Trickle-D: High Fairness and Low Transmission Load with Dynamic Redundancy

Mališa Vučinić, Michal Król, Baptiste Jonglez, Titouan Coladon, Bernard Tourancheau

Trickle-D: High Fairness and Low Transmission Load with Dynamic Redundancy

Mališa Vučinić* Michał Król† Baptiste Jonglez[3] Titouan Coladon[3]
Bernard Tourancheau‡

Corresponding author: Bernard.Tourancheau@imag.fr

Abstract

Embedded devices of the Internet of Things form the so-called low-power and lossy networks. In these networks, nodes are constrained in terms of energy, memory and processing. Links are lossy and exhibit a transient behavior. From the point of view of energy expenditure, governing control overhead emission is crucial and is the role of the Trickle algorithm. We address Trickle's fairness problem to evenly distribute the transmission load across the network, while keeping the total message count low. First, we analytically analyze two underlying causes of unfairness in Trickle networks: desynchronization among nodes and non-uniform topologies. Based on our analysis, we propose a first algorithm whose performance and parameters we study in an emulated environment. From this feedback, we design a second algorithm *Trickle-D* that adapts the redundancy parameter to achieve high fairness while keeping the transmission load low. We validate *Trickle-D* in real-life conditions using a large scale experimental testbed. *Trickle-D* requires *minimal* changes to Trickle, zero user input, emits 17.7% less messages than state-of-the-art and 37.2% less messages than state-of-practice, while guaranteeing high fairness across the network.

1 Introduction

The "Internet of Things" is becoming reality and for a multi-year lifetime its constrained devices must run on a very stringent energy budget. In this context, reducing energy consumption in all manners is a primary concern. As the consumption of the radio dominates over other on-board components, this goal translates into the reduction of radio operations, and on a higher level reduction of control overhead.

We address this and a related problem: *fairness* among nodes. That is, we want to ensure that each node in a multihop network consumes a similar amount of energy due to control overhead. Hence, no node will drain its battery significantly faster than others. Improving fairness allows to extend the whole network lifetime: see for instance the sink placement case in [22], avoiding situations, where some nodes fail sooner than others because of early battery depletion.

We focus our efforts on the *Trickle* [14] algorithm, which is used for data dissemination in constrained-node networks. Trickle has been standardized by the IETF [15], and is most notably used by the IETF Routing Protocol for Low-Power and Lossy Networks (RPL), see [25] and [24]. Other than RPL, Multicast Protocol for Low-Power and Lossy Networks [9] and other protocols like in [26, 7, 8, 23] and [20] build upon it, leveraging Trickle's benefits [4]. This makes the understanding of its behavior crucial for performance optimization of control overhead.

Our contribution is a twofold. First, we address the unfairness of Trickle and isolate two contributors: desynchronization and non-uniform topology. Second, we present a simple, yet effective algorithm to dynamically adapt one of Trickle's key parameters, while keeping the original Trickle unchanged. We then evaluate the algorithm in a realistic, emulated environment drawing conclusions about the effect of the parameters. The study allowed us to further simplify the algorithm and derive *Trickle-D*, a simplified version of the algorithm that dynamically adapts the redundancy notion of Trickle to achieve

*Inria-Paris, France, EVA team.
†Department of Electronic and Electrical Engineering, University College London, London, U.K.
‡Grenoble Alps University, CNRS, LIG laboratory, Grenoble, France.

high fairness and low transmission load. Finally, we validate *Trickle-D* in real-life conditions using large scale experimental testbed, showing performance improvements over both state-of-the-art and state-of-practice.

The remaining of the paper is organized as follows. Section 2 explains the Trickle algorithm and its usage within RPL. Section 3 presents the related works. Section 4 points out two causes of unfairness in Trickle and discusses their consequences under certain hypotheses. Section 5 then presents Trickle improved algorithms, and Section 6 gives simulation and experimental contexts that lead to validation and performance evaluation of our proposition on three network topologies of up-to-date hardware platforms.

2 Trickle

Trickle's original goal [14] was to quickly disseminate code updates throughout a network, while keeping the amount of communication at a minimum when every node has the newest version of the code. It turned out to be a simple yet effective way of distributing any kind of information. This makes its interest for IoT protocols such as RPL.

2.1 Trickle algorithm description

The main idea of Trickle revolves around the application-specific notion of *consistency*. On one hand, Trickle resolves any inconsistency by broadcasting the newest information as quickly as possible. On the other hand, when no new information is available, the number of messages should be kept at minimum which is achieved with a broadcast suppression mechanism. This saves energy in low-power networks by eliminating redundant communications.

Trickle organises time into intervals of variable size I. The first half of an interval is a listen-only period, in which a node listens but never transmits. A node picks a transmission time uniformly at random in the second half of the interval. When this selected transmission time is reached, the node decides whether to transmit or not based on the number of *consistent messages* received during the current interval.

To this end, a threshold *redundancy constant* k is introduced, describing how many consistent messages are necessary to be received in order to avoid the broadcast. Each node maintains a *consistent messages* counter c, updated according to the following rules:

1. increment c for each consistent message received
2. at the interval start, reset c to zero.

When the transmission time is reached, the node transmits if $c < k$. That is, if less than k consistent messages have been received in the current interval, then the current message is not considered redundant and it should be broadcasted. Otherwise, the message broadcast is considered redundant and is suppressed.

The size I of the Trickle interval is governed by the following rules:

1. at the end of an interval, I doubles, up to I_{max}
2. when a newer inconsistent message is received, I resets to I_{min}.

Figure 1 exemplifies the Trickle algorithm run in a trivial case which shows broadcast propagation and broadcast suppression mechanisms on a time interval.

