A Novel Adaptive and Efficient Routing Update Scheme for Low-power Lossy Networks in IoT

Baraq Ghaleb, Ahmed Al-Dubai, Elias Ekonomou, Imed Romdhani, Youssef Nasser, and Azzedine Boukerche

Abstract—**In this paper, we introduce Drizzle, a new algorithm for maintaining routing information in the Low-power and Lossy Networks (LLNs). The aim is to address the limitations of the currently standardized routing maintenance (i.e. Trickle algorithm) in such networks. Unlike Trickle, Drizzle has an adaptive suppression mechanism that assigns the nodes different transmission probabilities based on their transmission history so to boost the fairness in the network. In addition, Drizzle removes the listen-only period presented in Trickle intervals leading to faster convergence time. Furthermore, a new scheme for setting the redundancy counter has been introduced with the goal to mitigate the negative side effect of the short-listen problem presented when removing the listen-only period and boost further the fairness in the network. The performance of the proposed algorithm is validated through extensive simulation experiments under different scenarios and operation conditions. In particular, Drizzle is compared to four routing maintenance algorithms in terms of control-plane overhead, power consumption, convergence time and packet delivery ratio (PDR) under uniform and random distributions and with lossless and lossy links. The results indicated that Drizzle reduces the control-plane overhead, power consumption and the convergence time by up to 76%, 20% and 34% respectively while maintaining approximately the same PDR rates.**

Keywords—**RPL, Trickle algorithm, Low power and lossy networks, Load balancing, IoT.**

I. Introduction

The ever-tighter integration of physical world with computing has given birth to a new communication paradigm referred to as the Internet of Things (IoT) [1][2]. One of the building blocks of the IoT is the *Low-power and Lossy Network* (LLN), a collection of interconnected embedded devices, such as sensor nodes, typically characterized by constraints on both node resources and underlying communication technologies. The introduction of IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN) protocol has addressed the gap between these tiny devices and the Internet enabling such integration [1]. The network limitations in terms of energy, memory, and processing resources impose a set of challenges to design efficient routing protocols for LLNs [2]. In fact, various efforts have been made by the IETF Routing Over LLNs (ROLL) working group to address such issues. The Routing Protocol for LLNs (RPL) [2], the Collection Tree Protocol (CTP) [3], the Hybrid Routing Protocol for LLNs (Hydro) [4] and the Lightweight On-demand Ad hoc Distance vector routing protocol – Next generation (LOADng) [5] are among the latest standards proposed by research community. A key design principle of any routing protocol is to have an efficient mechanism for disseminating routing information through the network, and maintaining up-to-date information. One of the mechanisms to perform this task is to propagate the routing information periodically, which is widely used in unconstrained wired networks. When adopting such proactive update and maintenance scheme in resource-constrained large-scale LLNs, the performance of such networks decreases dramatically due to high traffic overhead [6][7].

To address this issue, the IETF ROLL group proposed Trickle algorithm to regulate the emission of routing information in LLNs [8][9]. The basic idea behind Trickle is to equip resource-constrained nodes with a simple and energy-efficient primitive for disseminating routing information throughout the network. Trickle uses two mechanisms to achieve this goal. The first mechanism is to increase adaptively the signaling rate upon detecting a new routing information. In contrast, it exponentially reduces the signaling rate when the network state is up-to-date in order to save energy and bandwidth. The second is the suppression mechanism in which a node suppresses the transmission of its routing information if it detects that enough number of its neighbors have transmitted the same piece of information. The main issue with Trickle is its code propagation technique, which in one way or another has different characteristics in comparison with routing maintenance in a routing protocol. This is especially in the context of LLNs [10], which exhibit the features of Scarcity of resources, links unreliability, load balancing and Dynamic and various densities that will be addressed, in turn, below.

Scarcity of resources: The resource-constrained nature of LLNs imposes new restrictions on developing an efficient algorithm for disseminating routing information through such networks. Generally speaking, the small-battery size of a sensor node is the most restrictive factor and should be taken in a great consideration. A routing maintenance primitive should opt to send just enough updates to ensure the freshness of the constructed routes. Sufficient route updates can vary from transmitting one update every second to every bulk of minutes through the network lifetime depending on the current conditions of the network to ensure that application energy requirement is met [7].

Link unreliability: LLNs are characterized by lossy and unreliable links, and an update is not guaranteed to reach its destination when it is sent for the first time [7]. In some cases, the link loss rate in a network cannot be predicted beforehand, and even worse, the same link may exhibit different loss rate over time due to several different factors such as collisions at the receiver, hidden terminal problem and interference with other radios of neighboring sensor nodes [8].

Load balancing: a network can benefit from the presence of load-balancing mechanism among its sensor nodes in two ways: first, it enables the network to discover all the possible routes available for routing and second it distributes the load evenly in the network to maximize the network lifetime. In other words, the absence of load balancing may render some routes undiscoverable even though they might be more efficient than those already active in the network [10].

Dynamic and various densities: it is envisioned that LLNs would be deployed using different density scenarios, ranging from a few neighbors per node to hundreds. The density of a specific deployment may or may not be known in advance. Thus, the algorithm should be tailored to handle all cases and its parameters should be tuned according to the specific case it encounters. In other words, the algorithm has to be scalable and dynamic.

Considering the above-mentioned features, this paper proposes a new algorithm for disseminating routing information in LLNs, namely, the Drizzle algorithm. Elevating the shortcomings of the previous algorithms, Drizzle has a solid and configurable nature that makes it suitable for various application requirements. More specifically, it offers an adaptive suppression mechanism that permits the nodes to have different transmission probabilities, which are consistent with their transmission history. It removes also the listen-only period to fasten the convergence time and implements a new policy for setting the redundancy coefficient.

The rest of the paper is organized as follows: Section II presents an overview of related work. A detailed description of Drizzle algorithm is presented in Section III highlighting its main principles. Section IV introduces detailed description of the simulation environment and the obtained results. Finally, Section V overviews the entire study and then presents conclusions and recommendation for future work.

II. Related Work and State-of-The-Art

Ad-hoc routing strategies such as the Dynamic Source Routing (DSR) [11][12], the Ad hoc On-Demand Distance Vector (AODV) [13], the Optimized Link-state (OLSR) [14][15], and the Open Shortest Path First (OSPF) routing protocols [16], have been found to be unsatisfying for the unique routing requirements in LLNs [17]. Therefore, several standard bodies have assigned different working groups in order to develop the necessary protocols and standards that meet the new requirements imposed by LLNs. For instance, the MANET working group has developed the Dynamic MANET On-Demand Routing (DYMO) [18] and OLSRv2 [19] as successors for AODV and OLSR for routing in LLNs [20]. Additionally, 6LowPAN (IPv6 over low power WPAN) working group has presented several routing proposals including the 6LoWPAN Ad hoc On-Demand Distance Vector Routing (LOAD) [5]. Finally, the 6LowPAN delegated the ROLL working group for advising an efficient LLNs routing protocol. Their efforts has culminated in producing the proactive Routing Protocol for LLNs (RPL). A primary constituent part of any routing protocol is how to update and maintain the routing information in order to keep the network routing states up-to-date and insure the freshness of the active routes. In the reactive routing protocols such as AODV and LOAD, the route maintenance process is simplified as it is only triggered when a node has a data packet to send and, thus, there is no need for a periodic update of routing information. On the other hand, proactive routing protocols such as OLSRv2 and RPL use more complex route maintenance process as the routes are created, and thus need to be maintained regularly. The motivation behind the prior constructing of network topology is that it enables the data packet to be sent immediately, avoiding unnecessary delays. Each reactive routing protocol has its own mechanism to handle the routing maintenance process. For instance, OLSRv2 maintains its state by having each router transmits HELLO messages proactively at a regular rate. The rate which HELLO messages are transmitted at, may be constant or dynamic, for example, it might be backed off due to transmission problems such as collisions, congestion or stability of the network. The Babel [21] routing protocol uses a more sophisticated mechanism for updating the routing information. First, each Babel speaker propagates its routes every specified interval of time. Second, upon discovering that a significant change in network topology has occurred, Babel speakers advertises what they called “a triggered update” in a timely manner in order to alert the network of this abrupt change. A major problem associated with the periodic update of routing information is that every sensor node must advertise a regular routing updates, even though, there is no change in the routing information. This will result in an excessive use of the battery power in addition to generating unnecessary routing overhead, which in return affects negatively the network performance.

