IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 4, 2020, accepted November 19, 2020, date of publication November 24, 2020,
date of current version December 10, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3040160

A^2-Trickle: Adaptive & Aligned Trickle for Rapid and Reliable Dissemination in Low-Power Wireless Networks

GUNMO JEONG, MINGYU PARK, (Graduate Student Member, IEEE), AND JEONGYEUP PAEK, (Senior Member, IEEE)
Department of Computer Science and Engineering, Chung-Ang University, Seoul 06947, Republic of Korea

Corresponding author: Jeongyeup Paek (jpaek@cau.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIT) under Grant 2020R1F1A1051282, and in part by the Chung-Ang University Graduate Research Scholarship in 2019.

ABSTRACT Trickle is a scalable and robust flooding algorithm designed for low-power multihop wireless networks. It aims to reduce redundant packet transmissions and collisions while maintaining information consistency, and has been popular due to its simple yet efficient operation on most general topologies. However, Trickle fails to achieve reliability and low latency under certain, but not uncommon, scenarios. To tackle this problem, we propose A^2-*Trickle*, a light-weight enhancement to the Trickle algorithm that guarantees rapid and reliable dissemination under any topology. A^2-*Trickle* aligns the interval boundary at propagation times without synchronizing the global clock, and adapts to the network topology for nodes that could suffer from Trickle's naïve suppression mechanism. A^2-*Trickle* is implemented on a real embedded device and is evaluated in various scenarios and topologies through both testbed experiments and simulations. The results reveal that A^2-*Trickle* adapts to the network and enables faster and more energy-efficient dissemination while maintaining >99% reliability.

INDEX TERMS Low-power and lossy network (LLN), trickle algorithm, dissemination protocol, wireless sensor network (WSN), Internet of Things (IoT).

I. INTRODUCTION

With the development of embedded devices and low-power wireless networking technology, the Internet of Things (IoT) era has arrived. Today, IoT technology is applied to numerous real world applications such as smart factory, smart market, smart hospital, smart home, and smart grid AMI [1]–[8]. These IoT applications primarily collect sensor data from nodes in the network, but also command the nodes to execute specific tasks. Moreover, a central node (e.g. server, gateway, or similar) could propagate information to the entire network for various purposes such as network configuration or over-the-air (OTA) software updates. Therefore, IoT applications require effective mechanisms not only to collect data but also to disseminate information.

Information dissemination can be done easily in a single-hop network with a simple broadcast. However, as the network topology becomes more complex with multihop and non-uniform physical deployment, data propagation becomes more challenging and the complexity increases dramatically with the number of hops in the network, especially for low-power and lossy networks (LLN). Three basic building blocks exist to propagate data over a multihop wireless network: iterative unicast, recursive broadcast (a.k.a flooding), and the Trickle timer algorithm [9]. Many network protocols are built on top of these primitives.

Iterative unicast has an advantage of being able to check the progress of each node and support end-to-end reliability. This is the reason why iterative unicast is widely used in many high-end devices. However, it does not scale well. For example, to forward a message to N 1-hop neighbors, iterative unicast requires at least N transmissions whereas broadcast requires only one. For a binary tree-like h-hop multihop network with $N = 2^h$ nodes, iterative unicast requires $\Omega(N \log N)$ transmissions whereas recursive broadcast (explained below) requires $\Omega(N)$, not to mention significantly longer network-wide latency.

Recursive broadcast is widely used in resource-constrained embedded devices due to its simplicity where a node simply re-broadcasts what it has heard. Ideally, it is significantly

The associate editor coordinating the review of this manuscript and approving it for publication was Chun-Wei Tsai.

faster than iterative unicast for dissemination. While doing so, a node usually uses a random time jitter within a specified range to mitigate synchronized transmissions and collision problem, and also ignores the previously-heard packets to avoid broadcast-storm problem. Increasing the time jitter range and transmitting slowly can lower the probability of collision and achieve higher reliability, but at the cost of latency increase. (Re)transmitting several identical packets can also enhance reliability, but wastes the channel bandwidth and energy. This means the recursive broadcast method has an explicit trade-off between reliability, latency, and overhead as the configuration changes.

Trickle algorithm is designed to address the above issues for multihop networks [9]. It has two representative techniques to reduce the number of redundant packets and collisions: halving with doubled intervals, and suppression. At the end of each transmission, Trickle *doubles the transmission interval* to reduce channel usage and collision probability unless it detects network inconsistency. In addition, it divides each interval in half and randomly chooses the transmission time within the *later half* of the interval. Until the chosen time, it listens to and counts the number of duplicate packets. At transmission time, Trickle *suppresses* the transmission if the duplicate counter exceeds a certain threshold. With these simple schemes, Trickle enables reliable, responsive, and energy-efficient dissemination over a multihop network with less overhead. For these reasons, Trickle has been widely applied in various standard protocols and IoT applications such as the IPv6 Routing Protocol for LLN (RPL) [10], [11], Multicast Protocol for LLN (MPL) [12], OTA programming for wireless sensor networks (WSN) (e.g. Deluge [13]), and task propagation in WSN (e.g. Maté [14], Tenet [15]).

Despite its advantages, Trickle algorithm still has some aspects that must be improved. Although Trickle tries to avoid collisions by using only the later half of an interval, collisions can still occur frequently because the transmission periods overlap between neighboring levels of dissemination (*tx-interval overlap problem*). In addition, its suppression scheme may *delay the propagation or even fail to deliver* in some but not uncommon topology with limited paths (*na ive suppression problem*). This leads to longer latency, low packet reception ratio (PRR), and low throughput for information propagation and convergence.

