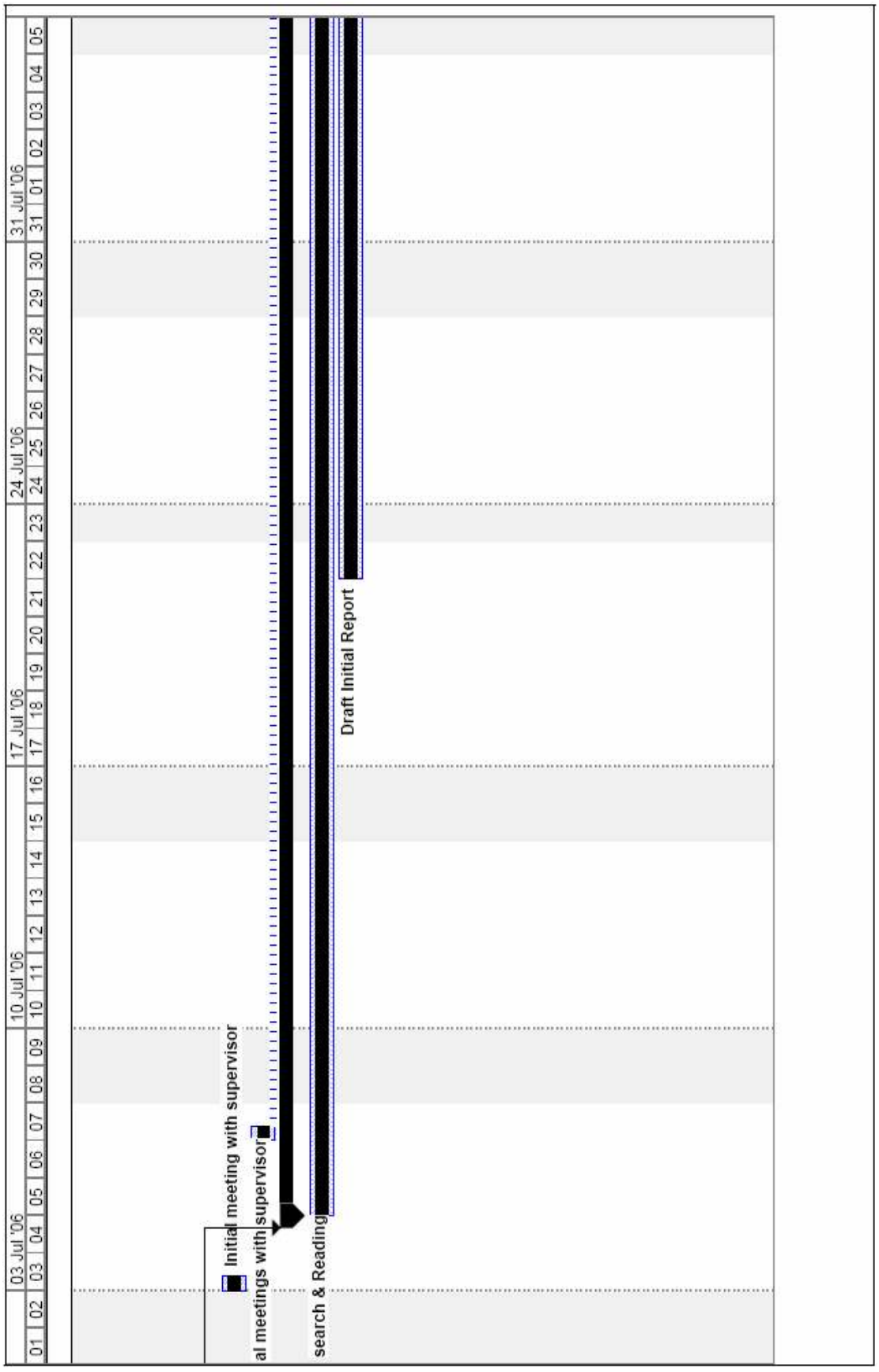
```
 <fo:layout-master-set>
  <fo:simple-page-master margin-right="1in" margin-left="1in" margin-
bottom="1in" margin-top="1in" page-width="11in" page-height="8.5in"
master-name="simple">
   <fo:region-body margin-top="0in"/>
  </fo:simple-page-master>
 </fo:layout-master-set>

 <fo:page-sequence master-reference="simple">
  <fo:flow flow-name="xsl-region-body">
   <fo:block border-after-color="cyan" border-after-width="4pt"
border-after-style="solid" space-after="12pt" background-color="blue"
color="white" font-family="sans-serif" font-size="20pt"> Discovered
Network Nodes
   </fo:block>
   <fo:block border-after-color="black" border-after-width="4pt"
border-after-style="solid" background-color="blue" color="white"
font-family="sans-serif" font-size="20pt">
    <fo:external-graphic src="windows.jpg"/>NEONFISH
   </fo:block>
   <fo:table font-size="10pt" table-layout="fixed" space-
after="12pt">
    <fo:table-column column-width="proportional-column-width(.5)"
column-number="1"/>
    <fo:table-column column-width="proportional-column-width(2)"
column-number="2"/>
    <fo:table-column column-width="proportional-column-width(2)"
column-number="3"/>
    <fo:table-column column-width="proportional-column-width(1)"
column-number="4"/>
    <fo:table-column column-width="proportional-column-width(.5)"
column-number="5"/>
    <fo:table-column column-width="proportional-column-width(1)"
column-number="6"/>
    <fo:table-body>
     <fo:table-row background-color="gray">
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>Index</fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>Description</fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>Interface Type</fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>Speed</fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>Status</fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>IP Address</fo:block>
       <fo:block>NetMask</fo:block>
```