A recent approach for updating routing information in LLNs is that adopted by RPL routing protocol, namely, *Trickle algorithm*. The basic idea behind Trickle is to equip the nodes with a simple, yet scalable and energy-efficient primitive for exchanging routing information. Trickle relies on two primary mechanisms to disseminate efficiently the routing information. The first mechanism is to change adaptively the signaling rate according to the conditions that are currently present in the network. The second is the suppression mechanism in which a node blocks the transmission of its control packet if it detects that it is redundant. The adaptive signaling rate in addition to suppressing redundant information enables the network to use its available resources efficiently, consequently save energy and bandwidth. However, several research studies have recently reported some issues that limit the efficiency of Trickle algorithm in LLNs. For instance, the study in [22] has indicated that introducing the listen-only period in the first half of each Trickle interval (I) would exhibit growing delay while propagating transmissions intended to resolve the discovered inconsistency in routing information.

In fact, the goal behind introducing the listening period is to solve the so-called short-listen problem in asynchronous networks. In the asynchronous network with no listen-only period, a node may start emitting its current DIO message (DODAG Information Object) very soon after starting a new interval, a behavior that may result in turning down the suppression mechanism in the current interval and the subsequent intervals leading to significant redundant transmissions and, thus, limiting the algorithm scalability [8]. However, introducing the listen-only period has its own shortcomings. Firstly, this period will impose a delay of at least half of the interval before trying to propagate an update. In *m-hop* network, the inherited delay will be progressively accumulated at each hop resulting in an overall delay proportional to the number of hops. Secondly, this period may also result in uneven load distribution with some nodes transmitting less than others. In the worst-case scenario, the transmission period of a node may completely overlap with the listen-only period of another neighboring node consequently, forbidding that node from transmitting for a long time.

A major issue in this scenario is that the forbidden node might be a critical node whose transmission is vital for resolving network inconsistences. Consequently, this will have a negative impact on the convergence time of the network. In addition, the absence of load balancing scheme may render some routes undiscoverable even though they might be more efficient than the active paths, which may affect the network reliability. Pertaining to Trickle's suppression mechanism, it is shown that the incorrect configuration of the redundancy constant may lead to creating sup-optimal routes especially in heterogeneous topologies composed of regions of different densities [10]. This is attributed to the fact that Trickle is originally designed to disseminate code updates, which are quite similar in the context of reprograming protocols. However, this is not the case in the context of routing as two routing update messages originated from different sources may carry different routing information and thus "*suppressing one transmission or another is not always equivalent*" [10]. To address the aforementioned issues, several routing maintenance primitives have been proposed. For instance, the study in [10] proposes an enhanced version of Trickle named Trickle-F in an attempt to guarantee a fair multicast suppression among RPL nodes. Trickle-F gives each node a priority to send its scheduled DIO based on how many consequent DIOs have been suppressed recently. In other words, the more the node suppresses its DIO, the higher the chance it would transmit in the next interval frame. The proposed enhancement is compared to the original Trickle under RPL by means of simulations and in terms of network stretch, average energy consumption and the distribution of suppressed messages. The evaluation results show that Trickle-F has managed to reduce the number of nodes with sup-optimal routes while shown the same energy consumption profile. Although Trickle-F has succeeded to some extent in solving the sub-optimality of constructed routes, the algorithm still suffers from the slow convergence time due to the listen-only period and higher overhead due to the un-adaptivity of its suppression mechanism.

The work in [23] highlights the ambiguity associated with configuring the redundancy parameters *k* in RPL-based networks. For instance, Trickle RFC [9] states that the typical values for *k* are 1-5, while RPL RFC [2] has set 10 as the default for *k*. However, the adequate value for the redundancy constant is claimed to be between three and five in the last IETF draft titled "*Recommendations for Efficient Implementation of RPL*" [24]. Finally, it is recommended in the RFC of the Multicast Protocol for Low-Power and Lossy Networks (MPL) to set the default value of *k* to one [25]. The different recommendations for setting the redundancy constant indicates that its optimal setting is not trivial task and relies greatly on application scenario. Thus, the authors propose a new algorithm named *adaptive-k* in which they suggest setting the value of *k* for each node individually based on that node degree (density). They used the number of Trickle messages received during a specific window as an implicit indication of that node degree. It was shown by simulations and testbed experiments that adaptive-*k* improves the performance of RPL in terms of control-plane overhead while discovering more optimal routes. However, it is unclear why the study resorts to the number of messages and not the number of actual neighbors, received at specific node to indirectly estimate the network density at that node. Although this method might give approximately accurate estimation for the node degree when the network is characterized by synchronized intervals among its nodes, it may suffer from an inaccurate estimation in asynchronized networks. For instance, in asynchronized network, the frequency of transmission may differ significantly from a node currently in its minimal interval to another node currently in its maximum interval. Hence, the former node will transmit more frequently giving the receiver node the impression that it has more neighbors than it actually has affecting negatively the accuracy of the network density estimation at that node.

In [26], it has been shown by mathematical analysis that the single redundancy constant adopted by Trickle may result in higher transmission load and consequently higher power consumption rates for those nodes having less number of neighbors. To alleviate this issue, the study proposes an enhancement of Trickle in which each node calculates its own version of the redundancy constant as function of its degree. Each node with a number of neighbors less than a pre-specified threshold called the *offset* will set its redundancy constant to *one*. The redundancy constant of other nodes is set by subtracting the number of neighbors from the offset and getting the ceiling of dividing the result by another predetermined value called the *step*. The simulations show that the proposed algorithm has balanced the transmission distribution among network nodes in comparison with the standard Trickle. However, the study does not demonstrate the impact of the proposed enhancement either on the quality of constructed routes neither on the network power consumption. In addition, introducing two new parameters, the *step* and the *offset*, will further add a complexity on how to configure Trickle parameters which is to be avoided.

In [22], the authors highlight the problem of increased latency resulting from introducing the listen-only period. To address this problem, an optimized version of Trickle, namely opt-Trickle, is proposed. The authors point out that the nodes receiving inconsistent transmissions simultaneously will reset their timers (returning to I_{min}) immediately, consequently exhibiting a form of an implicit synchronization in the first interval among these nodes. Such synchronization will eliminate the need for the listen-only period in the first interval and allow the respected nodes to pick the random time, *t*, from the range [0, I_{min}], which is their only modification to the original Trickle. However, this study assumes a MAC protocol with 100% duty-cycle, which is neither reasonable nor realistic.

Table I. Summary of Trickle extensions

The name	Brief description
Trickle-F	Gives the node a priority to send its scheduled DIO based on its recent history of transmission.
opt-Trickle	Allows nodes to pick the random time, *t*, from the range *[0, Imin]* in the first interval.
adaptive-k	Allow each node to tune its redundancy factor dynamically based on the number of its neighbors
Trickle-offset	Calculate the redundancy factor as a function of node degree.

Furthermore, opt-Trickle still has a listen-only period in the subsequent intervals that will contribute to the increased latency especially in a lossy network where it is not guaranteed that the firstly transmitted multicast message will reach all of its destinations. Other studies has focused on the modeling and analysis aspects of Trickle [27][28][29][30]. Table I summarizes the Trickle's different solutions.