To address these problems, this work proposes A^2-*Trickle*, an adaptive & interval-aligned Trickle for rapid and reliable data dissemination in low-power multihop wireless networks. A^2-*Trickle* (1) *aligns and tiles* the transmission period boundaries hop-by-hop to reduce collisions from *tx-interval overlap problem*, and (2) applies an *adaptive suppression* scheme to overcome *na ive suppression problem* in the limited path scenario. A^2-*Trickle* is implemented on a real low-power embedded device, and is evaluated in various scenarios and topologies through both a real-world experiment and simulations. The results indicate that A^2-*Trickle* can choose a better suppression counter according to the network topology, and reduce the number of transmissions and convergence time while ensuring superior packet reception ratio (PRR).

The contributions of this paper are as follows:

- The problem of overlapping transmission periods is demonstrated, and a transmission period aligning & tiling scheme with little overhead is suggested.
- A topology-adaptive mechanism is proposed that reduces latency and energy usage while maintaining >99% PRR for 2 message disseminations per second even in bottlenecked topology with 100 nodes.
- A^2-*Trickle* is implemented on a real low-power embedded platform, and is compared with the standard Trickle algorithm through testbed experiments on 31 devices and extensive simulations on ∼100 nodes.

This paper is organized as follows. We first discuss how the paper contributes to the body of knowledge in comparison against related literature in Section II. Then, Section III provides an overview of the Trickle timer algorithm, and discusses the limitations of Trickle using several examples. We present the design of A^2-*Trickle* in Section IV, and then evaluate A^2-*Trickle* through simulations and testbed experiments in Section V. Finally, we conclude the paper with future work in Section VI.

II. RELATED WORK

Trickle was originally designed for over-the-air (OTA) code update in wireless sensor networks [9]. However, it was soon adopted by many other WSN/LLN protocols and applications for its potential as an effective data propagation and multicast mechanism [10], [12]–[16]. For the same reason, several research work have been conducted to improve the performance of Trickle in various aspects [17]–[22].

Dynamic Trickle [17] adaptively resizes the boundary of listen-only period in each node based on the number of neighboring nodes to reduce the convergence time when a node has few neighbors. However, it encounters an energy-efficiency issue when nodes have few neighbors because the transmission interval becomes too small. The authors claimed that this problem can be mitigated by never resetting c unless inconsistency is detected, but this is a critical problem if a dissemination bottleneck exists in the network as discussed in Section III. There have been attempts to make Trickle recover quickly from inconsistency by choosing t from $[0, I_{\min})$ instead of $[I_{\min}/2, I_{\min})$ for the first interval [18], [19]. In these approaches, however, node transmission intervals could overlap, harming the original rationale of collision avoidance between neighboring nodes in the standard Trickle.

Moreover, a few studies have considered the load balancing problem in Trickle. For example, the work in [20] argues that the battery imbalance of nodes eventually influences convergence time, and proposes *Energy-Aware Adaptive Trickle* (EAAT) that uses battery usage as a metric. It chooses the threshold K based on the predicted energy usage and residual battery of each node to balance both the energy and flow of transmissions. In *Adaptive-K* [21], each node calculates

the threshold K by multiplying the redundant counter c with an additional constant α to achieve load balancing and fast convergence. However, none of these studies consider nor resolve the bottlenecked topology issue where some node are left behind unnotified due to naïve suppression.

In *Trickle++* [22], a receiver resizes $I_{\min}$ based on the distance from the sender predicted using RSSI and LQI. If a node sends new data, all receivers switch to the *proactive mode*, and the farther receivers forward data faster than the nearer ones to disseminate the data farther quickly. However, if outdated data are detected, closer nodes have a better chance to transmit the recovering data in *reactive mode*. By switching modes, *Trickle++* can reduce redundant packets, but collision may occur frequently in a sparse network due to its non-uniformity of $I_{\min}$ in each node based on distance.

As an alternative to Trickle focusing on code dissemination in duty cycled WSNs, Shu *et al.* [23] proposed proportion to duty cycle length based broadcast (PDLB) scheme. In PDLB, each node selects appropriate k for broadcasting, and counter-based priority broadcasting approach is used to limit the broadcasting of nodes to enable data to be fully diffused with reduced latency and energy consumption. This work presented an insightful analysis on the relationship between duty cycle and transmission delay. However, the proposed protocol is geared specifically towards code dissemination in duty-cycled WSN, and is quite complex while one of the main advantage of Trickle and *A²-Trickle* is their simplicity and ease of use in general settings.

III. PROBLEM AND MOTIVATION

Trickle algorithm was first proposed as an efficient flooding algorithm for wireless sensor networks, and it has later been standardized in RFC 6206 [16] for low-power and lossy wireless networks in general. Trickle aims to reduce redundant packets in the network while maintaining network consistency by controlling the timing and rate of flooding. To do so, Trickle adopts two representative schemes: the transmission rate adaptation, and redundant packet suppression. Figure 1 illustrates the operation of Trickle.

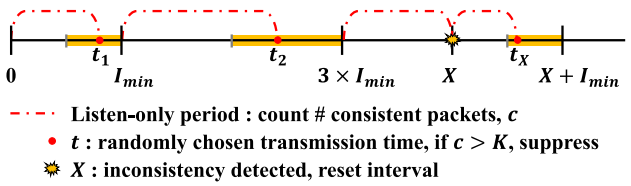


FIGURE 1. Basic operation of Trickle.