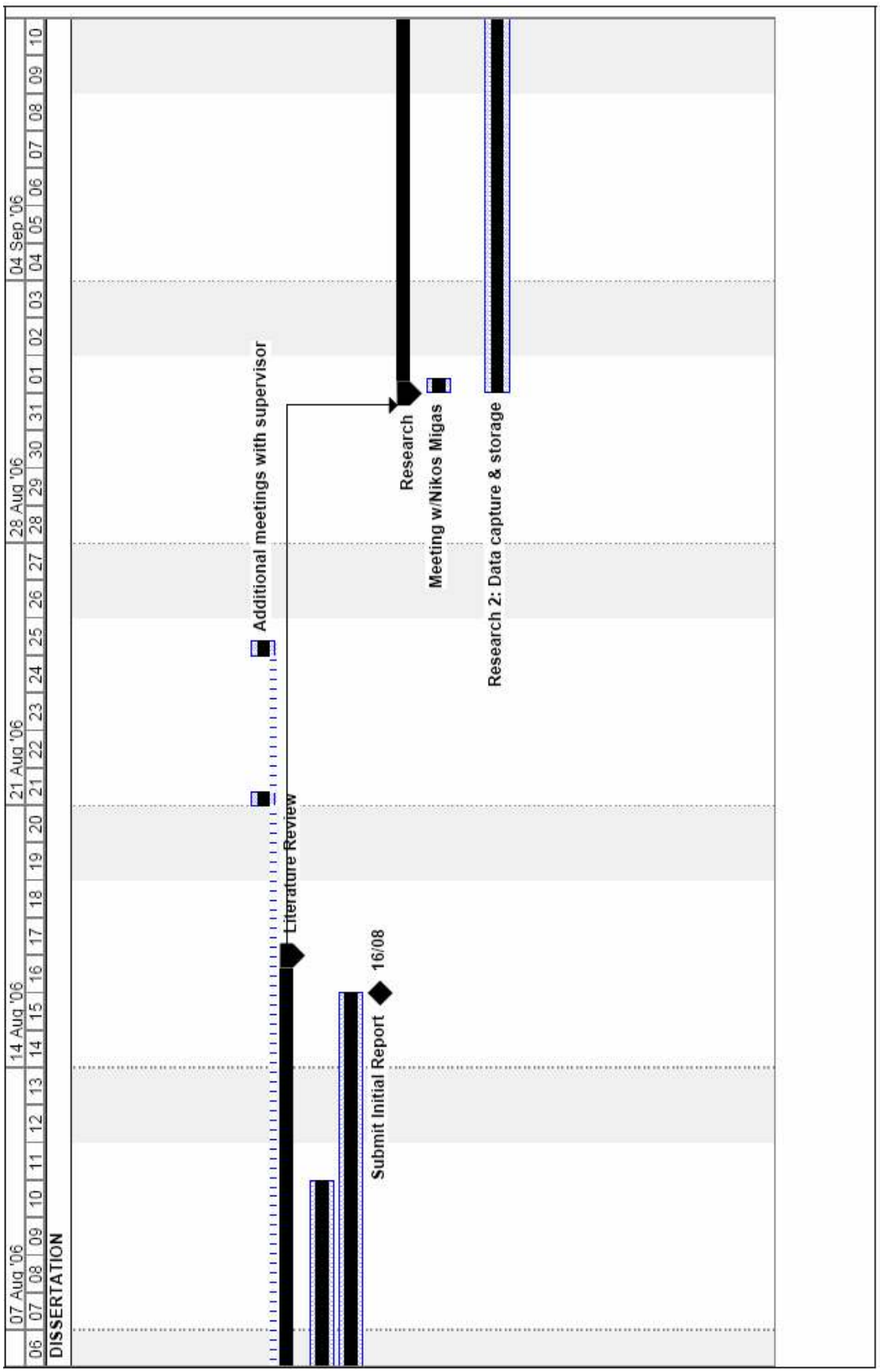
```
      </fo:table-cell>
     </fo:table-row>
     <fo:table-row>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>1</fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>MS TCP Loopback interface</fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>softwareLoopback</fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>10 Mbps</fo:block>
      </fo:table-cell>
      <fo:table-cell text-align="center" border="solid black 1px"
border-collapse="collapse">
       <fo:block>
        <fo:inline color="green">UP</fo:inline>
       </fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>127.0.0.1</fo:block>
       <fo:block>255.0.0.0</fo:block>
      </fo:table-cell>
     </fo:table-row>
     <fo:table-row>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>2</fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>Intel(R) PRO/Wireless 2200BG Network Connection -
Packet Scheduler Miniport</fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>ethernetCsmacd</fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>54 Mbps</fo:block>
      </fo:table-cell>
      <fo:table-cell text-align="center" border="solid black 1px"
border-collapse="collapse">
       <fo:block>
        <fo:inline color="red">DOWN</fo:inline>
       </fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>0.0.0.0</fo:block>
       <fo:block>0.0.0.0</fo:block>
      </fo:table-cell>
     </fo:table-row>
```

