Optimizing the Trickle Algorithm

Badis Djamaa and Mark Richardson

***Abstract*—The Trickle Algorithm has enjoyed much popularity and widespread use as a basic network primitive ensuring low-cost data consistency in lossy networks. Trickle is shaped by the so-called short-listen problem, hence the imposition of a listen-only period. Such a period allows Trickle to robustly address the short-listen problem at the expense of increased latency. In this letter, we introduce a simple yet powerful optimization to Trickle that can dramatically decrease Trickle's latency with virtually no additional overhead to its scalability and robustness. Extensive simulation and testbed experiments are reported here, yielding greater than a factor of 10 decrease in propagation time.**

***Index Terms*—The Trickle algorithm, Low-power and lossy networks, RPL, MPL.**

I. Introduction

THE Trickle algorithm, first introduced in [1] and later standardized as RFC 6206 in [2], has gained much popularity and emerged as a basic network primitive that can ensure fast and reliable resolution of data inconsistencies with low maintenance cost while scaling well with network density. For these attractive features, Trickle makes the basis of many internet standards [2][3], and it is deployed in many other applications such as reliable broadcast/dissemination and distributed service/resource discovery.

The rationale behind Trickle is to provide constrained networked nodes with a local, robust, energy-efficient, simple, and scalable information exchange primitive. To this end, Trickle deploys two basic techniques: adaptive transmission periods and a timer-based suppression mechanism. Dynamically adjusting transmission periods to the network context allows Trickle to achieve quick resolution of inconsistencies, while only generating few consistency control packets when the network is in a steady state. On the other hand, a simple timer-based suppression mechanism allows Trickle's communication cost to scale logarithmically with network density. In addition, Trickle implementations only require very few states in terms of memory and computational resources.

The final shape of the Trickle algorithm, described below, is the fruit of years of experimenting and running Trickle across various application domains. Based on which, [2] advises not to try to tweak Trickle's behavior without great experimental work. Nevertheless, Trickle in its actual final format suffers from many problems, the principal one being increased propagation latency. This motivated us to carry out the present work in order to propose a simple, yet powerful optimization to Trickle.

This paragraph of the first footnote will contain the date on which you submitted your paper for review.

Badis Djamaa and Mark Richardson are with the Center for Electronic Warfare, Cranfield University, Defence Academy of the United Kingdom, Shrivenham SN6 8LA. Email: {b.djamaa, m.a.richardson}@cranfield.ac.uk

II. The Trickle Algorithm

A node using the Trickle algorithm periodically broadcasts its data unless it has recently heard identical ones (referred to by the generic term *consistent* in what follows). As long as nodes agree on what data they have, Trickle exponentially increases the transmission window. When data disagreements (*inconsistencies*) are detected, Trickle starts transmitting more quickly. To realize this behavior, and as by [2]'s notation, the Trickle algorithm maintains three variables; namely:

- A consistency counter c.
- A Trickle interval I.
- A transmission time t within an interval I.

In addition, Trickle defines three configuration parameters:

- The minimum interval size $Imin$ ($Imin$ is defined in units of time, e.g., milliseconds, seconds).
- The maximum interval size $Imax$ ($Imax$ is described as a number of doublings of $Imin$ and hence the time specified by $Imax$ would be $Imin \times 2^{Imax}$).
- The redundancy constant k (k integer greater than zero).

Trickle can be expressed by the six rules (steps) below:

- ***Step 1***: When Trickle starts execution, it picks I uniformly at random from $[Imin; Imin \times 2^{Imax}]$ and begins the first interval.
- ***Step 2***: When an interval I begins, Trickle resets c to 0 and picks t uniformly at random from the range $[I/2; I)$.
- ***Step 3***: Whenever a node hears a consistent transmission, Trickle increments the consistency counter c.
- ***Step 4***: At time t, Trickle transmits if and only if c is less than k ($c < k$). Otherwise, the transmission is suppressed.
- ***Step 5***: When the interval I expires, Trickle doubles the interval length up to the time specified by $Imax$. Trickle then starts a new interval as in ***Step 2***.
- ***Step 6***: If Trickle hears an inconsistent transmission while I is greater than $Imin$, it resets the Trickle timer. To do so, Trickle sets I to $Imin$ and starts a new interval as in ***Step 2***. Otherwise, i.e. I was equal to $Imin$ when detecting the inconsistency, Trickle does nothing. Note that the Trickle timer can also be reset in response to external events.