A. TRANSMISSION RATE ADAPTATION

When a node receives a packet with new information/data that needs to be disseminated throughout the network, it should re-broadcast the packet to pass the data to its neighbors that has not yet received them. Quick and aggressive transmission at each node may achieve faster propagation and inconsistency resolution in the network. However, it could incur *broadcast storm* problem similar to *ack explosion* problem, leading to network congestion and waste of bandwidth. On the other hand, if transmitted too slowly, network could suffer from low reactivity and longer latency, which leads to data inconsistency.

Trickle adapts the transmission rate considering this trade-off by repeatedly doubling the transmission interval, thus transmitting fast in the beginning and slowing down exponentially. Upon detecting new information, Trickle chooses its transmission time t within the second half of the initial interval $I_{\min}$ (e.g. t_1 in Figure 1). At the end of each interval, Trickle doubles the interval length I and chooses t again from that interval. The interval is doubled until it reaches $I_{\max}$, and if so, the interval is fixed as $I_{\max}$. To respond to network dynamics promptly, Trickle resets the interval length to the initial value $I_{\min}$ when it detects any data inconsistency (including new data) or configuration changes.

B. REDUNDANT PACKET SUPPRESSION

In a single hop network with N nodes where all nodes are within transmission range of each other, a single link broadcast will suffice for dissemination. If all N-1 receiving nodes rebroadcast, that will be N-1 redundant, useless and unnecessary packet transmissions (in an ideal case of 100% reception, of course). Even in a more realistic scenario where link PRR is not 100% and a few retransmissions may be required (especially due to lack of ACK for link broadcasts), simultaneous transmission of too many identical packets will do nothing but cause network congestion and channel waste [24]. For this reason, Trickle tries to send minimal redundant packets while maintaining network consistency by adopting a suppression mechanism.

A Trickle node divides its transmission interval into half and uses each for a *listen-only period* and *transmission period*, respectively. Then, the node randomly chooses a transmission time t within the *transmission period* (second half of the transmission interval), not the entire I period, and listens to the number of identical packets c from neighboring nodes. At time t, the node checks whether c exceeds a certain threshold K. If c is greater or equal to K, it presumes that there has been sufficient amount of identical transmissions for every neighboring node to have already received the information and does not transmit to reduce redundant packets. Moreover, as the interval is halved, the probability of collision decreases between 1 hop depth away (subsequent level) neighbors.

C. PROBLEM

Trickle has been successful in various use cases, leading to an IETF standard and adoption in many network protcols [10], [12]–[16]. However, Trickle may still exhibit extremely poor performance in several specific scenarios.

First, although the collision probability between next hop/depth nodes is reduced by halving the intervals, it could still be high due to enlarged overlapping areas as the number of neighboring nodes increases. In Figure 2, for example,

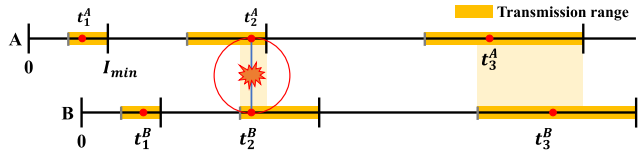


FIGURE 2. Possible scenarios for un-synchronized Trickle.

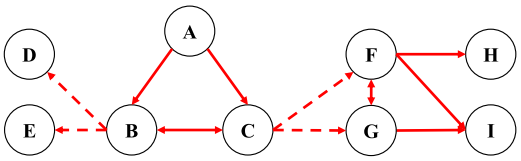


FIGURE 3. Bottleneck scenario.

node A randomly chooses its transmission time t_1^A. At t_1^A, node A transmits a packet (with new data), and node B starts its transmission interval timer upon reception. In this case, subsequent intervals of node B overlaps with the intervals of node A as shown in the figure, leading to collisions between the two. Furthermore, the overlapped area enlarges as time passes due to doubling. This means that a Trickle node must compete with not only the nodes in the same level of dissemination, but also nodes at different levels, which nullifies the whole intension of 'halving the interval' scheme. This problem is due to unsynchronized interval boundaries, and becomes more severe as the number of neighboring nodes increases.

Second problem of Trickle is the *na ive suppression problem*. A Trickle node suppresses its transmission if the redundancy counter c equals or is larger than the preconfigured threshold K. In Figure 3, for example, suppose node A initiates a dissemination and K is set to 1. If node C chooses its transmission time to be later than node B, node C will receive node B's message before its transmission and increase c to 1. Then at transmission time, node C will suppress its transmission. However, nodes F and G can receive new messages only from node C as illustrated in the figure. Therefore, nodes F and G cannot receive the packet, and nodes H and I will also never be updated. However, if node C transmits earlier than node B, nodes D and E will not receive the message for the same reason. This can lead to a critical problem of *data inconsistency* and *information partitioning* if any bottleneck exists in the network.

These problems reveal the necessity of an improved and adaptive mechanism to disseminate information over a multihop network, which we design in the next section.

IV. DESIGN

A^2-*Trickle* includes the following simple yet novel methods to enhance Trickle: 1) aligning the interval boundary, 2) tiling the transmission period, and 3) adaptive suppression. Interval alignment helps neighboring nodes to determine when the interval of the previous sender has started without synchronizing the clock. This accelerates the start of a new interval and helps avoid overlaps or collisions together with tiling. Tiling is performed to eliminate overlaps in transmission periods between adjacent levels of dissemination, even for the enlarged interval ranges. Through adaptive suppression, an A^2-*Trickle* node counts neighbors that can receive messages only from itself, and forwards messages without suppression in those cases to avoid information partitioning and achieve robust dissemination.