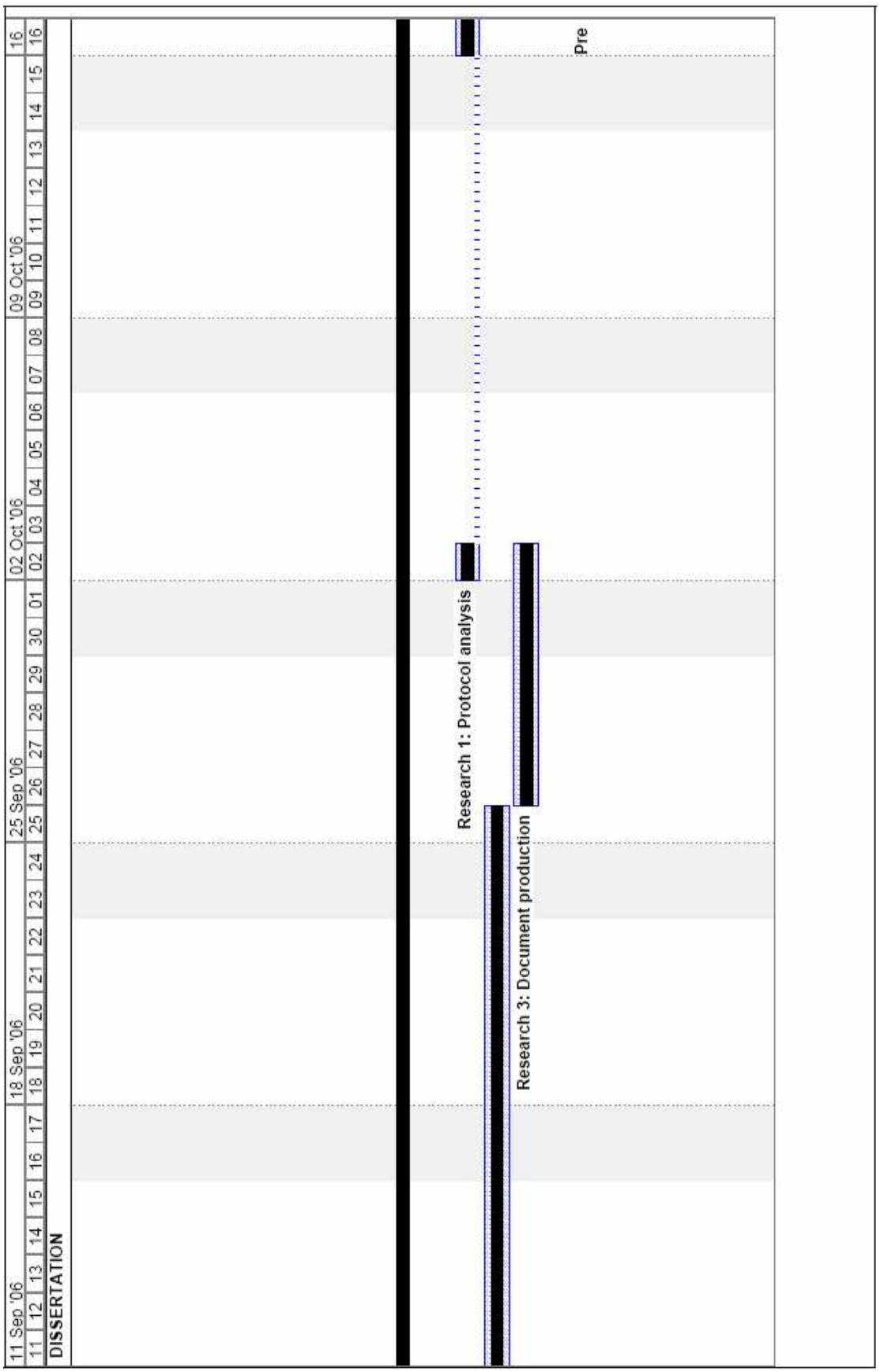
```
     <fo:table-row>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>65540</fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>Broadcom 440x 10/100 Integrated Controller - Packet
Scheduler Miniport</fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>ethernetCsmacd</fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>100 Mbps</fo:block>
      </fo:table-cell>
      <fo:table-cell text-align="center" border="solid black 1px"
border-collapse="collapse">
       <fo:block>
        <fo:inline color="green">UP</fo:inline>
       </fo:block>
      </fo:table-cell>
      <fo:table-cell border="solid black 1px" border-
collapse="collapse">
       <fo:block>192.168.0.3</fo:block>
       <fo:block>255.255.255.0</fo:block>
      </fo:table-cell>
     </fo:table-row>
    </fo:table-body>
   </fo:table>
                  <!-- TRUNCATED CONTENT -->
  </fo:flow>
 </fo:page-sequence>
</fo:root>
```