In this paper, a new algorithm for maintaining the network topology in LLNs is introduced to address Trickle limitations, namely, Drizzle algorithm. This is an extended version of our previous work in [31] in which we evaluated Drizzle under restricted scenarios (i.e. we compared only to Trickle and under uniform distributions). In this extended version, three more Trickle's extensions [10][22][23] have been implemented, analyzed and compared to Drizzle highlighting the major differences and similarities among the compared protocols and under both random and uniform distribution with lossless and lossy links.

III. The Proposed Drizzle Algorithm

Compared to Trickle, Drizzle has many distinguishing features and different policies that endorse its superiority as a promising solution for routing maintenance in LLNs. Drizzle differs in two major ways. First, the suppression mechanism in Drizzle is adaptive so that the nodes have the capacity to adjust their transmission probability according to their transmission history. This, in one hand, relieves the network administrator from the concern of configuring the redundancy coefficient. On the other hand, it will ensure the fairness of the algorithm, as the nodes that have transmitted more in the previous intervals would have less probability to send in the current interval. The fairness of the algorithm has been further supported by assigning each node a transmission slot within each interval also depending on their transmission history. Second, Drizzle eliminates the listen-only period presented in Trickle intervals so that each node can schedule its transmission at any point throughout the interval rather than the second half only. This would enable the nodes to contend in a wider window reducing the collision probability. Another advantage of this primitive is that any change in the network state will have the chance to be propagated more rapidly than in other techniques such as in Trickle algorithm. In this regards, Drizzle uses the same number of parameters used by Trickle and seven maintaining-state variables. In what follows, we define the parameters used by Drizzle to configure its timeline.

Definition 1: The minimum interval length (I_{min}): This is the fastest transmission rate in time units when a significant change in the network has been discovered (inconsistency).

Definition 2: The maximum interval length (I_{max}): This is the slowest transmission rate in time units of a node in the steady state.

Definition 3: The redundancy factor (k): represents the number for received consistent messages that a node should receive during a specific period before suppressing its own transmission.

In addition, Table I outlines the seven variables used by Drizzle to maintain its current state.

Table II: Drizzle Variables

Variable	Meaning
s	This represents the number of DIO transmissions by a specific node until that node resets Drizzle to its minimum interval (i.e. a counter to count number of transmitted DIO that is reset to zero when entering the minimum interval).
n	This counter keeps a track of the number of intervals between two resets to the minimum interval.
rFlag	This is a flag that is set to 0 or 1 according to the case that produced the inconsistency state.
ck	This variable represents the current value of the redundancy coefficient as the node increases or decreases it.
I	Length of the current interval in time units (e.g. seconds).
t	This is the time slot selected by a node within the current interval, at which that node may transmit its scheduled DIO.
c	Message counter to keep a track of number of received consistent messages within the current interval.

The following steps illustrates in details the operations of Drizzle algorithm whereas the algorithm pseudo-code is presented in Algorithm 1:

1. Drizzle starts its operation by setting its first interval to I_{min}, and the redundancy value, *ck*, to the initial value of the redundancy coefficient, *k*. It also set the broadcasted messages number, *s*, and the consistency counter, *c*, to zero. Finally, it sets the *rFlag* and the number of intervals, *n*, to one.
2. On the beginning of each interval Drizzle assigns a randomly selected value in the interval to the variable, *t*, taken from the range:

$$[\, s * \frac{I}{n}, (s+1) * \frac{I}{n}\,] \qquad (1)$$

3. Upon receiving a consistent message, Drizzle increments its consistency counter by one.
4. When a node running Drizzle detects inconsistency state, Drizzle resets its timer by setting *I* to I_{min}, if it was not already set, resets the interval counter, and the message counter to zero while it resets the value of interval counter to one. It also sets the value of the *rFlag* to either one or zero according to the case that produced the inconsistency. We limit the cases in which the *rFlag* is set to one to only three cases: (a) when the root establishes the construction of the DODAG, (b) when the root initiates a global repair, and (c) when a node firstly joins the DODAG.
5. At the randomly selected time, if the consistency counter

is less than the redundancy coefficient, Drizzle transmits its scheduled message; otherwise, the message is suppressed. At this time, Drizzle also resets the consistency counter to zero.

6. If the scheduled message has been transmitted, Drizzle increases the broadcasted messages number by one. It also decrements the redundancy coefficient current value by one. If the value of redundancy coefficient would be less than zero, Drizzle sets it to zero.
7. If the scheduled message has been suppressed, Drizzle increments the redundancy coefficient current value by one. If its value would exceed the initial value of the redundancy coefficient, k, Drizzle sets it to k.
8. Once the interval I expires, Drizzle decreases its transmission rate through doubling the length of the interval providing that the *rFlag* value is one. If the value of the *rFlag* is equal to zero, Drizzle decreases its transmission rate through entering directly the slowest transmission rate. In all cases, if the size of the new interval would exceed the I_{max}. Drizzle sets the interval size I to I_{max} and re-executes the steps from step 2. The interval counter, then, is increased by one.

IV. Performance Evaluation and Discussion

In this section, we present an analytical analysis for the proposed algorithm highlighting its main advantages over the standardized algorithm for LLNs.

A. Rapid Propagation

One of the observable issues presented in the standardized algorithm (i.e. Trickle) for routing maintenance in LLNs is introducing the listen-only period in the first half of each interval with the goal to solve the so-called short-listen problem in asynchronous networks. The short-listen problem may turn down the suppression mechanism of Trickle resulting in significant redundant transmissions and, thus, limiting the algorithm scalability [8]. This short-listen problem is illustrated in Fig. 1with three nodes (N1, N2, N3) operating Trickle without the listen-only period and $k=2$. You can notice that none of the three nodes has managed to suppress any DIO due to the short-listen problem as each node begins transmitting directly after starting its new interval and resetting its redundancy counter to zero. Trickle introduces the idea of listen-only period in which a node must select, t, from the second half of the interval to avoid the short-listen problem. However, introducing the listen-only period comes with its own shortcomings. First, the listen-only period will impose a delay of at least $I/2$ (i.e. half of the interval) before trying to propagate the new information. In *m-hop* network, the inherited delay will be progressively accumulated at each hop resulting in an overall delay proportional to the number of hops. Indeed, we found that turning down of suppression mechanism is not mainly caused by the absence of listen-only period especially in the subsequent intervals. Instead, this problem mainly occurs because the node ignores all the received control messages from the randomly selected time in the previous interval to the end of that interval [32].

Algorithm 1 : Drizzle Algorithm

```
1:  procedure Initialization
2:      I ← $I_{min}$ , ck ← k
3:      s ← 0, c ← 0
4:      n ← 1, rFlag ← 1
5:  end procedure
6:  Procedure New Interval
7:      Start t_Timer as in
            $[\, s * \frac{I}{n}, (s+1) * \frac{I}{n} \,]$
8:      if ConsistentTransmissionReceived then
9:          c ← c + 1
10:     end if
11:     if InconsistencyDetected then
12:         I ← $I_{min}$ , c ← 0
13:         n ← 1, s ← 0
14:         if InitDODAG , JoinDODAG , or GRepair then
15:             rFlag ← 1
16:         else
17:             rFlag ← 0
18:         end if
19:     end if
20: end procedure
21: Procedure t_Timer Expired
22:     if c < ck then
23:         Transmit ScheduledMessage
24:         s ← s + 1
25:         ck ← ck - 1
26:         if ck < 0 then
27:             ck = 0
28:         end if
29:     else
30:         ck ← ck +1
31:         if c > ck then
32:             ck ← k
33:         end if
34:     end if
35:     c = 0
36: end procedure
37: procedure Interval Expired
38:     if rFlag = 1 then
39:         I ← 2 * I
40:         if I > $I_{max}$ then
41:             I ← $I_{max}$
42:         end if
43:     else
44:         I ← $I_{max}$
45:     end if
46:     n ← n +1
47: end procedure
```