A. ALIGNING INTERVAL BOUNDARY

To address the *tx-interval overlap problem* described in Section III, A^2-*Trickle* first aligns the interval boundary among neighboring nodes. If all nodes are synchronized to a common network-wide clock, it becomes trivial for the receivers to synchronize their Trickle intervals with senders. However, sharing a global clock not only requires significant extra overhead and sophistigated protocols [25]–[27], but it may be an overkill since only the interval boundaries need to be synchronized. Thus, to maintain simplicity of Trickle, we consider a light-weight approach.

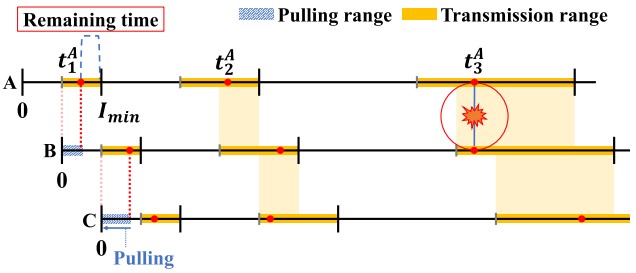


FIGURE 4. Interval boundary alignment.

A^2-*Trickle* aligns the interval boundary among neighboring nodes by informing them of its own progressed time within its current transmission interval. To inform the neighbors, an A^2-*Trickle* node piggybacks the 'remaining time' in the scheduled forwarding packet that it transmits. For example, consider sender A and receiver B in Figure 4. Node A transmits the remaining time calculated as $I_{\min} - t^A$, where t^A is the transmission time chosen by node A. Then, receiver B can start its interval timer at the same time as the start of the first transmission period of node A, which is half of the transmission interval of node A (see Figure 4). In this way, node B aligns the start of the interval with transmission period of the sender, and eliminates the overlap for the 'first intervals' of subsequent nodes (e.g. A, B, and also C). In addition, as the beginning of the interval is pulled forward for every subsequent nodes, propagation is more likely to be faster. However, this is insufficient to eliminate the overlaps in the second and later intervals as shown in Figure 4, which leads to the second technique below.

B. TILING THE INTERVAL

Since the transmission intervals are now aligned using the aforementioned scheme, overlapped areas in the transmission

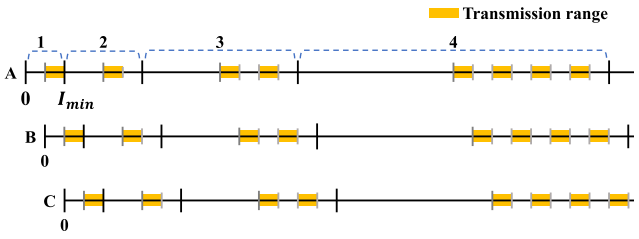
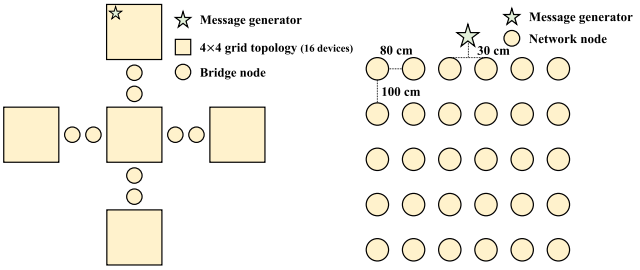


FIGURE 5. Tiled intervals can eliminate overlaps in transmission periods.



(a) Cross topology for simulation (b) A real world testbed

FIGURE 6. Evaluation topologies.

periods can be eliminated by the 'tiling' scheme. Specifically, *A*^{2}*-Trickle* re-organizes the transmission period into tile-pattern like blocks with repeated small ranges as displayed in Figure 5. The size of each block (or tile) is identical to the transmission period of the first interval, and the number of blocks in each previously a transmission period must be an even number since Trickle always doubles the interval. Therefore, always choosing the odd (or even) tiles as the new transmission period (orange shaded area) completely isolates the transmission period between neighboring nodes as the first interval. The total amount of resulting transmission period within each interval is halved compared to the original Trickle after the second interval, but this neither delays the propagation nor increases the collision probability considering the exponential doubling of intervals; this is no different from shifting the doubling by just one step, and it only reduces the collision probability through isolation. In Figure 5, for example, node *B* chooses a random transmission time *t* among odd-numbered tiles (i.e., the highlighted ones), and 1-hop away nodes *A* and *C* are both in the listen-only state. As a result, *A*^{2}*-Trickle* nodes only compete with the same level neighbors, never with the next or previous level nodes. Therefore, aligning and tiling the interval together reduces collisions and allows faster and smoother propagation.

C. ADAPTIVE SUPPRESSION

We have discussed in Section III that using a fixed K value for suppression threshold may cause problems in a bottlenecked topology such as Figure 3; a node that can only receives messages from one previous hop neighbor may never receive that data if the neighbor suppresses packets through Trickle's naïve suppression. A higher suppression counter can mitigate the problem, but it also has the disadvantage of channel waste and congestion. Therefore, a network manager should choose an appropriate parameter considering various scenarios such as node failure or uneven physical deployment. Or better would be to adapt the parameter to the network dynamically.