2.2 Trickle in RPL

The *Routing Protocol for Low-Power and Lossy Networks*, specified in RFC 6550 [25], defines routing for constrained-node networks. A device in such network typically only knows how to reach the gateway of the network called the "sink". In order to participate in a RPL network, each node broadcasts control messages using the Trickle algorithm. The default redundancy constant specified by the RPL protocol is relatively high, $k = 10$. Although it is supposed to be configurable by the operator, deployments typically rely on this default value.

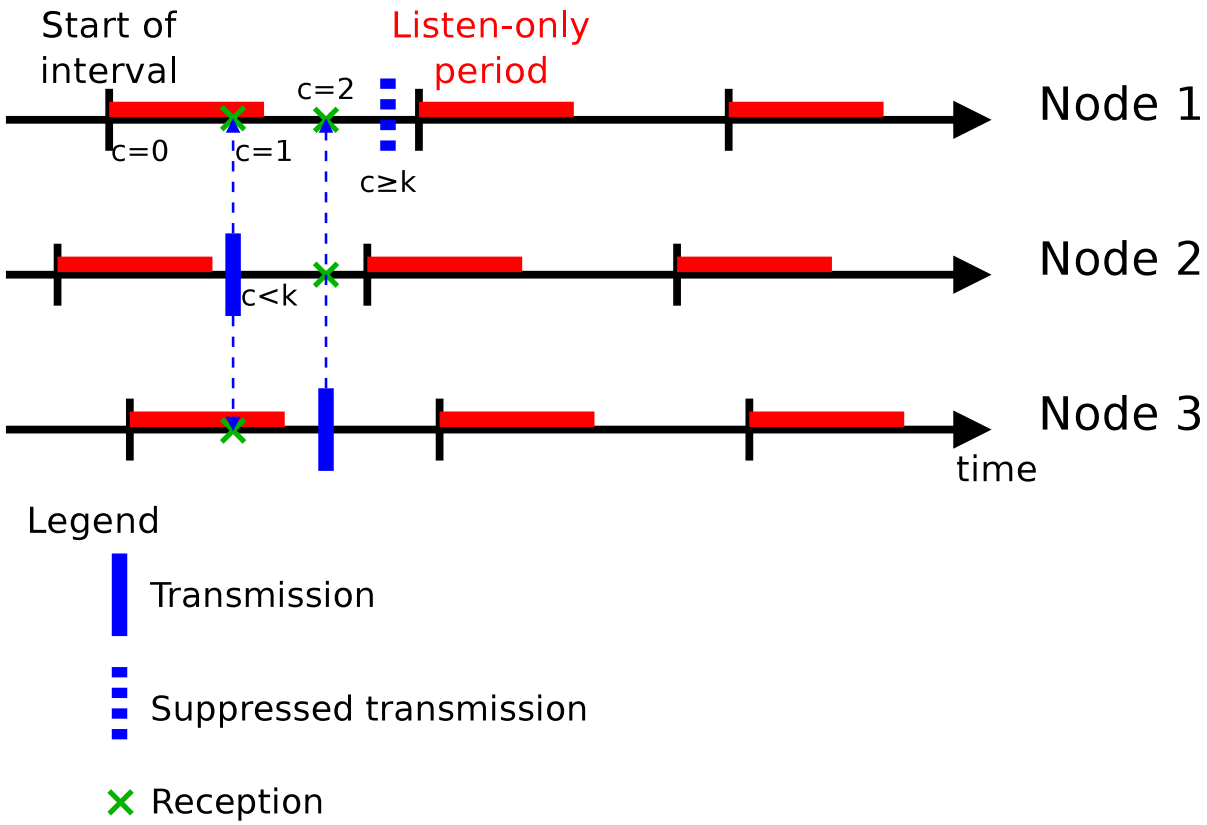

Figure 1: The Trickle algorithm time lines in $I = I_{max}$ permanent regime steady-state, with a full-mesh of 3 nodes, $k = 2$, and assuming all sent messages are consistent.

3 Related work

Several efforts have been made to model the behavior of Trickle, in particular to predict the total number of transmissions. In [13], authors derive the total message count of a steady-state, static and synchronized Trickle network, assuming a uniform spatial distribution of nodes. In [3] and [4], authors build an analytical model for a more general case: arbitrary network topology, and the nodes are not assumed to be synchronized. Additionally, the model predicts the probability of transmission of each individual node in the network, and not just the total transmission count for the network. While this gives a first measure of unfairness among nodes, the model only takes into account unfairness caused by the topology, and not unfairness caused by desynchronization.

The unfairness of Trickle has been mentioned in some works, in particular [15] and [19]. The effect of desynchronization on fairness has been analyzed in [18], in a more general case. The authors extend Trickle by introducing a new parameter $\eta \in [0, 1]$ controlling the length of the listen-only period. In the original Trickle algorithm, $\eta = 0.5$. This parameter impacts Trickle's fairness, in particular, high values of η decrease both fairness and message count, so that choosing a value for η involves a tradeoff. The authors also present a model to predict the nodes transmission probability in a single-cell network.

Trickle-F [23] is a modification of the Trickle algorithm designed to improve fairness. However, the authors' primary concern is to avoid communication starvation of nodes[1], while we are primarily interested in energy consumption. The idea of Trickle-F is to *"prioritize each node strictly depending on the number of consecutive suppressions: the longer the time spent by a node without transmitting, the higher its transmission priority in the next round"* [23]. To this end, each node keeps a counter s of the number of times its transmission has been suppressed, initialized at $s = 0$. Then, the node selects its transmission time t in $[\frac{I}{2^{s+1}}, \frac{I}{2^s}]$ instead of $[\frac{I}{2}, I]$. Thus, the longer a node has been suppressed, the earlier it will try to transmit in the following interval. With a synchronized network, all nodes with a

[1] A node starves if it cannot transmit during a very long time compared to its normal timescale of operation.

higher priority s will necessarily transmit before the nodes with a lower priority.