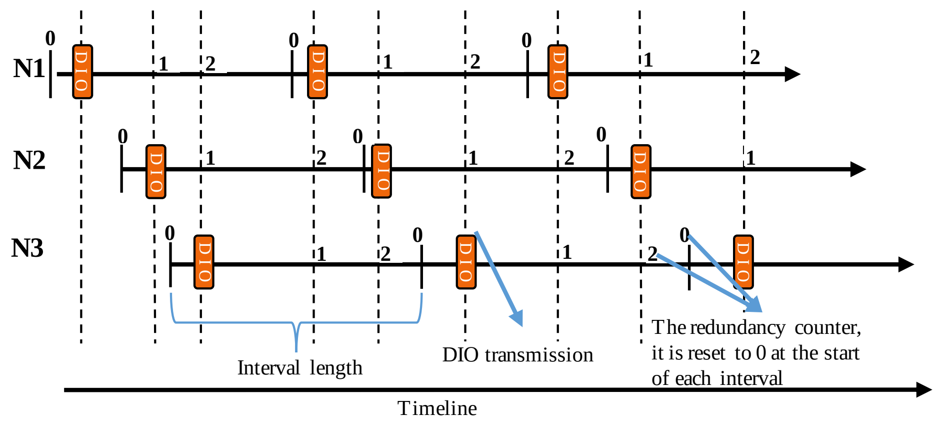
N1
N2
N3
DIO
0
1
2
Interval length
DIO transmission
The redundancy counter, it is reset to 0 at the start of each interval
Timeline

Fig. 1. Trickle short-listen problem in three asynchronous nodes; no suppressed transmissions at the absence of listen-only period

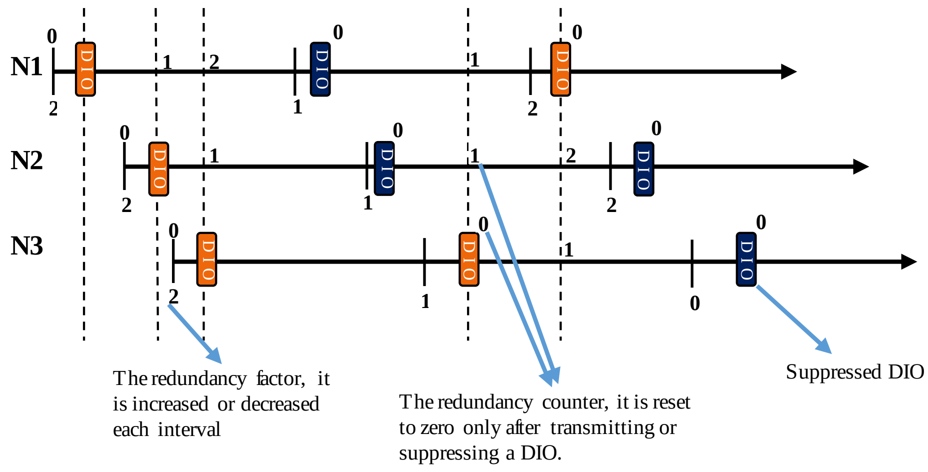
N1
N2
N3
DIO
0
1
2
The redundancy factor, it is increased or decreased each interval
The redundancy counter, it is reset to zero only after transmitting or suppressing a DIO.
Suppressed DIO

Fig. 2. Drizzle operations in three asynchronous nodes; reinforce suppression mechanism even at the absence of listen-only period

In Drizzle, the listen-only period is removed in order to facilitate faster propagation of the new information as each node would schedule its transmission from the range $\left[s * \frac{I}{n}, (s+1) * \frac{I}{n}\right]$ rather than [I/2, I.]. In order to mitigate the effect of the short-listen problem, Drizzle maintains track of all the received messages until the next scheduled time slot rather than the beginning of the next interval. Hence, instead of resetting the redundancy coefficient at the beginning of each interval, Drizzle resets it only at the beginning of the minimum interval and at the randomly selected time, *t*. The operations of this behavior is illustrated in Fig. 2. In Fig.2, you can observe that the three nodes have started their first interval at different times (i.e. they are not synchronized). Hence, as all nodes randomly selected transmission slots at the beginning of each interval, N2 and N3 suffers from the short-listen problem and fail to suppress their transmissions. However, this problem disappeared form the second interval and subsequent intervals. Looking again at Fig. 2, you can observe that N1 did not reset its redundancy counter, *c*, at the end of the first interval, instead, N1 waited until after its scheduled transmission slot to reset that counter. Thus, N1 has suppressed its transmission in the second interval, as the value of the redundancy counter is still greater than the redundancy coefficient, *k*, at the time of taking the transmission decision. This is not possible with Trickle as at the time of taking the transmission decision, the redundancy counter would have been reset to zero. Thanks to these new policies, Drizzle is able to resolve inconsistencies and propagate the new information much faster than other algorithms without even suffering from the short-listen problem, except the first interval, endorsing its energy-efficiency and scalability.

B. *Load-Balancing*

The distribution of the overhead evenly among nodes is one of the primary goals of any routing primitive primarily for the sake of avoiding disconnected regions in the network, which may lead to some kind of service disruption. In fact, the uneven-load distribution among nodes may lead to have some nodes drain their power faster than other nodes and consequently shortening their lifetime. For instance, 100 messages evenly disseminated by 100 nodes, does not incur a high cost. However, 100 messages disseminated only by one node does incur high cost [8] and might lead to an earlier death of this over-burdened node. This may have a serious impact on the connectivity of the network as whole especially if the nodes, which drain their power faster, are those representing the only-route to the base station (bottleneck nodes). The death of a bottleneck node means disconnecting that part of the network that forwards its data through that node which affects, in turn, the reliability of running applications and even denying some of the network services. In this regard, Drizzle introduces two mechanisms in order to guarantee efficient load-distribution among network nodes. First, on the interval-level, a node is given a broadcast transmission probability according to how many transmissions it has sent. In other words, the higher the number of broadcasted transmissions, the lower the probability that a node would transmit in the current interval. This is has been realized by introducing the parameters *n* and *s* that will allow nodes who transmitted less messages to select an earlier *t* for the current interval so to have more priority to transmit. For example, if the length of the current interval, *I* , is 100 s, assuming that the current interval is the 4th interval, and assuming that three nodes A, B, and C have 0, 1, and 2 transmissions respectively in the three previous intervals, (i.e. A has never transmitted any DIO during the three intervals, B has only transmitted once, and C has transmitted two DIOs). According to our algorithm the three nodes should select their transmission slots, *t*, according to the equation $[s*I/n,(s+1)*I/n]$ as follows:

$A_t = [0 * 100/4, 1 * 100/4] = [0, 25]$.
$B_t = [1 * 100/4, 2 * 100/4] = [25, 50]$.
$C_t = [2 * 100/4, 3 * 100/4] = [50, 75]$.

You can observe from the above ranges that A will have a better chance to transmit in the current interval (i.e. 4th interval) by selecting *t* from the range [0, 25].

Second Drizzle allows each node to have its own value for the Suppression Coefficient, *k*, referred to as *ck*. Each node changes the value of its initial, *k*, autonomously according to how many transmissions have been suppressed or sent during the previous intervals. This is different from that of the standard Trickle algorithm where a node is given the same broadcast probability every interval, even though, it might never have had a chance to transmit. The unequal broadcast probability gives the opportunity for each node to broadcast its routing information as soon as possible enabling more efficient discovering of all possible paths and, distribute load evenly among respective nodes.