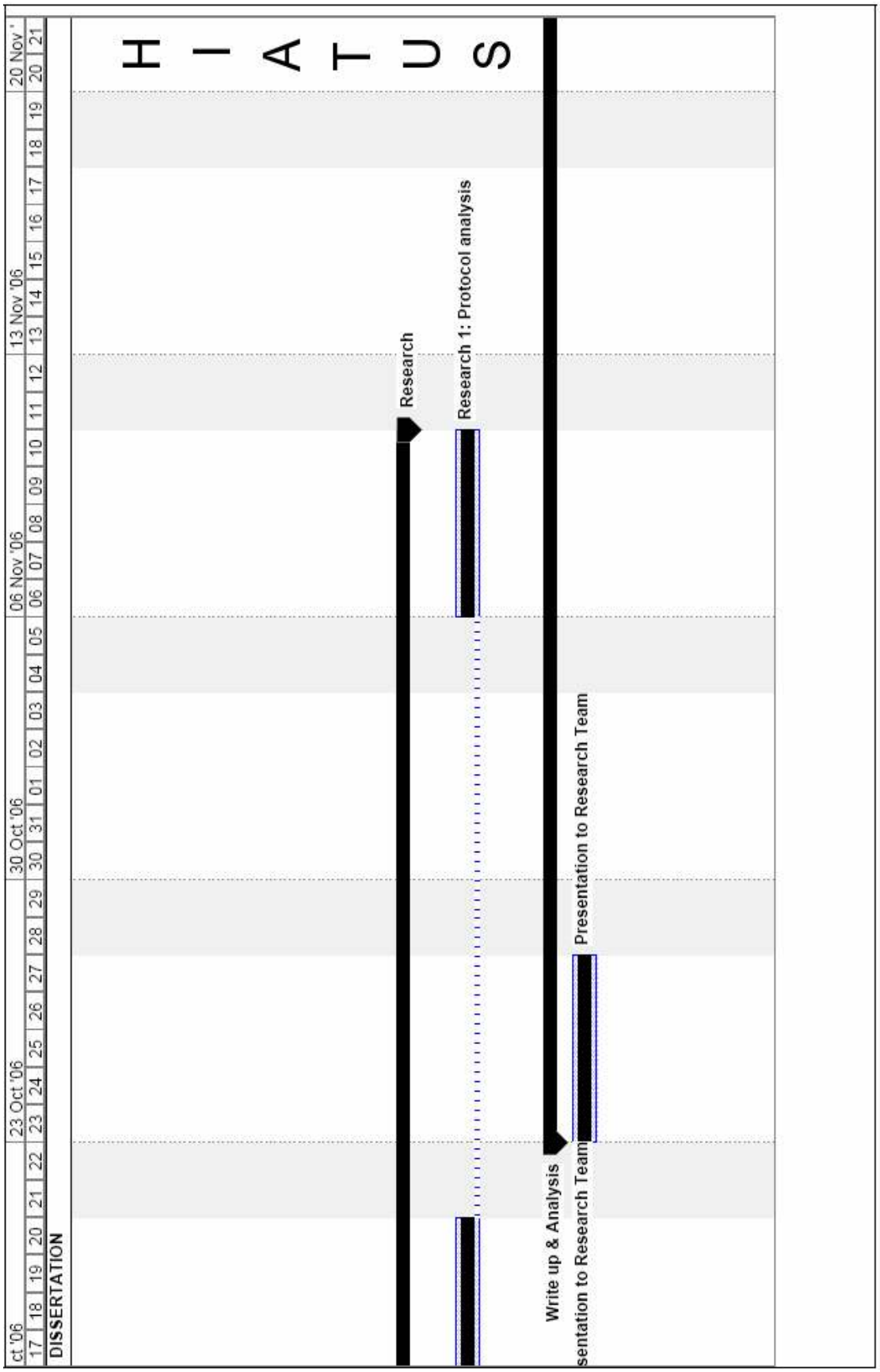
Project Management

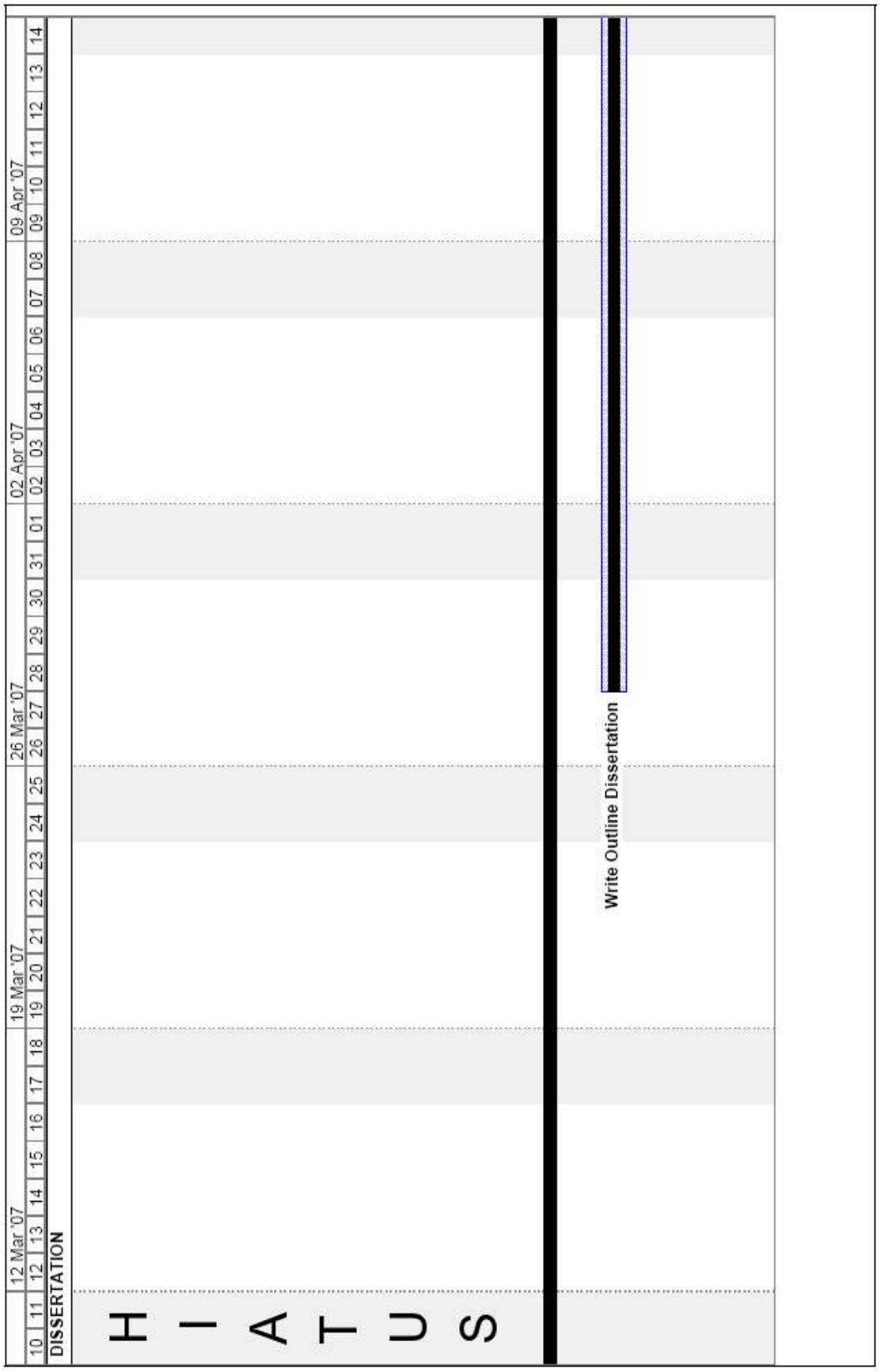


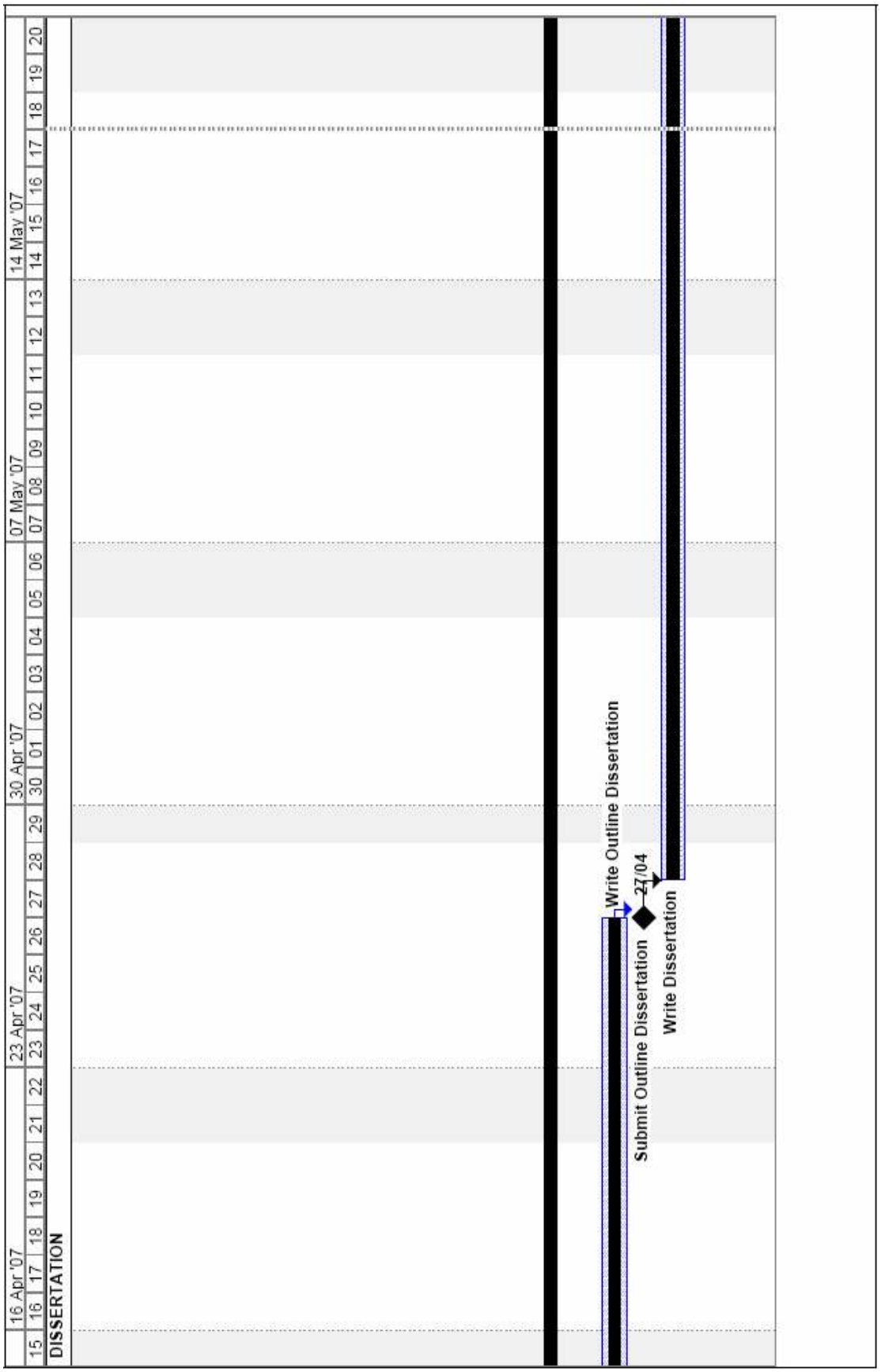

03 Jul '06
10 Jul '06
17 Jul '06
24 Jul '06
31 Jul '06
Initial meeting with supervisor
al meetings with supervisor
search & Reading
Draft Initial Report