A^2*-Trickle* adapts the suppression counter by counting the number of neighboring nodes. An *A*^{2}*-Trickle* node remembers and increases its *rank* (i.e. level, in terms of breath-first search like propagation) for each dissemination before forwarding. Rank indicates the shortest hop count from the message generator to this node. Then, whenever a node receives a message, it may take one of the three actions according to the rank of the sender of that message. First, the node increases the number of same-ranked nodes N_{same} if the node receives messages from same ranked neighbors. If the node receives messages from lower-ranked neighbors, it counts the number of lower-ranked nodes N_{lower}, and carries the number into the forwarding messages. The node compares and records N_{min} as the minimum N_{lower} value whenever the node receives messages from *higher-ranked* neighbors. At the end of each interval, the node calculates the K value as expressed in Eq.(1).

$$K_{\text{next}} = \begin{cases} \infty & \text{if } N_{\text{min}} == 1 \\ \lceil N_{\text{same}}/N_{\text{min}} \rceil & \text{if } N_{\text{min}} > 1 \\ K_{\text{default}} & \text{if } N_{\text{min}} == 0 \end{cases} \quad (1)$$

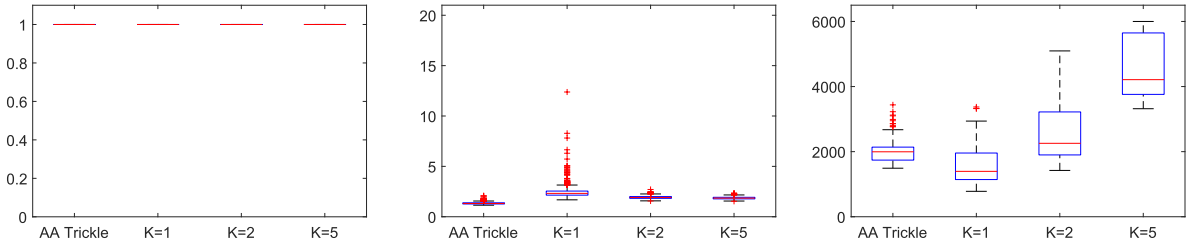
Intuition is that, if N_{min} is 1, then there must be neighboring nodes that can receive new messages from this node only. Thus the node must forward messages for information consistency regardless of suppression counter. This scheme ensures that a node that can receive messages only from one node can receive messages faster and more reliably. From Eq.(1), an *A*^{2}*-Trickle* node uses a low K value if its higher-ranked neighbors are capable of receiving from enough other nodes, but selects a high K if it observes that its higher-ranked neighbor(s) have few lower-ranked neighbors. Because *A*^{2}*-Trickle* changes the suppression threshold for every interval, it can adapt to network dynamics promptly.

V. EVALUATION

In this section, we evaluate *A*^{2}*-Trickle* and compare it with the standard Trickle algorithm with varying configurations. We experiment on a real 31-node indoor testbed, and also run simulations on 102 different topologies (1 grid, 1 cross, and 100 random) with ∼100 devices.

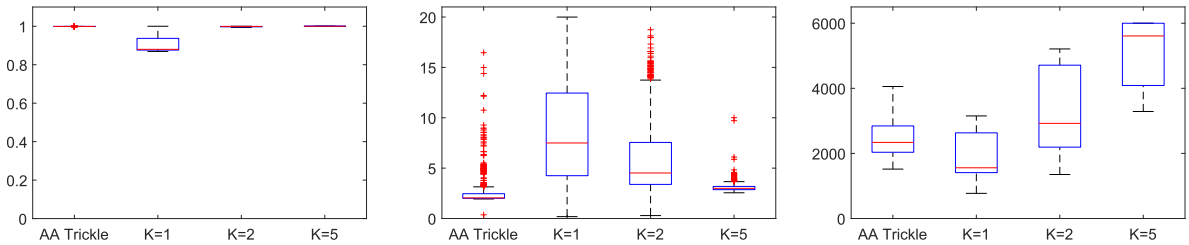
A. EVALUATION SETUP

We implement *A*^{2}*-Trickle* and Trickle algorithms using TinyOS 2.1.2 [28] on TelosB [29] platform which consist of an MSP430 microcontroller and a CC2420 radio transmiting messages over IEEE 802.15.4 links. For the default application layer, a source node generates 1000 unique messages every 20 seconds (unless stated otherwise), and we define loss as when a node does not receive a message until a new message is generated.



(a) Packet reception ratio. (b) Convergence time (seconds). (c) Number of transmissions

FIGURE 7. Simulation results from the GRID topology.



(a) Packet reception ratio (b) Convergence time (seconds). (c) Number of transmissions

FIGURE 8. Simulation results from the CROSS topology.

Performance of Trickle depends on how the suppression threshold parameter K is configured. Using a small K allows energy-efficient operation, and a large K improves reliability (and maybe latency) of data propagation. Since we evaluate on 103 different topologies (1 testbed, 102 simulations), there is no good K value that fits all. Therefore, every experiment and simulation was conducted using three different values of K: 1, 2 and 5.

Three metrics are used for evaluation: 'convergence time', 'packet reception ratio', and 'total number of transmissions'.