Choosing t from the second half of an interval in ***Step 2*** ensures a half-interval listen-only period, which is introduced in response to the challenging short-listen problem discussed in [1]. However, the listen-only period trades off transmission cost for increased delays accumulated at every hop.

III. The New Trickle Algorithm

In this section, we introduce the proposed optimization to Trickle with a description of its rationale and main benefits.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCOMM.2015.2408339, IEEE Communications Letters

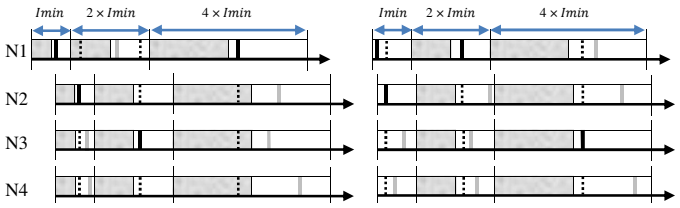

Fig. 1. Trickle (left) and New-Trickle (right). The gray rectangle represents the listen-only period. Black lines are transmissions, gray ones are suppressed transmissions and dotted lines are receptions.

A. The New Trickle Algorithm

Replace Step 2 in the Trickle description above as follows:

- ***Step 2***: When an interval I begins, Trickle resets c to 0 and picks t uniformly at random from the range:
  - $[0; Imin)$, if the interval began as a result of ***Step 6*** (because of an *inconsistency* or external *events*).
  - $[I/2; I)$, otherwise (the interval began as a result of ***Step 1*** or ***Step 5***).

In a nutshell, the proposed optimization says: if a new interval is started as result from resetting I to $Imin$ (***Step 6***) then choose the transmission time t from $[0; Imin)$ instead of $[Imin/2; Imin)$. The rest remains unchanged.

B. Rationale

Choosing t from $[0; Imin)$ if a Trickle interval begins as a result of ***Step 6*** allows Trickle to resolve inconsistencies much faster, while does not introduce noticeable extra cost. This is so thanks to ***Step 6*** in the Trickle algorithm, which enables the nodes hearing an inconsistency to immediately shrink their intervals to $Imin$. This fact can constitute an implicit synchronization between such nodes in this specific interval and allow them to choose t from $[0; Imin)$ without experiencing a short-listen problem with each other as shown in Fig .1. Note, however, that whichever receiver transmits (e.g. N2, N3 or N4 in Fig .1), it might experience a short-listen with the originator (N1). The impact of this does not affect Trickle's scalability, as will be discussed in section V.

Although neighbors can experience non-synchronized $Imin$ intervals as a result of losses and/or the multi-hop nature, an implicit synchronization in the transmission periods of these intervals remains valid, as will be shown from the obtained results and be detailed in a future work. Note that there is no guarantee of implicit synchronization in the following intervals, and hence the listen-only period is deployed.

C. Main Advantages

Since Trickle resolves inconsistencies in $Imin$-sized intervals, the above optimization is expected to drastically decrease Trickle's propagation time at virtually no extra cost. At first glance, one can think of the propagation time to be halved. However, many parameters (e.g. $Imin$ value) can impact the propagation time, allowing it to be much faster as will be discussed in section V. Note that other promising benefits were also observed and are planned as future work.

TABLE I
EXPERIMENTAL PARAMETERS

Parameter	Value
Simulation radio medium	Unit Disk Graph Medium (UDGM)
$Imin/Imax$	From 62 ms to 2 seconds / 3 ($Imin \times 2^3$)
k	1, 2, 3, 5, 7, 9
#nodes	16, 36, 64, 100, 196, 400
Transmission power levels	11, 15, 18, 23, 31
Adaptation layer/MAC	6LoWPAN/CSMA with 100% duty cycle
MAC Retransmissions	0 (Trickle ensures such a task)

IV. Methodology and Experimental Design

We used the Contiki OS Trickle library as a basis for our modifications and evaluations in both cycle-accurate simulations and public testbed platforms. Simulation experiments give us controlled environments, while public testbed experiments validate the results in real-world deployments. To put the results into context, we compared the new Trickle algorithm (New-Trickle) with both Trickle and Short-Trickle: a version of Trickle that takes t from $[0; I)$.

We used a reference simulation scenario consisting of 400 nodes deployed in a grid topology. In single-hop experiments, all the nodes were within communication range of each other. In the multi-hop scenario, we opted for a dense network where nodes have around 36 neighbors. The network diameter was about 13 hops. The default value of $Imin$ was one second and that of k was one. Node 1 in the upper left corner generated a new packet that gets propagated and exchanged using one of the three protocols. Starting from this reference scenario, we varied physical link loss rate (loss probability is proportional to the square of the sender-receiver distance), network density, $Imin$ and k, and we measured the number of transmissions per interval together with the consistency time (time from issuing an update until all the nodes get updated). Each simulation runs on 10 virtual minutes and was repeated 25 times. We plot in the graphs the mean with its standard error. TABLE I lists the main parameters used/varied in our experiments.