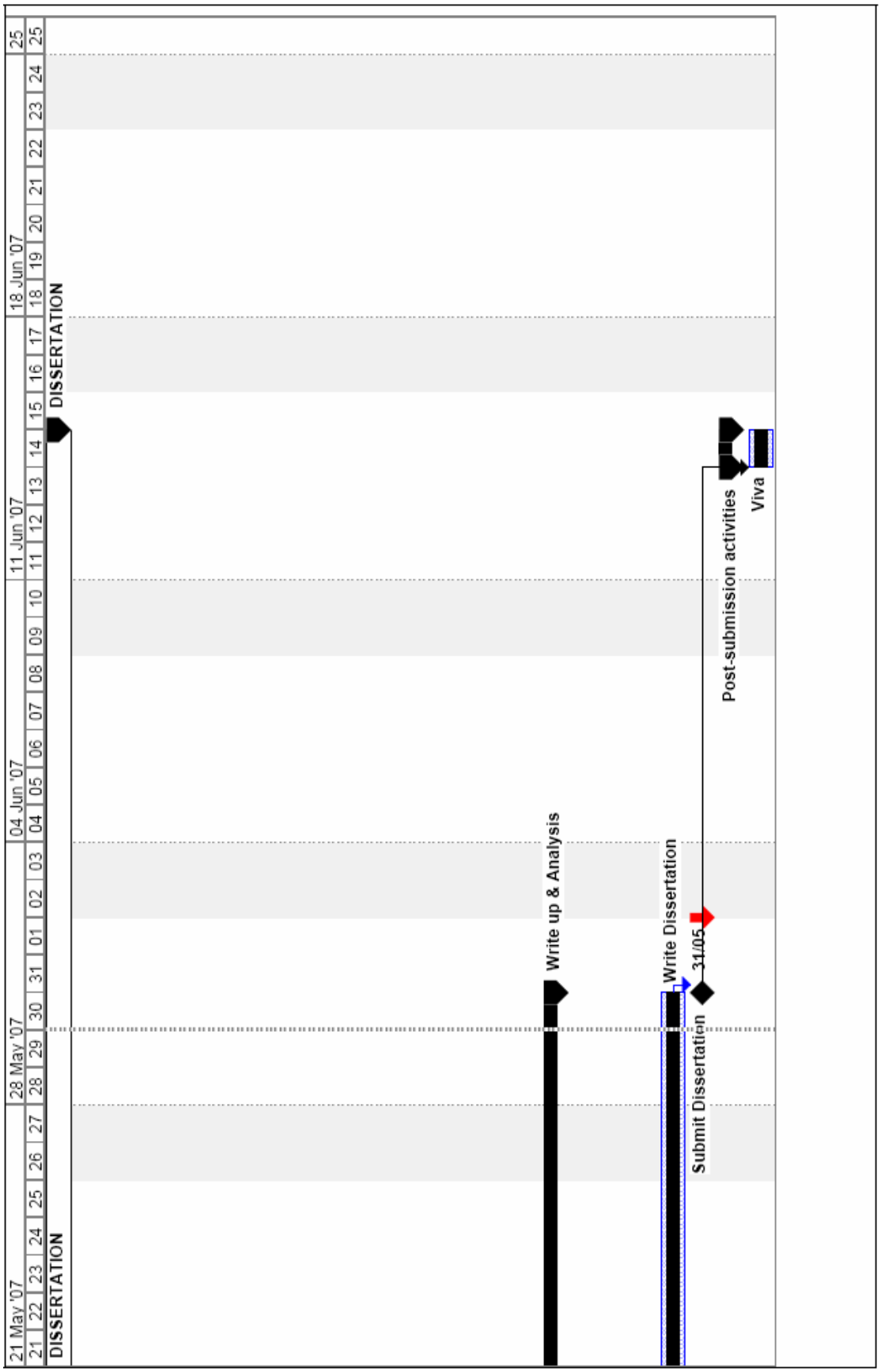

07 Aug '06
14 Aug '06
21 Aug '06
28 Aug '06
04 Sep '06
06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 01 02 03 04 05 06 07 08 09 10
DISSERTATION
Additional meetings with supervisor
Literature Review
Research
Meeting w/Nikos Migas
Research 2: Data capture & storage
Submit Initial Report
16/08


11 Sep '06
18 Sep '06
25 Sep '06
02 Oct '06
09 Oct '06
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16
DISSERTATION
Research 1: Protocol analysis
Research 3: Document production
Pre


ct '06
17 18 19 20 21 22
23 Oct '06
23 24 25 26 27 28 29
30 Oct '06
30 31 01 02 03 04 05
06 Nov '06
06 07 08 09 10 11 12
13 Nov '06
13 14 15 16 17 18 19
20 Nov '
20 21
DISSERTATION
H I A T U S
Research
Research 1: Protocol analysis
Write up & Analysis
sentation to Research Team
Presentation to Research Team


12 Mar '07
19 Mar '07
26 Mar '07
02 Apr '07
09 Apr '07
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 01 02 03 04 05 06 07 08 09 10 11 12 13 14
DISSERTATION
H I A T U S
Write Outline Dissertation


16 Apr '07
23 Apr '07
30 Apr '07
07 May '07
14 May '07
15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20
DISSERTATION
Write Outline Dissertation
27/04
Submit Outline Dissertation
Write Dissertation


21 May '07
28 May '07
04 Jun '07
11 Jun '07
18 Jun '07
DISSERTATION
DISSERTATION
Write up & Analysis
Write Dissertation
Submit Dissertation
31/05
Post-submission activities
Viva

References

Alderson, D., Li, L., Willinger, W., & Doyle, J. C. (2005). Understanding Internet Topology: Principles, Models, and Validation. *IEEE/ACM Transactions on Networking, 13*(6), 1205-1218. Retrieved 1 June, 2006 from ACM Digital Library at http://portal.acm.org/citation.cfm?id=1115527.

Anderson, D. P., & Fedak, G. (2006). The Computational and Storage Potential of Volunteer Computing. *Proceedings of the 2006 IEEE/ACM International Symposium on Cluster Computing and the Grid*, Singapore, 73-80. Retrieved 21 August, 2006 at http://boinc.berkeley.edu/papers.php.