In order to dynamically balance Trickle's communications, [4] proposes to adapt k as a function of the node's degree using $k = \alpha * d$, where d is the node degree. This assumes d is known which is usually the case in the MAC layers of wireless networks. In [20], the authors use a similar approach with $k = \alpha * c$, where c can be seen as an approximation of the node degree. The value c is dynamically computed from scratch for each interval and adjusted within the interval. While the number of neighbors is not needed for c computation, the experiment results show that k is roughly proportional to the node degree.

Fairness is an important issue also in the MAC layer. For instance, in [16] authors address this issue using a hybrid mechanism for hybrid M2M networks. The solution uses priorities and both contention and transmission periods to achieve better performance in heterogeneous networks.

4 Trickle's unfairness

The following analysis demonstrates why the transmission load in a Trickle network is unequally spread between nodes, *i.e.,* some nodes (re)transmit more than others on average. We identify two main causes: desynchronization between nodes and non-uniform topologies.

We consider a static network of nodes, where nodes are able to talk to each other in an arbitrary topology, depending on their communication range. Each node runs Trickle to reach a consistent state with its neighbors with the same parameters I_{min} and I_{max}. As in [3, 4] and [20], we assume that the redundancy constant k is not necessarily the same on all nodes. The network is assumed to be in steady-state, *i.e.,* $I = I_{max}$, all nodes are consistent, and no node triggers a Trickle reset because of external events.

4.1 Desynchronization unfairness

Trickle does not assume that nodes are synchronized. Furthermore, desynchronization is the reason for the introduction of the *listen-only* period, fixing the *short-listen* problem [14]. This listen-only period reduces the number of transmitted messages, as described in Section 2. However, the listen-only period also introduces a bias in the transmission load, by hindering the ability of some nodes to transmit. This is referred to as the *capturing problem* in [18].

To understand this effect, we fully analyze the simple case of two unsynchronized Trickle nodes, with $k = 1$[2]. We derive the analytical probability of transmission for each node, as a function of the phase φ between the nodes as illustrated in Figure 2.

For the sake of simplicity, we assume that the Trickle interval is of unit length and that $0 \leq \varphi \leq 0.5$ because the situation of the two nodes is symmetric. $\varphi = 0$ means that the two nodes are synchronized, and $\varphi = 0.5$ means that the nodes are dephased by exactly half a Trickle interval.

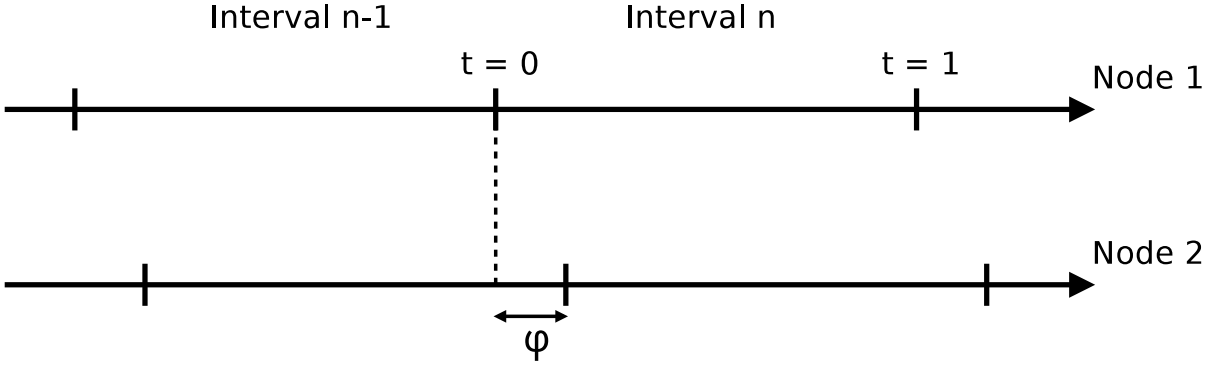

Figure 2: Definition of the phase φ between the two nodes.

[2]Notice that in Trickle's behavior, $k = 1$ corresponds to the minimum broadcast propagation case. Thus this is the best case to show unfairness. On the opposite, if k is very large, all the nodes always propagate broadcasts and there is no fairness issue nor any communication efficiency.

We model the behavior of each node as a *discrete-time Markov chain* with two states. For each Trickle interval n, where n is modeled as the discrete time of the Markov chain, the node either transmits (state T) or suppresses its transmission (state $\overline{T}$). We denote by T_n the event $<$*The node transmits during interval* $n>$ and $\overline{T}_n$ its opposite. The transitions of this Markov chain are the probabilities $\mathbb{P}(T_n|T_{n-1})$, $\mathbb{P}(T_n|\overline{T}_{n-1})$, $\mathbb{P}(\overline{T}_n|T_{n-1})$ and $\mathbb{P}(\overline{T}_n|\overline{T}_{n-1})$ for each node (see Figure 3).

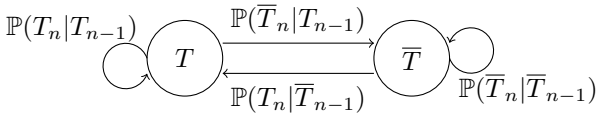

Figure 3: Two-nodes Markov chain model with $k = 1$.

Computation of these probabilities is detailed in Section 4.2 and [12]. Figure 4 summarizes the results with a node transmission probability plotted as a function of the phase φ. Notice that this transmission probability function is periodic, since both $\varphi = 0$ and $\varphi = 1$ actually describe the same situation: the two nodes have synchronized intervals.