C. Simulation Experiments

In this subsection, we compare the proposed scheme with the standardized Trickle algorithm as well as three Trickle variances in the literature namely, opt-Trickle [22], Trickle-F [10], and the adaptive-k (Trickle-Ad) [23] in terms of control-plane overhead, convergence time, power consumption and Packet Delivery Ratio (PDR). The compared algorithms have been implemented in Contiki, a lightweight and open-source operating system designed specifically for the low-power resource-constrained IoT devices[33]. Contiki features a highly optimized networking stack including several IoT standards such as CoAP, UDP, 6LoWPAN and IPv6. It also features implementations for the RPL standard fundamental mechanisms including the routing maintenance mechanism (Trickle) within a library called ContikiRPL[34], which was used as a ground for our implementation. We used Cooja, java-based cross-level simulator for the Contiki operating system, to carry out the simulation experiments. One advantage of using Cooja with Contiki is that it allows us to emulate the exact binary code that run on a real mote hardware[35]. Cooja incorporates an internal hardware emulator called MSPsim [36], which is used in our simulations to emulate accurately (i.e. impose hardware constraints) the Tmote Sky platform, an MSP430-based board with an ultra-low power IEEE 802.15.4 compliant CC2420 radio chip. The Unit Disk Graph Radio Medium (UDGM) with different loss rates was used in order to simulate the radio propagation in lossless and lossy networks. At the MAC layer, we used The CSMA/CA protocol while the ContikiMac was used at the radio duty cycling (RDC) layer. The Minimum Rank with Hysteresis Objective Function (MRHOF) with ETX metric is selected for calculating the ranks of nodes and building the DODAG due to its efficiency characterizing the quality of links. At the application layer, we simulate a periodic data collection application where each node send to the sink one packet every 60 seconds (the time of sending is randomly chosen within the 60 seconds period). We have considered in our simulations uniform and random topologies where nodes are spread in a square area of 200 x 200m dimensions. The border router (sink) is placed in the middle of the network. For each scenario, ten simulation experiments with different seeds are run in order to get statistically solid results. The graphs show the average (mean) values of the results and the error bars at the 95% confidence interval of the mean. The simulation time is selected to be 20 virtual minutes for each experiment. For brevity, other simulation parameters are provided in Table II.

TABLE II. SIMULATION PARAMETERS

Parameter Name	Values
Number of nodes	100
Redundancy Factor (*k*)	1,3,5,7,10
I_{min} (ms) / I_{max} (ms)	$2^{10}/2^{20}$
Simulation time	20 minutes
Data Packet Rate	60 s
Mac/Adaptation Layer	ContikiMac/6LoWPAN
Radio Medium	Unit Disk Graph Medium (UDGM)
Loss model	Distance loss
Loss Ratio	0,10,30,50
Range	30 m
Interference Range	35 m

In the first set of experiments, we compare the five algorithms in lossy networks under the distance loss model varying the physical link loss rate between 0% and 50%. The 0% loss rate means that the network is lossless and as result does not experience any loss due to signal fading. However, the loss may still occur due to other factors such as hidden terminals and collisions. Figs. 3, 4, and 5 show the compared protocols performance in terms of control-plane overhead, which is defined here as the number of control messages, power consumption, and the PDR respectively.

As can be observed from Fig. 3, the compared algorithms increase their control traffic overhead as the loss rate increases. However, Trickle' variances suffer heavily in terms of scalability in comparison with Drizzle especially when the network is characterized by higher loss rates. In the worst-case scenario (50% loss rate), Drizzle registers an overhead rate of approximately seven times less than that of Trickle while it registers also an overhead of approximately three times less than that of Trickle-adaptive. In fact, Trickle-adaptive uses a density-based mechanism to control the value of the redundancy factor. Although Trickle-adaptive has managed to reduce the control-plane overhead compared to other Trickle variances, it is not as efficient as Drizzle. Trickle-adaptive uses the number of DIO messages received by a specific node to estimate indirectly the network density at that node. Although this method might give approximately accurate estimation for the node degree when the network is characterized by synchronized intervals among its nodes, it may suffer from inaccurate estimation in asynchronized networks. For instance, in asynchronized network, the frequency of transmission may differ significantly from a node currently in its minimal interval to another node currently in its maximum interval. Hence, the node in its minimum interval would transmit more frequently giving the receiver node an impression that it has more neighbors than it actually has affecting negatively the accuracy of the network density estimation at that node. On the other hand, the superiority of Drizzle can be attributed to its adaptive suppression mechanism that allows a node to decrease autonomously its own transmission probability in the current interval according to how many control messages it has sent previously. In other words, the higher the control messages a node has sent, the lower its probability to transmit in the current interval and, therefore, bringing down the number of redundant control messages. Another reason behind the lower control-plane overhead of Drizzle is that it does not gradually double the current interval each time it receives inconsistent control message. In several cases, according to the value of the *rFlag*, Drizzle moves directly, and not gradually, to the lowest transmission rate skipping the intermediate intervals and by that suppressing many redundant transmissions.

The decline in the number of transmitted control messages has resulted in lower power consumption of Drizzle in comparison with other algorithms as depicted in Fig. 4. However, it is not with the same rate of that of control-plane overhead. This is because the main factor contributing to energy consumption is the underlying layers' algorithms and not the number of control packets. With respect to packet delivery ratio, Drizzle slightly performs better than all Trickle variances as shown in Fig. 5. However, it is very important to point that this PDR rate of Trickle's variances is obtained through generating more control packets than that of Drizzle and consuming more power. This indicates that Drizzle is able to discover optimal paths slightly more efficient than any Trickle variance, however, with much less control messages. Fig. 6 compares the algorithms in terms of convergence time. The convergence time here refers to the time at which the node has joined the network. Hence, the average convergence time is the convergence time of all nodes divided by the number of the nodes in the network. This is different from the works in [22] [29], where they define the convergence time as the time at which the last node has joined the network.

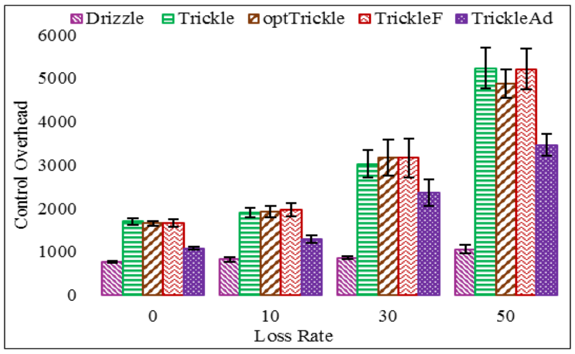

Fig. 3. Control overhead under different loss rates (uniform)

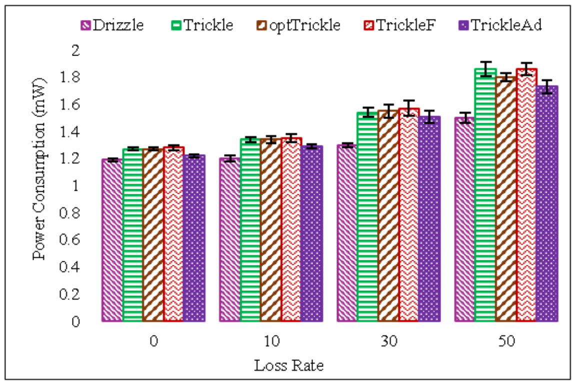

Fig. 4. Average power consumption with various loss rates (uniform)

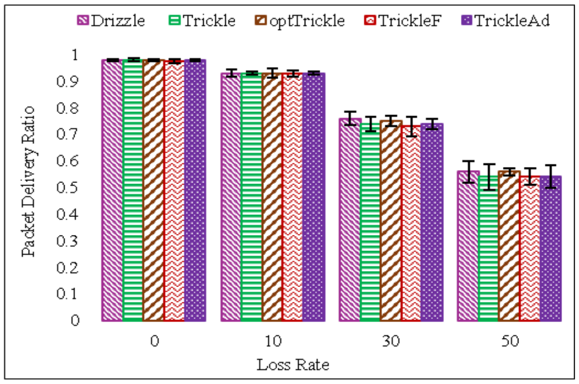

Fig. 5. PDR under different loss rates (uniform)