Au, S. C., Leckie, C., Parhar, A., & Wong, G. (2004). Efficient visualization of large routing topologies. *International Journal of Network Management, 14*(2), 105-118. Retrieved 1 June, 2006 from ACM Digital Library at http://portal.acm.org/citation.cfm?id=987189.

Barrett, R., Kandogan, E., Maglio, P. P., Haber, E. M., Takayama, L. A., & Prabaker, M. (2004). Field Studies of Computer System Administrators: Analysis of System Management Tools and Practices. *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work,* Chicago, IL, USA, 388-395. Retrieved 13 June, 2006 from ACM Digital Library at http://doi.acm.org/10.1145/1031607.1031672.

Bierman, A. & Jones, K. (2000). *RFC 2922: Physical Topology MIB*. Retrieved 4 July, 2006 from The Internet Engineering Task Force at http://www.ietf.org/rfc/rfc2922.

Black, U. (1995). *Network Management Standards: SNMP, CMIP, TMN, MIBs, and Object Libraries* (2nd ed.). New York, NY, USA: McGraw-Hill, Inc.

BOINC Combined Statistics (2006). *BOINC Combined Statistics*. Retrieved 19 August, 2006 at http://boinc.netsoft-online.com.

Buchanan, M. C. & Zellweger, P. T. (2005). Automatic Temporal Layout Mechanisms Revisited. *ACM Transactions on Multimedia Computing, Communications and Applications, 1*(1), 60-88. Retrieved 13 June, 2006 from ACM Digital Library at http://doi.acm.org/10.1145/1047936.1047942.

Dietel, K. (2004). Mastering IT Change Management Step Two: Moving from Ignorant Anarchy to Informed Anarchy. *Proceedings of the 32nd Annual ACM SIGUCCS Conference on User Services*, Baltimore, MD, USA, 188-190. Retrieved 13 June, 2006 from ACM Digital Library at http://doi.acm.org/10.1145/1027802.1027846.

Dijker, B. (1998). *A Day in the Life of System Administrators*. Retrieved 24 July, 2006 from SAGE at http://www.sage.org/field/ditl.pdf.

DIMES (2006). The DIMES home/statistics and data. Retrieved 19 August, 2006 at http://www.netdimes.org/data.php.

Donnet, B., Raoult, P., Friedman, T., & Crovella, M. (2005). Efficient Algorithms for Large-Scale Topology Discovery. *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Banff, Alberta, Canada, 327-338. Retrieved 4 July, 2006 from ACM Digital Library at http://doi.acm.org/10.1145/1064212.1064256.

Han, J. Y. (2005). Low-cost multi-touch sensing through frustrated total internal reflection. *Proceedings of the 18th Annual ACM Symposium on User interface Software and Technology*, Seattle, WA, USA, 115-118. Retrieved 13 June, 2006 from ACM Digital Library at http://doi.acm.org/10.1145/1095034.1095054.

Hawkinson, J. & Bates, T. (1996). *RFC 1930: Guidelines for creation, selection, and registration of an Autonomous System (AS)*. Retrieved 22 August, 2006 from The Internet Engineering Task Force at http://www.ietf.org/rfc/rfc1930.txt.

Huffaker, B., Plummer, D., Moore, D., & Claffy, K. (2002). Topology Discovery by Active Probing. *Proceedings of the 2002 Symposium on Applications and the Internet*, Washington, D.C., USA, 90. Retrieved 18 August, 2006 from CAIDA at http://www.caida.org/publications/papers/2002/SkitterOverview.

IBM (2001). Autonomic Computing: IBM's Perspective on the State of Information Technology. Armonk, NY, USA: IBM. Retrieved 7 August, 2006 at http://www.ibm.com/industries/government/doc/content/resource/thought/278606109.html.

IEEE (2005). *802.1AB IEEE Standard for Local and metropolitan area networks: Station and Media Access Control Connectivity Discovery*. New York, NY, USA: Institute of Electrical and Electronics Engineers, Inc. Retrieved 19 September, 2006 at http://standards.ieee.org/getieee802/802.1.html.

ISO (1989). *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 4: Management framework*. International Organization for Standardization. Retrieved 8 June, 2006 at http://isotc.iso.org/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm.

Kephart, J. O. & Chess, D. M. (2003, January). The Vision of Autonomic Computing. *Computer 36*(1), 41-50. Retrieved 7 August, 2006 at http://www.research.ibm.com/autonomic/research/papers.

Lowekamp, B., O'Hallaron, D., & Gross, T. (2001). Topology Discovery for Large Ethernet Networks. *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications,* San Diego, CA, USA, 237-248. Retrieved 4 July, 2006 from ACM Digital Library at http://doi.acm.org/10.1145/383059.383078.

Madigan, E. M., Petrulich, C., & Motuk, K. (2004). The Cost of Non-Compliance: When Policies Fail. *Proceedings of the 32nd Annual ACM SIGUCCS Conference on User Services*, Baltimore, MD, USA, 47-51. Retrieved 13 June, 2006 from ACM Digital Library at http://doi.acm.org/10.1145/1027802.1027815.

McDonough, B. (2003). *netViz: Delivering Complex System Information Graphically*. Framingham, MA, USA: IDC. Retrieved 10 July, 2006 at http://www.netviz.com/forms/download_wp4.asp.

Nance, B. (n.d.). *Know Your Network*. Network Testing Labs. Retrieved 10 July, 2006 at http://www.netviz.com/forms/download_wp4.asp.