V. Results and Discussions

In this section, we present and discuss the obtained simulation and Indriya's testbed results.

A. Multi-hop Networks

Fig. 2 presents obtained results in the multi-hop network. First row of graphs in Fig. 2 shows the performance in the reference scenario of the three evaluated protocols under losses. As can be seen from these graphs, New-Trickle approached the propagation time of Short-Trickle even in a worst case of 90% loss rate. With increasing success rates the two protocols registered similar propagation time, which is about four times less than that of Trickle. While Short-Trickle achieved this by generating more packets, New-Trickle sent approximately the same packets as Trickle (slightly bigger in lossy networks). The quantification and reasons behind this extra cost will be discussed in section V.C.

The second row of graphs shows the performance of the three evaluated protocols when varying $Imin$ in a physically

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCOMM.2015.2408339, IEEE Communications Letters

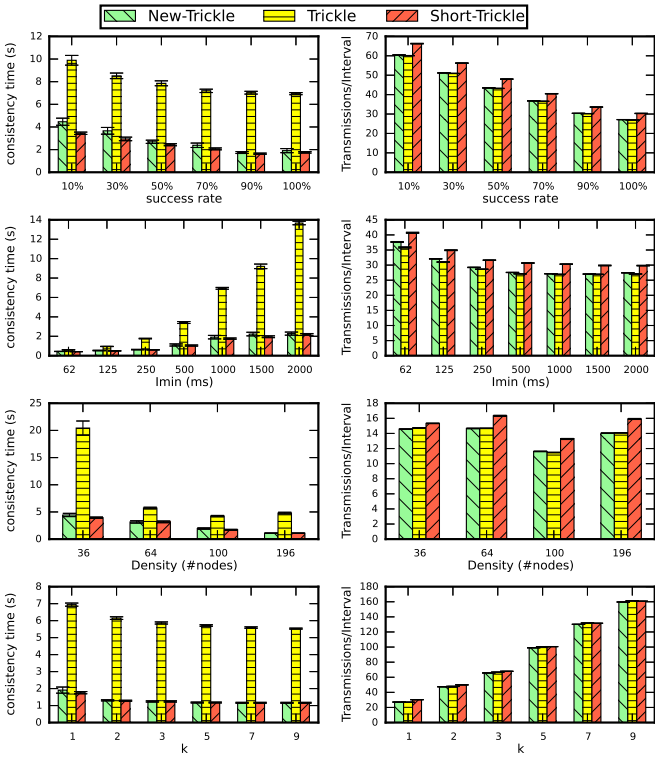

Fig. 2. Performance evaluation in multi-hop networks

lossless network. As expected the propagation time of New-Trickle approached that of Short-Trickle. Interestingly, unlike Trickle, the propagation time of New-Trickle does not heavily depend on $Imin$, making it propagate seven times faster in an $Imin$ of two seconds. This gap is expected to increase with increasing $Imin$ values. The cost of New-Trickle is similar to that of Trickle with a small increase caused mainly by losses. Thus, even the physical topology is lossless; losses can occur as a result of network dynamics, e.g. hidden terminals.

The third row of graphs in Fig. 2 presents the performance when varying network density. Similarly to previous graphs, New-Trickle generated as many transmissions as Trickle while propagating as fast as Short-Trickle, which is about four times faster than Trickle in a sparse 36-node network.

The fourth row of graphs in Fig. 2 shows the performance in the reference scenario when increasing k. Increasing k slightly decreased the propagation time, while considerably increased the cost of the three protocols. As expected, New-Trickle propagated about 3.5 times faster than Trickle while providing similar cost.