As can be observed from in Fig. 6, a network running Drizzle has the fastest convergence time compared to Trickle, Trickle-F and Trickle-adaptive even when the network suffers from higher loss rates. The case is somewhat different when considering opt-Trickle. Drizzle slightly outperforms opt-Trickle in terms of convergence time. The superiority of Drizzle in terms convergence time stems mainly from eliminating the listen-only period that allows the node to schedule its transmission as early as possible without even experiencing short-listen problem. The slight degradation of opt-Trickle in lossy networks stems from the fact that it only permits removing the listen-only from the first interval. In a lossless medium this might not be a problematic as the probability of DIOs being lost in the first interval is very small. Thus, having listen-only period in the other intervals would have no effect on the convergence time. Conversely, the probability of DIO loss increases in lossy medium. Hence, a DIO message, not delivered in the first interval, would have to go through a listen-only period in the subsequent intervals probably delaying the joining of other nodes. The fact that Drizzle does not experience the short-listen problem can be confirmed by observing that Drizzle achieved faster convergence time, however, with generating much less control messages as illustrated in Fig. 3. It could be also observed from the results that the higher the value of loss rate, the slower the convergence time in all algorithms. This is could be explained by the fact that the higher the loss rate, the higher the probability that the control packet would be lost delaying the joining process until the next successfully received packet.

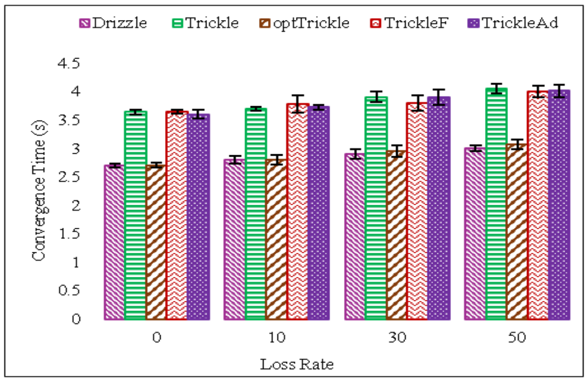

Fig. 6 Average convergence time under various loss rates (uniform)

Figs. 7, 8, 9, and 10 presents a comparison among the five algorithms in a random topology with various loss rates in terms of control overhead, power consumption, PDR and convergence time respectively. Similarly, the results illustrate that Drizzle has the least amount of overhead under the various loss rates (e.g. drizzle has an overhead of approximately four times less than Trickle under loss rate of 50%). Drizzle has also the least amount of power consumption, and along with opt-Trickle, it has the fastest convergence time while featuring relatively a higher packet delivery ratio.

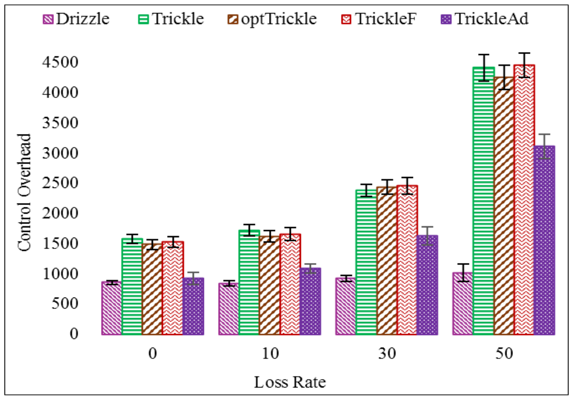

Fig. 7 Control overhead under different loss rates (random)

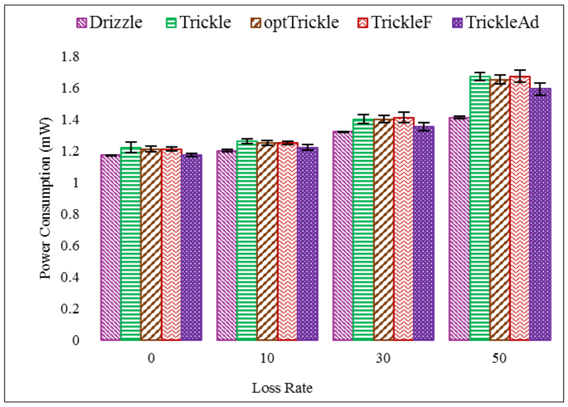

Fig. 8 Average power consumption with various loss rates (random)

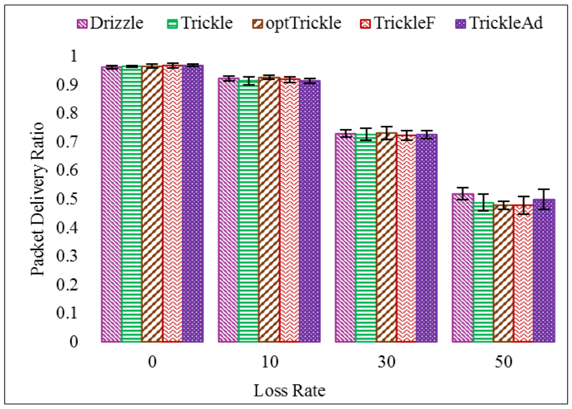

Fig. 9. PDR under different loss rates (random)

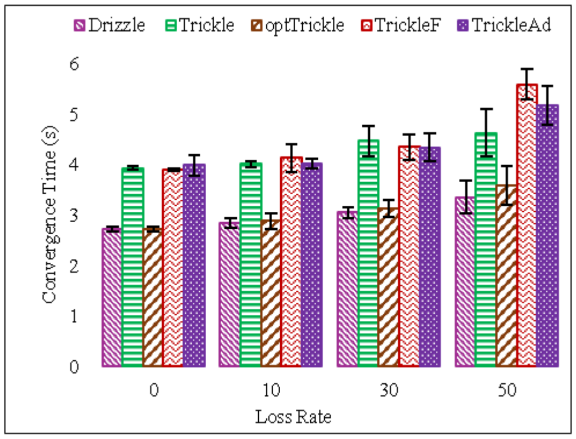

Fig. 10 Average convergence time under various loss rates (random)

In the second set of experiments, we have evaluated the impact of the redundancy coefficient on network performance in two variants of LLNs (lossless and lossy with 50% loss rate) in both random and uniform distributions (showing only the results of uniform distributions as the random has similar results). As observed from Figs 11 and 12, it is clear that increasing the redundancy factor result in higher traffic overhead for Drizzle, Trickle, opt-Trickle and Trickle-F and in both kinds of networks (lossy and lossless). A noticeable point here is the behavior of TrickleAd under varies redundancy values. It seems that there is no correlation between the initial value of the redundancy factor and control plane overhead. This is interpreted by the fact that the value of k is dynamically changed based on the node degree so whatever is the initial value; it will be decreased or increased to the extent that reflects the network density at that node. However, Drizzle still shows the best results in terms of traffic overhead in comparison with Trickle's variances including Trickle-Ad under different values of k cases. The positive correlation between the value of k and traffic overhead in the compared algorithms (except TrickleAd) can be explained easily by the fact that the nodes tend to suppress less messages as the k increases. On the other side, the superiority of Drizzle in terms of traffic overhead again can be attributed to the adaptivity of Drizzle's suppression mechanism, which allows the nodes to change dynamically their suppression coefficient according to their transmission history. Regardless of the initial value of the redundancy coefficient, a node running Drizzle is able to decrease its version each time it sends a message reducing its priority to transmit in the next interval, thus, bringing down the number of unnecessary transmissions.