Nance, B. (2005). *Best Practices for IT Documentation*. Network Testing Labs. Retrieved 10 July, 2006 at http://www.netviz.com/forms/download_wp4.asp.

netViz Corporation (2003). *netViz – A New Approach to Information Visualization and System Modeling*. Gaithersburg, MD, USA: netViz Corporation. Retrieved 10 July, 2006 at http://www.netviz.com/forms/download_wp4.asp.

OGC, (2002). *ICT Infrastructure Management*. London: Great Britain Office of Government Commerce.

Parker, J. (2005). *FCAPS, TMN, & ITIL: Three Key Ingredients to Effective IT Management*. OpenWater Solutions, LLC. Retrieved 7 June, 2006 at http://www.openwatersolutions.com/docs/FCAPS_TMN_%20ITIL.pdf.

Patterson, D. A., Brown, A., Broadwell, P., Candea, G., Chen, M., Cutler, J., Enriquez, P., Fox, A., Kiciman, E., Merzbacher, M., Oppenheimer, D., Sastry, N., Tetzlaff, W., Traupman, J., & Treuhaft, N. (2002, March 15). Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies. *UC Berkeley Computer Science Technical Report UCB//CSD-02-1175*. Retrieved 7 August, 2006 at http://roc.cs.berkeley.edu/#pubs.

Pawson, D. (2002). *XSL-FO*. Sebastopol, CA, USA: O'Reilly Media, Inc.

Rainge, E. (2006). *Making the Business Case for Deploying Data Visualization and Network Diagramming Tools*. Framingham, MA, USA: IDC. Retrieved 10 July, 2006 at http://www.netviz.com/forms/download_wp4.asp.

Ratliff, E. (2005). The Zombie Hunters. *The New Yorker, 10 October, 2005*, 44-49.

Ray, E. T. (2003). *Learning XML* (2nd ed.). Sebastopol, CA, USA: O'Reilly Media, Inc.

Shavitt, Y. & Shir, E., (2005). DIMES: Let the Internet Measure Itself. *ACM SIGCOMM Computer Communication Review 35*(5), 71-74. Retrieved 4 July, 2006 from ACM Digital Library at http://doi.acm.org/10.1145/1096536.1096546.

Siamwalla, R., Sharma, R., & Keshav, S. (1998). *Discovering Internet Topology*. Retrieved 4 July, 2006 at http://www.cs.cornell.edu/skeshav/papers/discovery.pdf.

Tidwell, D. (2001). *XSLT*. Sebastopol, CA, USA: O'Reilly Media, Inc.

University of California, Berkeley (2006). *SETI@home*. Retrieved 19 August, 2006 at http://setiathome.berkeley.edu.

University of Oregon (2005, January 25). *Route Views Project Page*. Retrieved 18 August, 2006 at http://www.routeviews.org.

Vixie, P., Thomson, S., Rekhter, Y., & Bound, J. (1997). *RFC 2136: Dynamic Updates in the Domain Name System (DNS UPDATE)*. Retrieved 30 April, 2007 from The Internet Engineering Task Force at http://tools.ietf.org/html/rfc2136.

Whitehouse, D. (1999, May 25). Interstellar message says 'ET call Earth'. *BBC News Online*. Retrieved 19 August, 2006 at http://news.bbc.co.uk/1/hi/sci/tech/351461.stm.

Wikipedia (2007a). *Smurf attack*. Retrieved 11 April, 2007 at http://en.wikipedia.org/wiki/Smurf_attack.

Wikipedia (2007b). *Synchronized Multimedia Integration Language*. Retrieved 8 May, 2007 at http://en.wikipedia.org/wiki/Synchronized_Multimedia Integration_Language.

Bibliography

Bejerano, Y., Breitbart, Y., Garofalakis, M., & Rastogi, R. (2003). Physical Topology Discovery for Large Multi-Subnet Networks. *The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, San Francisco, CA, USA. Retrieved 24 July, 2006 from ACM Digital Library at http://www.comsoc.org/confs/ieee-infocom/2003/papers/09_02.PDF.

Brietbart, Y., Garofalakis, M., Jai, B., Martin, C., Rastogi, R., & Silverschatz, A. (2004). Topology Discovery in Heterogeneous IP Networks: The NetInventory System. *IEEE/ACM Transactions on Networking, 12*(3), 401-414. Retrieved 13 June, 2006 from IEEE Xplore at http://dx.doi.org/10.1109/TNET.2004.828963.

Czerwinski, M., Horvitz, E., & Wilhite, S. (2004). A Diary Study of Task Switching and Interruptions. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 6*(1), 175-182. Retrieved 13 June, 2006 from ACM Digital Library at http://doi.acm.org/10.1145/985692.985715.

Harold, E. R., & Means, W. S., (2004). *XML in a Nutshell* (3rd ed.). Sebastopol, CA, USA: O'Reilly Media, Inc.

Kamoun, F. (2005). Toward best maintenance practices in communications network management. *International Journal of Network Management, 15*(5), 321-334. Retrieved 13 June, 2006 from Wiley InterScience at http://dx.doi.org/10.1002/nem.576.

McDonough, B. (2004). *netViz : Visually Presenting Complex Network and Business Process Information*. Framingham, MA, USA: IDC. Retrieved 10 July, 2006 at http://www.netviz.com/forms/download_wp4.asp.

Opsware Inc. (n.d.). *Network Automation: A fundamental shift in network management*. Sunnyvale, CA, USA: Opsware Inc. Retrieved 14 July, 2006 at http://www.opsware.com/products/networkautomation/NetworkAutomation.pdf.

Wikipedia (2007a). *Information Technology Infrastructure Library*. Retrieved 13 May, 2007 at http://en.wikipedia.org/wiki/ITIL.