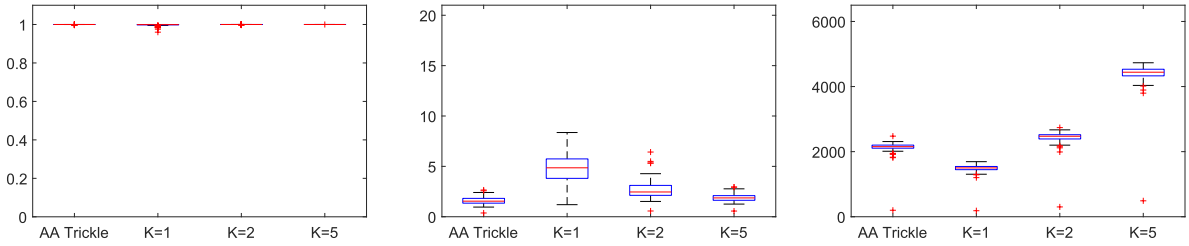
- Convergence time is the time needed to propagate a message into the entire network, and is the most important evaluation metric.
- Packet reception ratio (PRR) is the ratio of unique messages received by each node against the total number of messages generated by the source, end-to-end, excluding duplicates. Note that although Trickle ensures eventual consistency (100% PRR) for a single packet, a message is lost when a new message is received before receiving the previous one.[1]
- Total number of transmissions represents how the algorithm is effectively suppressing redundant packet for energy and channel usage efficiency.

B. SIMULATIONS ON VARIOUS TOPOLOGIES

Simulations are conducted using Cooja simulator [30] with Unit Disk Graph Medium (UDGM) propagation model. Out of 102 topologies used, the 'grid' and 'cross' are constructed manually, and 100 are generated randomly. For the grid topology, 100 devices are deployed regularly in 10 $\times$ 10 formation, and the source node (message generator) is placed at the top-left corner. Cross topology is depicted in Figure 6a where a square represents a small-scale network consisting of 16 nodes in 4 $\times$ 4 grid form, and circle nodes are bottleneck nodes that relay messages between these square networks. This topology is designed to mimic uneven physical deployments in the real-world where limited and bottlenecked paths exist in the network such that *na ive suppression problem* may cause information inconsistency. The distance between each node in both the grid and cross topology is set to 30 m, and the transmission range is set to 50 m.

Simulation results are plotted in Figures 7, 8 and 9 for grid, cross, and random topologies, respectively. The results are in large consistent: (1) Low K value (i.e. $K = 1$) results in lower number of transmissions and thus lower energy usage, but at the cost of lower PRR and significantly longer convergence time. (2) High K value (i.e. $K = 5$) results in 100% PRR and low convergence time, but at the cost of significantly larger number of transmissions leading to higher channel and energy usage. (3) The cross topology (Figure 8), where the *na ive suppression problem* is most likely, accentuates the trend the most with significant loss (<90% PRR) and significantly longer convergence time. (4) *A^2-Trickle* adaptively achieves best of all K's in all scenarios; 100% PRR with lowest convergence time while maintaining the number of transmitted packets to those between $K = 1$ and $K = 2$ configurations.

[1]Of course, some buffering can alleviate this loss like in [13]–[15], but does not fundamentally change the characteristics of flooding.



(a) Packet reception ratio. (b) Convergence time (seconds). (c) Average number of transmissions.

FIGURE 9. Simulation results averaged from 100 RANDOM topologies.

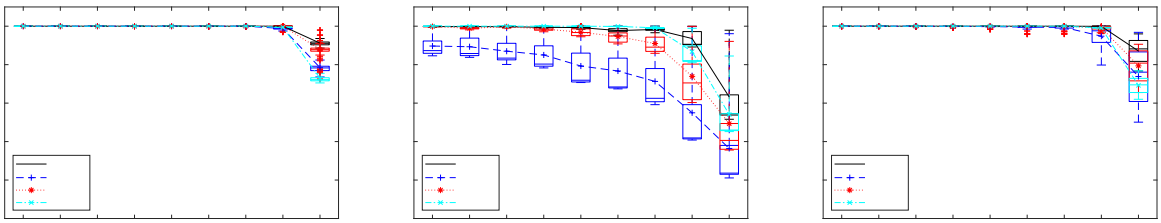
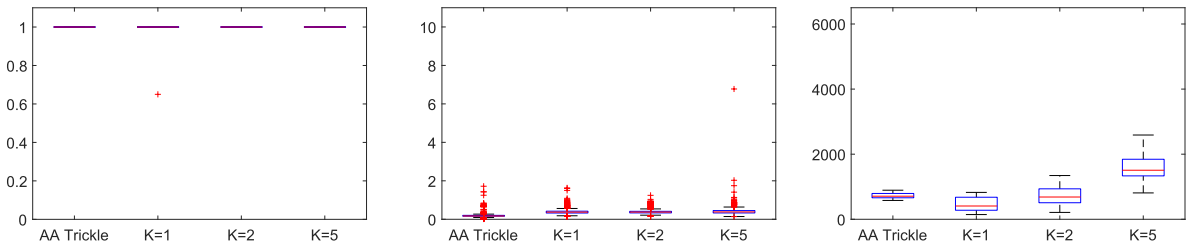


FIGURE 10. Dissemination success ratio vs. inter-packet generation interval (seconds). This result implies dissemination throughput.



(a) Packet reception ratio. (b) Convergence time (seconds). (c) Number of transmissions.

FIGURE 11. Testbed experiment results.