A key noticeable point here is the network performance in terms of Packet Delivery Ratio. While all compared algorithms achieve approximately similar results in the lossless scenario whatever is the value of k as illustrated in Fig. 13, the case is somewhat different when the network is experiencing losses and low redundancy factor values. Fig. 14 shows that Drizzle improves the PDR especially with lower values of k by up to 10% compared to other algorithm. The slightly better performance of Drizzle in terms of PDR in lossy networks indicates the capacity of Drizzle in discovering more optimal routes with much less traffic overhead. The main reason behind this efficiency lies in the way Drizzle distributes the transmission of control messages through the network. Drizzle's adaptive suppression mechanism, in addition to its slotting mechanism, ensures the fairness in the distribution of transmitted control messages. The fair distribution among nodes guarantees the optimal routes discovery for all the nodes and, thus, improving the packet delivery ratio. It is also clear from Fig. 15 and Fig. 16 that the superiority of Drizzle over Trickle's variances in terms of PDR has been achieved under low power consumption rates in both networks types (i.e. lossy and lossless) regardless of the value of the redundancy factor. This is also can be attributed to the capacity of Drizzle to minimize the overhead and discovering the optimal routes affecting positively the power consumption. Pertaining to convergence time, Drizzle also converges faster than Trickle's variances under different values of k, and whether the network is lossless or lossy as illustrated in Fig. 17 and Fig. 18 respectively. This also is attributed to the facts explained previously regarding removing the listen-only period which contributes into enhancing the convergence time.

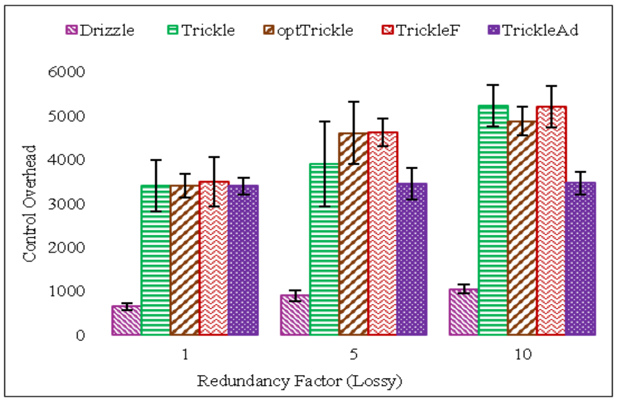

Fig. 11. Control overhead under various k (lossy)

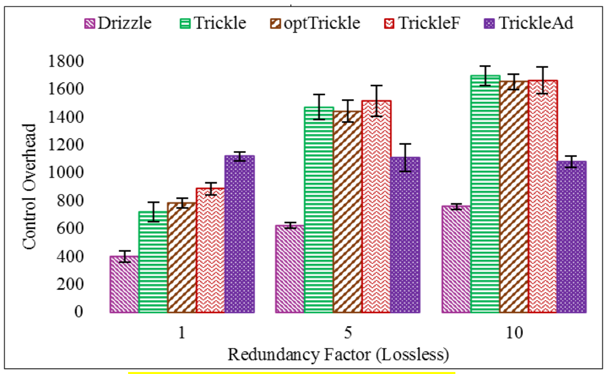

Fig. 12.Control overhead with various k (lossless)

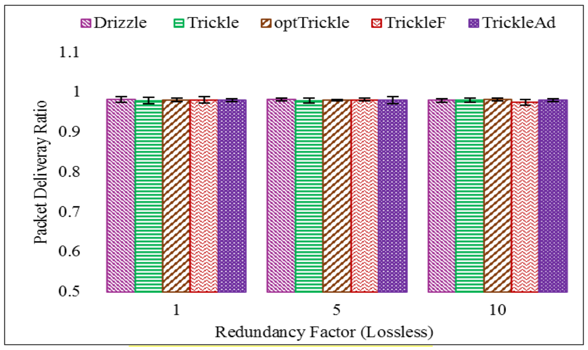

Fig. 13. . PDR with various values of k (lossless)

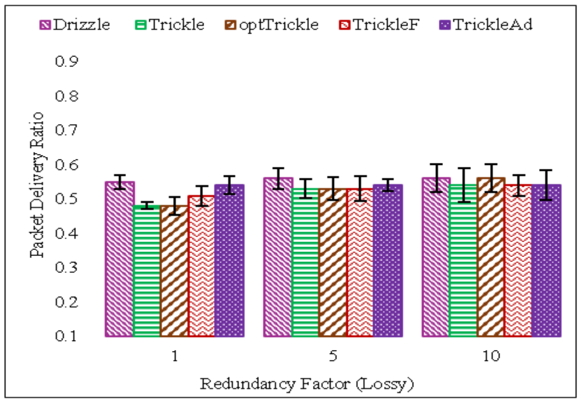

Fig. 14. PDR with various values of k (lossy)

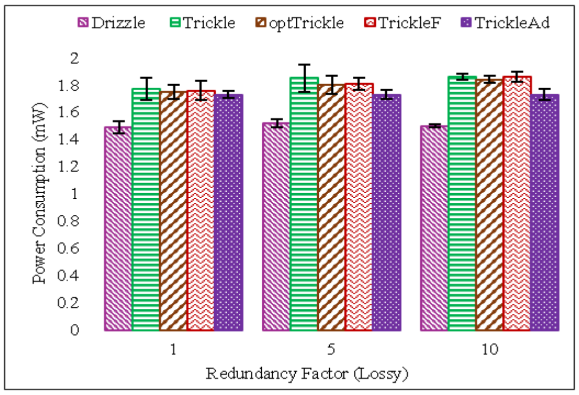

Fig. 15. Average power consumption with various *k* (lossy)

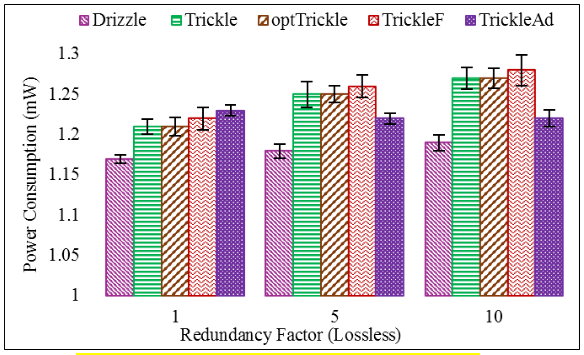

Fig. 16. Average power consumption with various *k* (lossless)

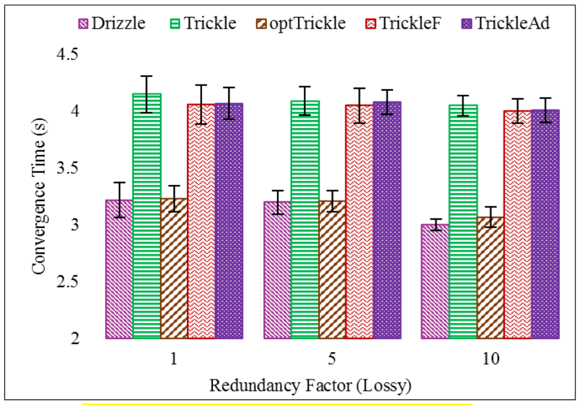

Fig. 17. Convergence time under various values of *k* (lossy)

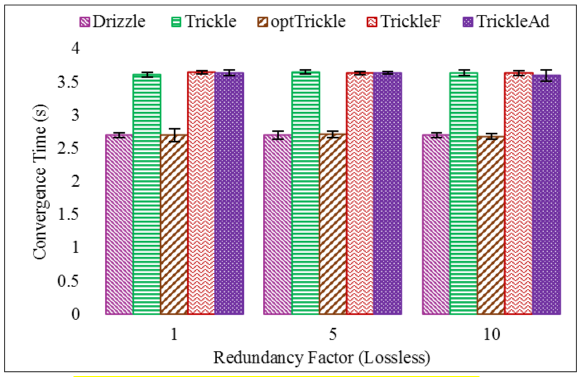

Fig. 18. Convergence time under various values of *k* (lossless)

V. CONCLUSION AND FUTURE WORK

In this study, a new routing primitive for route maintenance called Drizzle algorithm has been proposed for LLNs. Drizzle relies on the transmission history of nodes to configure their suppression mechanism. In addition, Drizzle introduces a new policy for mitigating the negative effect of so-called short-listen problem with the goal to limit transmission redundancy while providing faster convergence time and further boost the fairness in the network. A performance evaluation of the proposed algorithm in comparison with the state-of-the-art routing maintenance algorithms has been conducted. The results highlighted the efficiency of Drizzle algorithm. In addition, we demonstrated how Drizzle exhibits better load distribution and scalability in comparison with the standard IETF Trickle algorithm and its variances. Another direction for future work is to validate the efficiency of the proposed algorithm in real testbeds, and networks with different densities under a wide range of operating conditions.