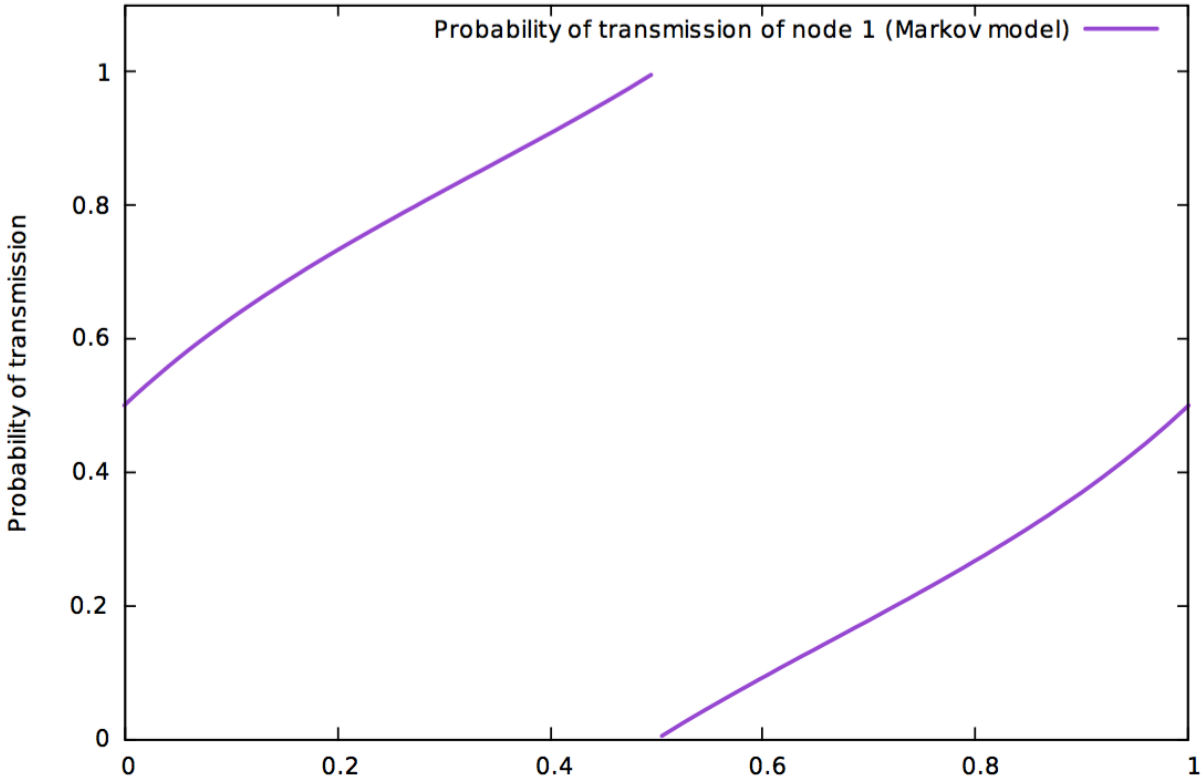

Figure 4: Node 1 transmission probability in the two-nodes Markov model with $k = 1$ as a function of the desynchronisation phase φ.

A truly noteworthy feature is the *non-continuity* of this function at $\varphi = 0.5$. Indeed, the case $\varphi = 0.5$ is special. The Markov chain becomes reducible, *i.e.,* each state ($\overline{T}$ or T) only has one transition, leading to itself. Practically, the first node to transmit will always suppress the other node's transmission, and will keep on transmitting forever. This is because the listen-only period of each node exactly matches the transmission period of the other node. This effect is analyzed in more depth in [18].

The case $\varphi = 0$ is straightforward. The nodes are synchronized, and the two nodes are perfectly fair to each other. This is because the listen-only period and transmission period do not overlap. For all other values of φ, one node has a clear advantage. The fundamental reason lies in the choice of the transmission time. The node starting its interval sooner (for instance, "node 1" in Figure 2) will have a

much higher probability of selecting an earlier time of transmission. The dependency of this probability with φ is quadratic, as computed in the Section 4.2. On the other hand, the node starting its interval sooner may suppress its transmission because of a transmission from the *previous* interval. However, the probability of this event is affine in φ, and is outweighed by the other quadratic factor (see Section 4.2).

4.2 Markov chain probabilities' computation

This section details the analytical model of the two-nodes network, with $k = 1$. Each node is modelled as a discrete-time Markov chain, for which the probabilities of transition are derived as a function of the phase φ (see Section 4.1).

4.2.1 Notations

We assume that the Trickle interval is of unit length. We further assume $0 \leq \varphi \leq 0.5$, because the situation of the two nodes is symmetric. $\varphi = 0$ means that the two nodes are synchronised, and $\varphi = 0.5$ means that the nodes are dephased by exactly half a Trickle interval.

For a node $i \in \{1, 2\}$ and an interval n, we denote by t_i^n the time of transmission chosen by node i for the nth interval. Note that t_i^n is *relative to the start of interval of node i*. We denote by $t_{i,abs}^n$ the absolute time of transmission, where the time reference is taken as the start of node 1's interval. Thus, we have the simple relations $t_{1,abs}^n = t_1^n$ and $t_{2,abs}^n = t_2^n + \varphi$. Similarly, we denote by $T_n^{(i)}$ the event $<$*Node i transmits during interval n*$>$, for $i \in \{1, 2\}$.

4.2.2 Events

We consider the following sets of complementary events, as illustrated in Figure 5:

1. $t_{2,abs}^n < t_{1,abs}^n$ and $t_{2,abs}^n \geq t_{1,abs}^n$, or equivalently $t_1^n - t_2^n > \varphi$ and $t_1^n - t_2^n \leq \varphi$. These events describe whether it is "node 1" or "node 2" that has chosen the earliest transmission time for a given interval. For instance, in Figure 5, "node 1" has chosen an earlier transmission time, meaning that the event $t_1^n - t_2^n \leq \varphi$ has occurred.

2. $t_2^{n-1} < 1 - \varphi$ and $t_2^{n-1} \geq 1 - \varphi$. The latter case is illustrated as the red event in Figure 5. That is, "node 2" transmits at interval $n - 1$, but the message is received during the nth interval of "node 1". Since $\varphi < 0.5$, "node 1" is still in its listen-only period, and cannot transmit before "node 2".