It would be a waste of resources to use a high K value under 100% PRR conditions, and thus, the network manager would need to manually determine the minimum K value while considering the tradeoff between the convergence time and energy efficiency. A^2-*Trickle* eliminates the need for such manual configuration, and adapts to the network autonomously to provide the best performance achieved by Trickle with any K configuration: higher PRR with lower convergence time with just enough transmissions.

Dissemination Throughput: Next, we explore how frequenty a sequence of messages can be propagated into the entire network without losing packets and maintaining information consistency. This determines, for example, how quickly we can complete OTA re-programming of the network. Figure 10 plots the dissemination success ratio (network-wide PRR) versus inter-packet generation interval (unit: seconds). Clearly, cross topology is the most challenging one, and A^2-*Trickle* outperforms Trickle regardless of K configurations on all topologies. In the grid or random topology, A^2-*Trickle* can maintain 99% delivery up until 1 dissemination per second, where as Trickle starts to see inconsistency at 3 seconds intervals. In the cross topology, while A^2-*Trickle* maintains 99% up to a message every 2 seconds, Trickle suffers devastatingly much early on (as can be seen in Figure 10).

C. TESTBED EXPERIMENT

Simulation results indicate that A^2-*Trickle* propagates messages faster and chooses an appropriate suppression threshold according to the network topology. In real-world environments, many unpredictable variables exist such as multipath and fading due to walls or human occupants, interference from other wireless technology and so on. Therefore, experiments must be conducted to ensure that A^2-*Trickle* works in real environments.

We configured an indoor testbed with 31 devices in an office environment. One device is the source node that generates unique messages, and 30 other devices receive those messages over multihop LLN. The 30 LLN nodes are deployed in a 6×5 grid formation as depicted in Figure 6b, and we set the radio tx power to -25 dBm to configure a multihop network.

Figure 11 presents the experimental results where the trend is consistent with the simulation results; all cases achieve 100% PRR except for when $K = 1$, the number of transmission for A^2-*Trickle* is in between when $K = 1$ and $K = 2$, and A^2-*Trickle* has the fastest convergence time while maintaining this property. The important finding from this result is that A^2-*Trickle* still outperforms the original Trickle in the real experiments on testbed environment.

VI. CONCLUSION

A^2-*Trickle* is an enhanced Trickle algorithm for rapid and reliable data dissemination in LLN environments. It was motivated by the fact that the widely used Trickle algorithm fails to achieve reliability and low latency under some, but not uncommon, topologies. To ensure robustness and faster convergence for information consistency, we adopted three ideas: (1) aligning the interval boundary, (2) tiling the intervals, and (3) adaptive suppression. Through these simple yet effective mechanisms, A^2-*Trickle* can forward messages faster, evade collisions better, and most importantly, avoid leaving any node unnotified. Furthermore, A^2-*Trickle* finds an appropriate set of parameters during the convergence and adapts to the network for the next propagation. A^2-*Trickle* is implemented on a real low-power embedded device, and has been evaluated through both experiments and simulations to show that it improves reliability, reactivity, and energy-efficiency compared to the standard Trickle algorithm.

We believe that A^2-*Trickle* can replace its predecessor in many Internet standard LLN protocols such as RPL [10] and MPL [12], and we leave analytical modeling of A^2-*Trickle* as our future work.

REFERENCES

[1] T. Voigt and F. Osterlind, "CoReDac: Collision-free command-response data collection," in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Autom.*, Sep. 2008, pp. 967–973.

[2] H.-S. Kim, H. Cho, M.-S. Lee, J. Paek, J. Ko, and S. Bahk, "MarketNet: An asymmetric transmission power-based wireless system for managing e-price tags in markets," in *Proc. 13th ACM Conf. Embedded Networked Sensor Syst.*, 2015, pp. 281–294.

[3] J. Ock, H. Kim, H.-S. Kim, J. Paek, and S. Bahk, "Low-power wireless with denseness: The case of an electronic shelf labeling system—Design and experience," *IEEE Access*, vol. 7, pp. 163887–163897, 2019.

[4] J. Park, W. Nam, J. Choi, T. Kim, D. Yoon, S. Lee, J. Paek, and J. Ko, "Glasses for the third eye: Improving the quality of clinical data analysis with motion sensor-based data filtering," in *Proc. 15th ACM Conf. Embedded Netw. Sensor Syst.*, Nov. 2017, pp. 1–14.

[5] D.-M. Han and J.-H. Lim, "Smart home energy management system using IEEE 802.15.4 and zigbee," *IEEE Trans. Consum. Electron.*, vol. 56, no. 3, pp. 1403–1410, Aug. 2010.

[6] B. L. Risteska Stojkoska and K. V. Trivodaliev, "A review of Internet of Things for smart home: Challenges and solutions," *J. Cleaner Prod.*, vol. 140, pp. 1454–1464, Jan. 2017.

[7] Z. Fan, P. Kulkarni, S. Gormus, C. Efthymiou, G. Kalogridis, M. Sooriyabandara, Z. Zhu, S. Lambotharan, and W. H. Chin, "Smart grid communications: Overview of research challenges, solutions, and standardization activities," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 1, pp. 21–38, 1st Quart., 2013.

[8] Cisco. (Nov. 2020). *Connected Grid Networks for Smart Grid—Field Area Network*. [Online]. Available: http://www.cisco.com/web/strategy/energy/field_area_network.html

[9] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proc. 1st Conf. Symp. Networked Syst. Design Implement. (NSDI)*, 2004, pp. 1–5.