REFERENCES

[1] J. W. Hui and D. E. Culler, "Extending IP to low-power, wireless personal area networks," Internet Computing, IEEE, vol. 12, no. 4, pp. 37–45, 2008.

[2] T. Winter, P. Thubert, A. Brandt, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik and J. Vasseur, "RPL : IPv6 Routing Protocol for Low power and Lossy Networks", RFC 6550, IETF ROLL WG, 2012.

[3] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in Proceedings of the 7th ACM conference on embedded networked sensor systems, 2009, pp. 1–14.

[4] S. Dawson-Haggerty, A. Tavakoli, and D. Culler, "Hydro: A hybrid routing protocol for low-power and lossy networks," in Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on, 2010, pp. 268–273.

[5] T. Clausen, A. C. de Verdiere, J. Yi, A. Niktash, Y. Igarashi, and U. Herberg, "The Lightweight On-demand Ad hoc Distance-vector Routing Protocol - Next Generation (LOADng)," IETF, Draft, Oct 2012.

[6] L. Pradittasnee, Y.-C. Tian, and D. Jayalath, "Efficient route update and maintenance processes for multipath routing in large-scale industrial wireless sensor networks," in Telecommunication Networks and Applications Conference (ATNAC), 2012 Australasian, 2012, pp. 1–6.

[7] H. Jiang and J. Garcia-Luna-Aceves, "Performance comparison of three routing protocols for ad hoc networks," in Computer Communications and Networks, 2001. Proc.. 10th Int. Conference on, 2001, pp. 547 –554.

[8] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in First USENIX/ ACM Symposium Networked Systems Design Implementation (NSDI 2004), San Francisco, CA, Mar. 2004.

[9] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, "The Trickle algorithm," RFC 6206, Internet Engineering Task Force (IETF), 2011.

[10] C. Vallati and E. Mingozzi, "Trickle-F: Fair broadcast suppression to improve energy-efficient route formation with the RPL routing protocol," in Sustainable Internet and ICT for Sustainability (SustainIT), 2013, pp. 1–9.

[11] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, Mobile Computing, vol 353, pp. 153–181. Kluwer Academic Publishers, 1996.

[12] D. A. Maltz and D. B. Johnson and Y. Hu. "The dynamic source routing protoco (DSR) for mobile ad hoc networks for IPv4", IETF RFC 4728, Feb 2007.

[13] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," IETF, RFC 3561, July 2003.

[14] T. Clausen (ed) and P. Jacquet (ed), "Optimized link state routing protocol (OLSR), "IETF RFC 3626, Experimental, October 2003,.

[15] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum and L. Viennot, "Optimized link state routing protocol for ad hoc networks,"

Proceedings. IEEE International Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century., 2001, pp. 62-68.

[16] J. Moy, “Open Shortest Path First Routing Protocol (OSPF version 2),” RFC 2328, Standards Track, April 1998..

[17] https://datatracker.ietf.org/wg/roll/charter/

[18] I. Chakeres and C. Perkins, “Dynamic MANET On-Demand (DYMO) Routing,” IETF Internet-Draft, draft-ietf-manet-aodvv2-16, work in progress , May. 2016.

[19] T. Clausen, C. Dearlove, P. Jacquet, and U. Herberg, “The Optimized Link State Routing Protocol version 2,” Internet Draft, draft-ietf-manetolsrv2- 19, work in progress,March 2013.

[20] J. Yi and T. Clausen, “Collection Tree Extension of Reactive Routing Protocol for Low-Power and Lossy Networks,” International Journal of Distributed Sensor Networks, vol. 2014, pp. 1–12, 2014.

[21] J. Chroboczek, “The Babel Routing Protocol”, RFC 6126, April 2011.

[22] B. Djamaa and M. Richardson, “Optimizing the Trickle Algorithm,” IEEE Communications Letters, vol. 19, no. 5, pp. 819–822, May 2015.

[23] T. M. M. Meyfroyt, M. Stolikj and J. J. Lukkien, "Adaptive broadcast suppression for Trickle-based protocols," The 16th IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), Boston, MA, , 2015, pp. 1-9.

[24] O. Gnawali and P. Levis, “Recommendations for Efficient Implementation of RPL,” Internet Draft, draft-gnawali-roll-rpl-recommendations-05,Mar. 2013.

[25] J. Hui and R. Kelsey, “Multicast Protocol for Low-Power and Lossy Networks (MPL),” RFC 7731, Feb. 2016.

[26] T. Coladon, M. Vučinić and B. Tourancheau, "Multiple redundancy constants with trickle," The 26th IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Hong Kong, , 2015, pp. 1951-1956.

[27] M. Becker, K. Kuladinithi, and C. Görg, “Modelling and simulating the Trickle algorithm,” in Mobile networks and management, Springer, pp. 135–144,2012.

[28] H. Kermajani, C. Gomez, and M. H. Arshad, “Modeling the Message Count of the Trickle Algorithm in a Steady-State, Static Wireless Sensor Network,” IEEE Communications Letters, vol. 16, no. 12, pp. 1960–1963, Dec. 2012.

[29] T. M. M. Meyfroyt, “Modeling and analyzing the Trickle algorithm,” Master’s Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2013.

[30] Y.-W. Lin and P.-H. Wang, “Performance Study of an Adaptive Trickle Scheme for Wireless Sensor Networks,” in Ubiquitous Computing Application and Wireless Sensor, vol. 331, J. J. Park, Y. Pan, H.-C. Chao, and G. Yi, Eds. Dordrecht: Springer Netherlands, pp. 163–173, 2015.

[31] B. Ghaleb, A. Al-Dubai, I. Romdha, Y. Nasser and A. Boukerche, "Drizzle: Adaptive and fair route maintenance algorithm for Low-power and Lossy Networks in IoT," 2017 IEEE International Conference on Communications (ICC), Paris, 2017, pp. 1-6.

[32] B. Ghaleb, A. Al-Dubai, E. Ekonomou, “E-Trickle: Enhanced Trickle Algorithm for Low- Power and Lossy Networks”, In: Proceedings of the 14th IEEE International Conference on Ubiquitous Computing and Communications (IUCC). Liverpool, UK: IEEE Communication Society, October 2015.

[33] Dunkels, B. Gronvall and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," 29th Annual IEEE International Conference on Local Computer Networks, 2004, pp. 455-462.

[34] Contik O.S and cooja simulator” http://www.contiki-os.org/.

[35] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne and T. Voigt, "Cross-Level Sensor Network Simulation with COOJA," Proceedings. 2006 31st IEEE Conference on Local Computer Networks, Tampa, FL, 2006, pp. 641-648

[36] J. Eriksson, A. Dunkels, N. Finne, F. ¨ Osterlind, and T. Voigt, “Mspsim an extensible simulator for msp430-equipped sensor boards,” in Proceedings of the European Conference on Wireless Sensor Networks (EWSN ’07), Delft, The Netherlands, 2007.