B. Single-hop Networks

The first row of graphs in Fig. 3 shows consistency times and transmission costs of the three evaluated protocols in a 400-node single-hop network when varying the loss rate. New-Trickle propagated about five times faster than Trickle with an approximately similar transmission cost (the small extra cost will be discussed in the following subsection). Note that in the particular case of 0% loss, Trickle achieved the same propagation, and even Short-Trickle transmitted less than

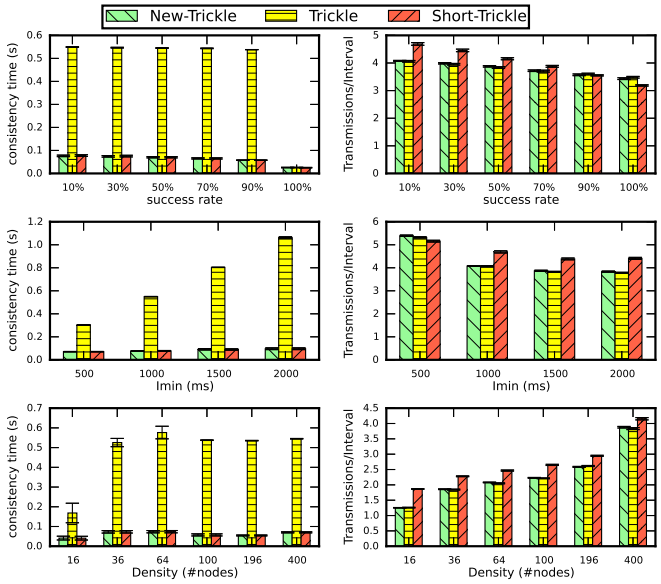

Fig. 3. Performance evaluation in single-hop networks

Trickle. This is because the network was synchronized as the inconsistent transmission was simultaneously received by all.

Second row of graphs presents the performance of the evaluated protocols when varying $Imin$ in a very lossy network (90% loss rate). As we can see from these graphs, New-Trickle propagation is loosely coupled with the value of $Imin$ allowing it to propagate about 11 times faster than Trickle in an $Imin$ of two seconds. This is achieved at approximately the same cost as Trickle. On the other hand, Short-Trickle's cost increased drastically with increasing losses.

The third row of graphs in Fig. 3 presents the performance of the three evaluated protocols in a network with 50% loss rate when varying the number of nodes. As can be seen from these graphs, New-Trickle propagated as fast as short-Trickle with only a very small extra cost when compared to Trickle, while propagating more than six times faster.

C. Causes of the Additional Cost

As observed previously, a small extra transmission cost is introduced by New-Trickle. Fig. 4 illustrates the reasons behind that and Fig. 5 quantifies it. Fig. 4 shows that, although the $Imin$ intervals' transmissions will not experience short-listen with each other, Trickle makes sure that such transmissions coincide with other intervals' listen-only periods (e.g. N1, N2 and N3 in Fig. 4), and hence might help in deleting their transmissions. New-Trickle, however; may not provide such a guarantee. This seems to only affect a few following intervals. To confirm this, we plot the average number of transmissions in the second, third and the rest of the intervals with increasing density in a network with 50% losses. As can be seen from Fig. 5, this small extra cost is degraded such that from the third interval its effect gets unnoticeable. Thankfully, this cost is independent of the density and hence does not affect Trickle's scalability. Note that a slight modification to New-Trickle by allowing nodes to start listening at time t of an $Imin$ interval can mitigate some of this extra cost.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCOMM.2015.2408339, IEEE Communications Letters

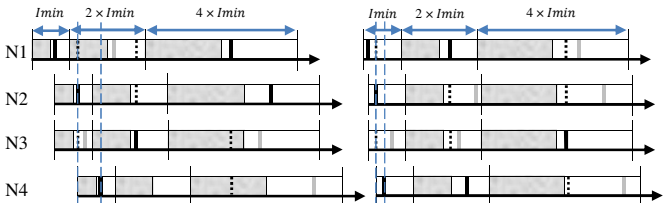

Fig. 4. Lossy network, Trickle (left) and New-Trickle (right). The dashed blue lines show the transmit-listen interplay between $Imin$-intervals' transmissions and following intervals listen-only periods.