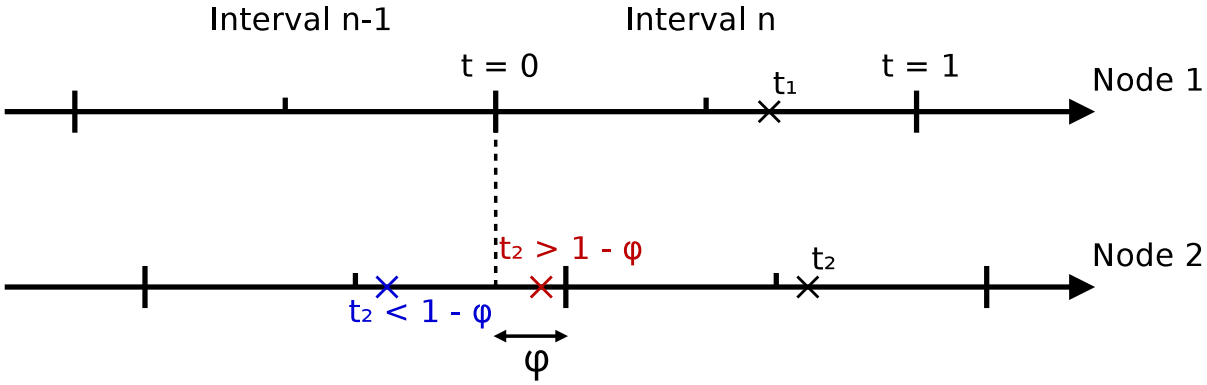

Figure 5: Principle and notations for the two-nodes modelisation with $k = 1$.

Since t_i is chosen uniformly between 0.5 and 1, it is easy to see that

$$\mathbb{P}(t_2 > 1 - \varphi) = 2\varphi \tag{1}$$

The first set of events is more complex. Since both t_1^n and t_2^n follow a uniform $\mathcal{U}([0.5, 1])$ distribution, their difference $X = t_1^n - t_2^n$ obeys the following probability density function:

$$f_X(x) = \begin{cases} 0 & \text{if } x \leq -0.5 \\ 4(x+0.5) & \text{if } -0.5 \leq x \leq 0 \\ 4(-x+0.5) & \text{if } 0 \leq x \leq 0.5 \\ 0 & \text{if } x \geq 0.5 \end{cases}$$

Thus, for $0 \leq \varphi \leq 0.5$, we can compute

$$\mathbb{P}(t_1^n - t_2^n \leq \varphi) \quad = \quad 0.5 + 2\varphi(1-\varphi) \tag{2}$$

4.2.3 Properties and theorems

Lemma During a given interval, exactly one of the two nodes transmits

Note that this property is not true in general. In this specific case (two nodes with $k = 1$), this property allows to simplify the analysis.

Proof. The crucial point is the following. Because $0 \leq \varphi \leq 0.5$ (see Figure 5), "node 2" can only hear *at most one transmission* from "node 1" during an interval. Indeed, during one interval of "node 2", "node 1" goes through the following sequence: a listen-only period (of length $1-\varphi$), then a transmission period (of length 0.5), then another listen-only period (of length φ).

With this fact in mind, consider that "node 1" transmits in interval n, at time t_1^n (see Figure 5). Then, necessarily, $t_{1,abs}^n < t_{2,abs}^n$. "node 2" thus hears a transmission in interval n, and suppresses its transmission.

Now assume that "node 1" suppresses its transmission in interval n (possibly, but not necessarily, because "node 2" transmitted in interval $n-1$). Then, because of the above property, "node 2" does not hear any transmission in interval n and thus transmits. □

Corollary $\forall n, T_n^{(1)} = \overline{T}_n^{(2)}$

That is, "node 1" transmits in an interval if and only if "node 2" does not transmit in the same interval.

Theorem of conditional total probability To compute the probabilities of transition of the Markov chain, we use this formula: $\forall A, B$ events, $\forall C_1, \dots C_n$ a partition of the probability space, we have $\mathbb{P}(A|B) = \sum_i \mathbb{P}(A|C_i \wedge B) \cdot \mathbb{P}(C_i|B)$.

This is simply the total probability theorem, restricted to a probability subspace by conditioning on a given event B.

4.2.4 Transition probabilities for node 1

We can now decompose $\mathbb{P}(T_n^{(1)}|T_{n-1}^{(1)})$, using the above complementary events and the theorem of conditional total probability. First, this quantity is equal to $\mathbb{P}(T_n^{(1)}|\overline{T}_{n-1}^{(2)})$, thanks to the lemma. Since we assume that "node 2" did not transmit during interval $n-1$, the only relevant event is the relative position of t_1^n and t_2^n. Node 1 transmits if it has selected an earlier transmission time than node 2 (*i.e.,* $t_1^n - t_2^n \leq \varphi$).

This gives the simple expression:

$$\begin{aligned} \mathbb{P}(T_n^{(1)}|T_{n-1}^{(1)}) &= \mathbb{P}(t_1^n - t_2^n \leq \varphi) && (3) \\ &= 0.5 + 2\varphi(1-\varphi) && (4) \end{aligned}$$

We now compute the other interesting quantity, $\mathbb{P}(T_n^{(1)}|\overline{T}_{n-1}^{(1)})$, *i.e.,* the probability that "node 1" transmits when it has not transmitted during the previous interval. Thanks to the lemma, this is again equal to $\mathbb{P}(T_n^{(1)}|T_{n-1}^{(2)})$.

In this case, "node 1" transmits during interval n if *both* of these independent events occur:

1. "node 1" has selected an earlier transmission time than node 2 (*i.e.,* $t_1^n - t_2^n \leq \varphi$), exactly as above

2. *and* during the previous interval $n-1$, "node 2" selected a transmission time before the start of node 1's nth interval (*i.e.,* $t_2^{n-1} \leq 1-\varphi$). This is the blue event in Figure 5.