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

[11] H.-S. Kim, J. Ko, D. E. Culler, and J. Paek, "Challenging the IPv6 routing protocol for low-power and lossy networks (RPL): A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2502–2525, Sep. 2017.

[12] J. Hui and R. Kelsey, *Multicast Protocol for Low-Power and Lossy Networks (MPL)*, document RFC 7731, 2016.

[13] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proc. 2nd Int. Conf. Embedded Netw. Sensor Syst.*, 2004, pp. 81–94.

[14] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks," in *Proc. 10th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2002, pp. 85–95.

[15] J. Paek, B. Greenstein, O. Gnawali, K.-Y. Jang, A. Joki, M. Vieira, J. Hicks, D. Estrin, R. Govindan, and E. Kohler, "The tenet architecture for tiered sensor networks," *ACM Trans. Sensor Netw.*, vol. 6, no. 4, pp. 1–44, Jul. 2010.

[16] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, "The trickle algorithm (RFC 6206)," in *Proc. Internet Engineering Task Force (IETF)*, 2011, pp. 1–13.

[17] M. B. Yassein, S. Aljawarneh, E. Masa'deh, B. Ghaleb, and R. Masa'deh, "A new dynamic trickle algorithm for low power and lossy networks," in *Proc. Int. Conf. Eng. MIS (ICEMIS)*, Sep. 2016, pp. 1–6.

[18] B. Djamaa and M. Richardson, "Optimizing the trickle algorithm," *IEEE Commun. Lett.*, vol. 19, no. 5, pp. 819–822, May 2015.

[19] B. Ghaleb, A. Al-Dubai, and E. Ekonomou, "E-trickle: Enhanced trickle algorithm for low-power and lossy networks," in *Proc. IEEE Int. Conf. Comput. Inf. Technol.*, Oct. 2015, pp. 1123–1129.

[20] A. Musaddiq, Y. B. Zikria, and S. W. Kim, "Energy-aware adaptive trickle timer algorithm for RPL-based routing in the Internet of things," in *Proc. 28th Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, Nov. 2018, pp. 1–6.

[21] T. M. M. Meyfroyt, M. Stolikj, and J. J. Lukkien, "Adaptive broadcast suppression for trickle-based protocols," in *Proc. IEEE 16th Int. Symp. World Wireless, Mobile Multimedia Netw. (WoWMoM)*, Jun. 2015, pp. 1–9.

[22] B. Djamaa, M. R. Senouci, and A. Mellouk, "Trickle++: A context-aware trickle algorithm," in *Proc. IEEE Global Commun. Conf.*, Dec. 2017, pp. 1–6.

[23] T. Shu, W. Liu, T. Wang, Q. Deng, M. Zhao, N. N. Xiong, X. Li, and A. Lssssiu, "Broadcast based code dissemination scheme for duty cycle based wireless sensor networks," *IEEE Access*, vol. 7, pp. 105258–105286, 2019.

[24] Y. Jang, Y. Kim, S. Park, and S. Choi, "Link adaptation strategies for IEEE 802.15.4 WPANs: Protocol design and performance evaluation," *J. Commun. Netw.*, vol. 21, no. 4, pp. 376–384, Aug. 2019.

[25] M. Marot, B. Kusy, G. Simon, and A. Ledeczi, "The flooding time synchronization protocol," in *Proc. 2nd Conf. Embedded Netw. Sensor Syst. (SenSys)*, Nov. 2004, pp. 39–49.

[26] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, 2011, pp. 73–84.

[27] (Aug. 2020).*IEEE Std 802.1AS—Timing and Synchronization*. [Online]. Available: https://www.ieee802.org/1/pages/802.1as.html

[28] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in *Proc. 9th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2000, pp. 93–104.

[29] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *Proc. 4th Int. Symp. Inf. Process. Sensor Netw.*, 2005, pp. 364–369.

[30] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with COOJA," in *Proc. 31st IEEE Conf. Local Comput. Netw.*, Nov. 2006, pp. 641–648.

GUNMO JEONG received the B.S. degree from the School of Computer Science and Engineering, Chung-Ang University, Seoul, Republic of Korea, in 2019. He is currently pursuing the M.S. degree with the Department of Computer Science and Engineering, Chung-Ang University. He is also a Research Assistant with the Networked Systems Laboratory (NSL) led by Dr. J. Paek, with research interest includes wireless sensor networking.

MINGYU PARK (Graduate Student Member, IEEE) received the B.S. degree in computer and information communications engineering from Hongik University, in 2017, and the M.S. degree in computer science and engineering from Chung-Ang University, Seoul, Republic of Korea, in 2019, where he is currently pursuing the Ph.D. degree in computer science and engineering. He is also a Research Assistant with the Networked Systems Laboratory (NSL) led by Dr. J. Paek.

JEONGYEUP PAEK (Senior Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, in 2003, and the M.S. degree in electrical engineering and the Ph.D. degree in computer science from the University of Southern California (USC), in 2005 and 2010, respectively. He worked as a Research Intern with Deutsche Telekom Inc., Research and Development Labs, USA, in 2010, and then joined Cisco Systems Inc. in 2011, where he was a Technical Leader with the Internet of Things Group (IoTG), Connected Energy Networks Business Unit (CENBU, formerly the Smart Grid BU). In 2014, he was an Assistant Professor with the Department of Computer Information Communication, Hongik University. He has been an Associate Professor with the School of Computer Science and Engineering, Chung-Ang University, Seoul, Republic of Korea, since 2015. He is also an ACM member.