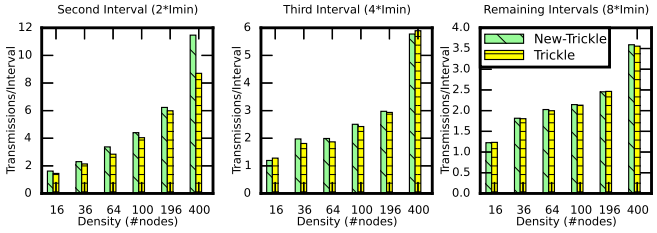

Fig. 5. Quantifying the additional cost

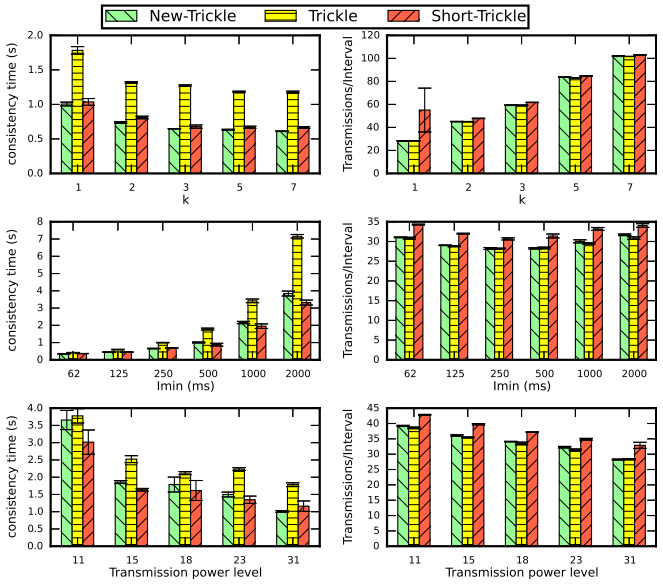

Fig. 6. Performance evaluation in the Indriya testbed

D. Testbed Results

The proposed algorithm was also evaluated in the public large-scale Indriya testbed [5]. Indriya contains around 100 active motes irregularly deployed in a three story building. At the time of experimentation almost all middle floor nodes were off, leaving us with 65 motes and a good opportunity to test in an irregular faulty real-world scenario. The Trickle seed (node 21 in the third floor) injected a new packet every 60 seconds. This is to create a network dominated by inconsistent traffic in order to show the impact of the observed small extra cost. Each experiment run for 30 minutes and was repeated three times. The default value of $Imin$ was half a second and that of k was one. Default transmission power level was 31. As in simulations, we report the mean along with its standard error.

Fig. 6 presents the consistency times and the transmission costs of the three evaluated protocols. As can be observed from the first row of graphs, New-Trickle provided the best of both Trickle and Short-Trickle, even in a faulty irregular network experiencing heavy inconsistencies. Thus, it propagated even better than Short-Trickle, which is about twice faster than Trickle at a similar transmission cost with increased k values. The second row of graphs in Fig. 6 shows New-Trickle performance when varying $Imin$. As can be seen from these graphs, New-Trickle propagated faster than Trickle while generating similar number of packets. The small additional cost is due to the fact explained earlier. Note that even in this irregular faulty network, New-Trickle's propagation is less affected by the value of $Imin$. The third row of graphs depicts the performance of the evaluated protocols when varying transmission power. Varying transmission power plays a double role; it changes both the density and the loss pattern of the network. New-Trickle propagated about twice faster than Trickle even in a lossy, less connected network (power level 15). This propagation time approached that of Short-Trickle while generating similar overhead to Trickle.

VI. Impact of the New Trickle Algorithm

All Trickle-based applications can benefit from New-Trickle by only modifying one line in their codes. Due to brevity, we focus here on RPL [3] and MPL [4]. Since Trickle is one of the main factors dictating the convergence of RPL networks, New-Trickle allows faster network convergence even when opting for bigger $Imin$ values (for the sake of minimizing collisions and hidden terminals). On the other hand, MPL will be able to deliver multicast packets much faster, especially as it heavily relies on Trickle for both its reactive and proactive modes.

VII. Conclusion

In this letter, we introduced a simple, yet very powerful optimization to Trickle. Results showed important performance enhancements in the consistency time of Trickle, yielding a factor greater than 10 times, while preserving Trickle's scalability. In-depth analysis of the impact of this optimization on Trickle-based applications along with other important benefits and improvements will be the subject of future work.

Acknowledgment

The authors would like to thank Philip Levis for the active insightful and fruitful discussions.

References

[1] P. Levis, N. Patel, D. Culler, and S. Shenker, Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks, in In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI, 2004, pp. 1528.

[2] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, RFC 6206: The Trickle algorithm, IETF, 2011.

[3] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. P. Vasseur, and R. Alexander, RFC 6550: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks, IETF, Mar. 2012.

[4] J. Hui and R. Kelsey, Multicast Protocol for Low power and Lossy Networks (MPL), Internet Draft, IETF, 2014.

[5] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda, Indriya: A low-cost, 3D wireless sensor network testbed, in Testbeds and Research Infrastructure. Development of Networks and Communities, Springer, 2012, pp. 302316.

Optimizing the trickle algorithm

Djamaa, Badis

2015-01-16

This work has been published in IEEE Communications Letters vol 19 part 5 pp 819-822