This gives:

$$\begin{aligned} \mathbb{P}(T_n^{(1)}|\overline{T}_{n-1}^{(1)}) &= \mathbb{P}(t_1^n - t_2^n \leq \varphi) \cdot \mathbb{P}(t_2^{n-1} < 1-\varphi) && (5) \\ &= 0.5 + \varphi - 6\varphi^2 + 4\varphi^3 && (6) \end{aligned}$$

4.2.5 Markov chain

The transition matrix of the resulting Markov chain for node 1, $P^{(1)}$ is:

$$\begin{bmatrix} 0.5 - \varphi + 6\varphi^2 - 4\varphi^3 & 0.5 + \varphi - 6\varphi^2 + 4\varphi^3 \\ 0.5 - 2\varphi + 2\varphi^2 & 0.5 + 2\varphi - 2\varphi^2 \end{bmatrix}$$

Except for $\varphi = 0.5$, this Markov chain is irreducible, and thus admits a stationary distribution π^1 satisfying $\pi^1 = \pi^1 P^{(1)}$. Intuitively, it describes the time spent in each state, and thus gives access to the *transmission probability* of node 1 as its second component, which we denote $P_{tr}^{(1)}$. An eigenvector computation using Maxima [17] gives the solution:

$$P_{tr}^{(1)} = \frac{4\varphi^2 - 4\varphi - 1}{4\varphi^2 - 2\varphi - 2} \quad \text{for } 0 \leq \varphi < 0.5 \tag{7}$$

4.2.6 Symmetry

So far, we have only considered the case of node 1, with $0 \leq \varphi < 0.5$. Let use denote:

$$f(\varphi) = \frac{4\varphi^2 - 4\varphi - 1}{4\varphi^2 - 2\varphi - 2}$$

Now, consider the case where $0.5 < \varphi \leq 1$. We just have to exchange the roles of "node 1" and node 2, and consider the phase of node 1 with respect to node 2, $\varphi' = 1 - \varphi$. Since the situation of node 2 is exactly the same as the situation of "node 1" in the previous section, with $0 \leq \varphi' < 0.5$, we have:

$$P_{tr}^{(2)} = f(\varphi') \quad \text{for } 0 \leq \varphi' < 0.5$$

But since $P_{tr}^{(1)} = 1 - P_{tr}^{(2)}$ thanks to the lemma, this gives:

$$P_{tr}^{(1)} = 1 - f(\varphi') \quad \text{for } 0 \leq \varphi' < 0.5$$

This gives the final result:

$$\begin{aligned} P_{tr}^{(1)} &= 1 - f(1-\varphi) \quad \text{for } 0.5 < \varphi \leq 1 && (8) \\ &= \frac{1-2\varphi}{4\varphi^2 - 6\varphi} \quad \text{for } 0.5 < \varphi \leq 1 && (9) \end{aligned}$$

4.3 Topology unfairness

Given our hypotheses, it has been shown in [3] and [4] that *the transmission probability of nodes is not uniform across the network*. This behaviour is also mentioned as a possible issue in [15]: "*Trickle suppression typically leads sparser nodes to transmit more than denser ones*". That is, even when the redundancy constant k has the same value on all nodes, some nodes may transmit more than others. This is due to the arbitrary network topology density: a node with few neighbours tends to transmit more often than a node with many neighbours. In a dense network area, most nodes suppress their transmission. On the opposite, an isolated node systematically transmits.

However, note that the transmission probability of a node is not only dependent on the *number* of its neighbours, but also on their own transmission probability, as shown in [4].

4.4 Consequences on energy consumption

While section 4.1 models a very specific topology, it can help to extrapolate general trends. First of all, desynchronization between nodes leads to unfairness, partly because of the listen-only period, but mostly because of Trickle's usage of intervals. If the phase between nodes is uniformly distributed (which is a reasonable assumption for real networks), then, *on average* over the possible phases, each node will have a fair transmission rate. However, the phase is determined by the initial startup time of each device, thus, under a steady-state assumption, it does not change afterwards[3]. Thus, a network can potentially stay in an unfair state for a large amount of time. In more complex topologies and for other values of k, desynchronization will still cause unfairness, although unfairness is expected to greatly decrease for high values of k at the cost of high transmission load. In both unfairness cases described in Sections 4.1 and 4.3, the result is similar: the transmission load is distributed unevenly among nodes.

As a consequence some nodes transmit more often and thus consumes more energy. This is an issue for battery-powered networks, see for instance in [5] and [22], since "talkative" nodes will deplete their battery faster and die. The network will then suffer partial breakdown, leading to instabilities, repeated energy-consuming reconfiguration phases, and eventually network partitioning.

Desynchronization φ depends on clock drift and initial nodes startup, so its control would require running synchronization protocol among nodes for Trickle's purposes, which would defeat the original design with listen-only period. Thus we propose to adjust the k parameter in order to provide steady-state fairness with as low as possible broadcast propagations.

5 Trickle's fairness improvement

We aim at improving Trickle's fairness while keeping its general structure and preserving its properties: broadcast suppression and low overhead. In practice, improving fairness means that all battery-powered nodes will deplete their battery at roughly the same rate and that the lifetime of the whole network will be extended. Considering the usage of Trickle with RPL, the application goal is to establish network routes. Our analysis only takes into account control messages sent and received using Trickle. This is obviously an approximation because application messages also require energy to be transmitted. However, application-generated packets will be transmitted anyway. Moreover, they are typically transmitted using unicast, while Trickle messages are using broadcast transmission, which is much more energy-consuming on typical MAC stacks. For instance, using ContikiMAC, a broadcast transmission consumes fifteen times more energy than a synchronized unicast transmission [5].

Our proposal in the following is to adapt k in order to improve global fairness. Whenever a node transmits, k is incremented or decremented as a function of the received messages count since the last transmission.

5.1 Fairness measure

Since the goal is to improve the fairness of Trickle, we need a proper measure to evaluate fairness. For this purpose, we use the well-known Jain's fairness index introduced in [11]. Given a sequence of N real numbers $S = (x_1, \dots x_N)$, its Jain index is defined as:

$$J(S) = \frac{(\sum_{i=1}^{N} x_i)^2}{N \sum_{i=1}^{N} x_i^2} \tag{10}$$

The x_i typically represent how a shared resource is distributed among users: each user i receives a share x_i. If all x_i are equal, then $J = 1$: this is a perfectly fair sharing of the resource. If k users equitably share the resource, and $n - k$ users receive nothing, then $J = \frac{k}{N}$. The Jain index has a number of desirable properties: it is normalized between 0 and 1, dimensionless, independent of the population size, and continuous with respect to the x_i.

For our networking purpose, x_i is node i transmissions number. Informally, the "resource" we consider is the communication channel[4]. If all nodes have the same number of transmissions, then J will be 1,

[3] The phase might change over time because of clock drift, but this is often negligible (a cycle every half year with +/- 30 ppm), especially because I_{max} is in the order of minutes.

[4] Although we indirectly focus on energy consumption balance.

indicating a perfectly fair network. The communication overhead is defined as the *message count*, that is, the total number of messages sent by a node. Thus, to save energy, the goal is to maintain a fair network, while keeping the overhead as low as possible[5]. To this end, we adopt a control approach.

5.2 Redundancy parameter control algorithm

An interesting metric for this problem is the *number of received messages between two transmissions*. This metric has been used to evaluate the fairness of the Distributed Coordination Function (DCF) of IEEE802.11 networks [2]. The main idea is the following: if a node transmits at the same rate as its n neighbours, then it should (on average) receive n messages between two of its transmissions.

Following this idea, in Figure 1, our *Trickle-d* algorithm adjusts the Trickle redundancy parameter[6] k based on the number of received messages between two transmissions of a node. This implements a control loop on the transmission rate, because if a node transmits more, its neighbors will suppress their broadcast more often and thus transmit less. In order to maximize J, the objective of the control loop is to receive exactly n messages between two transmissions, although the randomized nature of Trickle makes it hard to be exactly in this situation.

In order to implement this control improvement, the Trickle algorithm stays unchanged, except that the redundancy parameter k is modified over time. Note that k is only modified when a node transmits a message. When a broadcast is suppressed k remains unchanged. Additionally, we bound k using two parameters k_{min} and k_{max}. An operator can use these parameters to constrain the algorithm. For instance, to guarantee a certain level of redundancy, one may wish to set $k_{min} > 1$ and to ensure that energy consumption stays low, one may wish to set a relatively small value[7] for k_{max}, for instance $k_{max} = 8$. We also scale k proportionally to the "error" *i.e.,* the difference between the number of received messages since the last transmission and the number of neighbors (which may vary according to the network environment).

Data: k_{init}: initial redundancy parameter
Data: k_{min}: minimum redundancy parameter
Data: k_{max}: maximum redundancy parameter
Data: *scaling*: scaling constant when changing k
$n_{RX} = 0$ //received messages counter//;
$k = k_{init}$;
when receiving a packet
  n_{RX}++;
end
when sending a packet according to Trickle
  $k = k + (n_{RX}$ - neighbors#());
  $k = \min(k, k_{max})$;
  $k = \max(k, k_{min})$;
  $n_{RX} = 0$;
end

Algorithm 1: Algorithm *Trickle-d* on each node which dynamically adjusts the value of k.

5.3 Implementation notes

The *Trickle-d* algorithm is intentionally simple since it must be implemented on devices with very limited capabilities. Compared to the original Trickle algorithm, our modification requires a node to know its degree, *i.e.,* the number of its neighbors. This can be implemented on top of an IPv6 stack, using the built-in Neighbor Discovery mechanism. RPL, for instance, already uses Neighbor Discovery to populate a neighbor table. Thus, knowing the number of neighbors is mostly free in real life networking stacks.

[5]Notice that there are other possible goals. One might want to optimise coverage, for instance by satisfying constraints such as "Each node receives at least one message every t time units". This could be combined with the above goals, but we leave that for future work.

[6]Notice that we now refer to a *redundancy parameter* because this quantity is no longer constant.

[7]Notice that the k default value in RPL is ten.

6 Validation and Evaluation

In the following, we present tools used to evaluate our proposal and the main results of our findings.

Contiki [6] is a very lightweight operating system designed to run on constrained nodes, such as sensors and actuators. It features a highly optimized networking stack, including standard protocols such as IPv6, TCP, UDP, RPL and CoAP available under a Free Software license. We use Contiki for the evaluation, because it comes with an implementation of Trickle timers.

Cooja [21] is a simulator for the Contiki operating system with an easy to use GUI. It is designed to directly emulate binary code for several motes hardwares and to simulate several wireless networking propagation models. Thus, the same exact binary code, produced by the Contiki toolchain, can be used in the simulator and eventually implanted in a real device.

The FIT IoT-LAB testbed [1] is a scientific instrument for IoT research. It offers several large-scale experimental platforms populated with numerous up-to-date IoT devices that can be configured with existing IoT operating systems including Contiki. IoTlab proposes a remote access and configuration interface and experiments run remotely through a batch system.

The algorithm presented in Section 5 has been implemented in Contiki. The experimental evaluation was conducted using Cooja with a binary compiled for actual motes. Following these results we simplify the algorithm to derive *Trickle-D* and run real experiments on the IoT-LAB testbed.

6.1 Experiment goals

We measure the overhead of our approach because, for instance, systematically transmitting at each interval would achieve optimal fairness ($J = 1$), in a trivial way. However, this induces a very high message count, and thus a high energy consumption. Therefore, we define the *transmission load* as the sum of the total number of messages sent in the network, normalized by the number of Trickle intervals of the simulation times the number of nodes.

We use two different topologies to see the two factors identified in Section 4. The first topology is a complete graph, so that the only relevant cause of unfairness is desynchronization. The second topology is a random network of fifteen synchronized nodes, to expose the topology related cause of unfairness. Note that a real network would exhibit unfairness because of the two factors simultaneously. However, we assess the impact of each of the factors separately to gain a better understanding of their own contributions.

6.2 Complete graph with unsynchronized nodes

These experiments were conducted using a complete graph of five nodes, with each individual experiment using a random phase between nodes (unsynchronized case). We compare the original Trickle algorithm with our *Trickle-d* version. The results are presented in Figure 6, where we plot the transmission load against the Jain index for each set of parameters. Each ellipse is the result of twenty experiments with the exact same parameters, but a different random seed. The ellipse is centered on the average value (transmission load; Jain index), and the size of ellipse expresses the standard deviation across the twenty experiments. Our goal is to achieve a high fairness while keeping a low message count, so the better result is in the lower-right corner of the graph. The *Trickle-d* algorithm achieves a high fairness ($J \approx 1$) for all parameters, while keeping the transmission load to a low value ($0.55 < load < 0.8$) similar to the $k = 3$ constant case.

With the original Trickle algorithm, whenever k increases, both fairness and transmission load increase. This is expected, since for large k, nodes will always transmit. Also note that the variation in fairness is quite large for small k. This is the effect described in Section 4.1: when the nodes end up roughly synchronized, fairness is good, but whenever they are too desynchronized, some nodes will transmit much more than others. Since we take a random phase for twenty experiments, this introduces a large variation.

The two oblong vertical ellipses on the right correspond to our *Trickle-d* algorithm. We notice that the Jain index is very close to one, which indicates a perfectly fair network. The transmission load, on the other hand, does not increase too much. Roughly, transmission load stays as in the static case $k = 3$, but with a much higher fairness, which indicates that the algorithm improves the global control behavior.

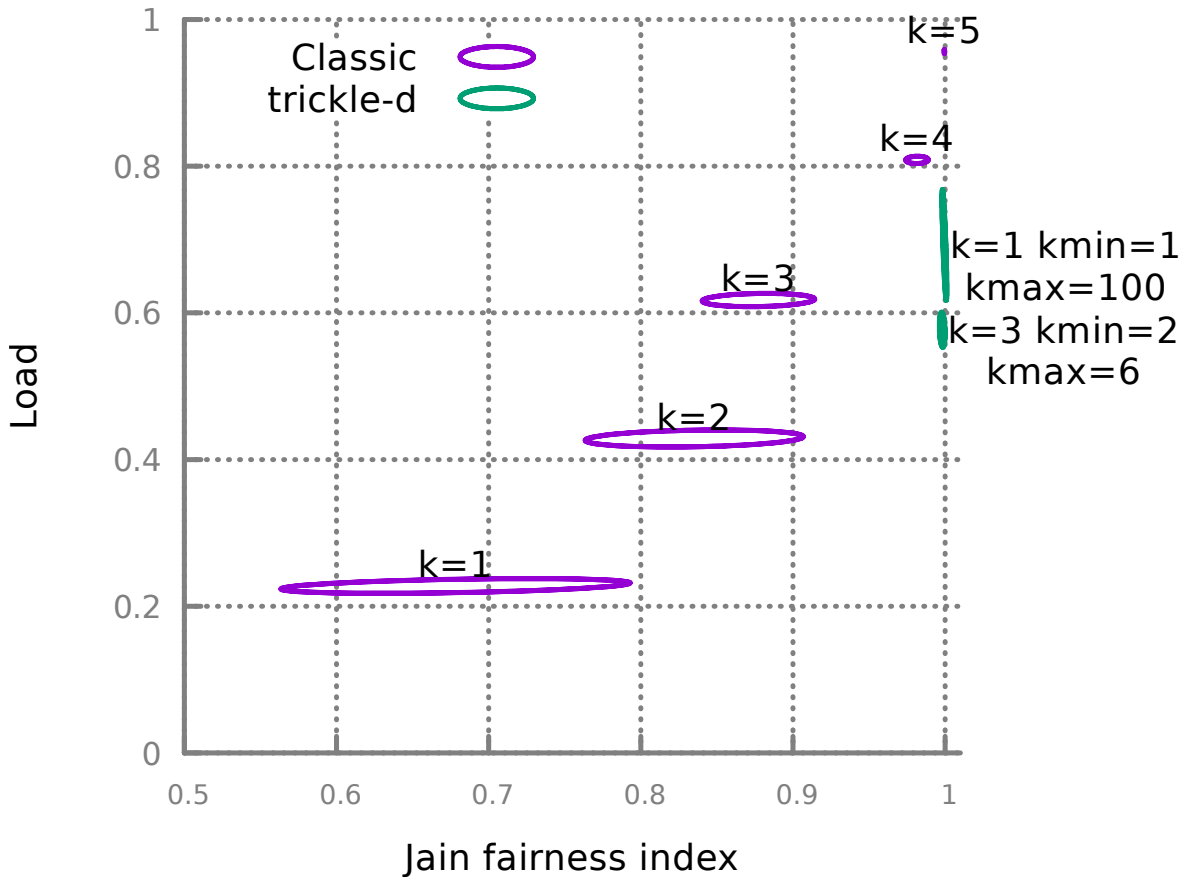

Figure 6: Trickle and improved algorithms comparison in an unsynchronized network of five nodes. Each axis represents a performance metric: Jain fairness against transmission load.

6.3 Random topology with synchronized nodes

The topology was obtained by randomly placing fifteen nodes in a square, where each node has a fixed radius of transmission. The resulting communication graph is shown in Figure 7. It features a cluster with high density, a second smaller clique, and some nodes with very low degree.

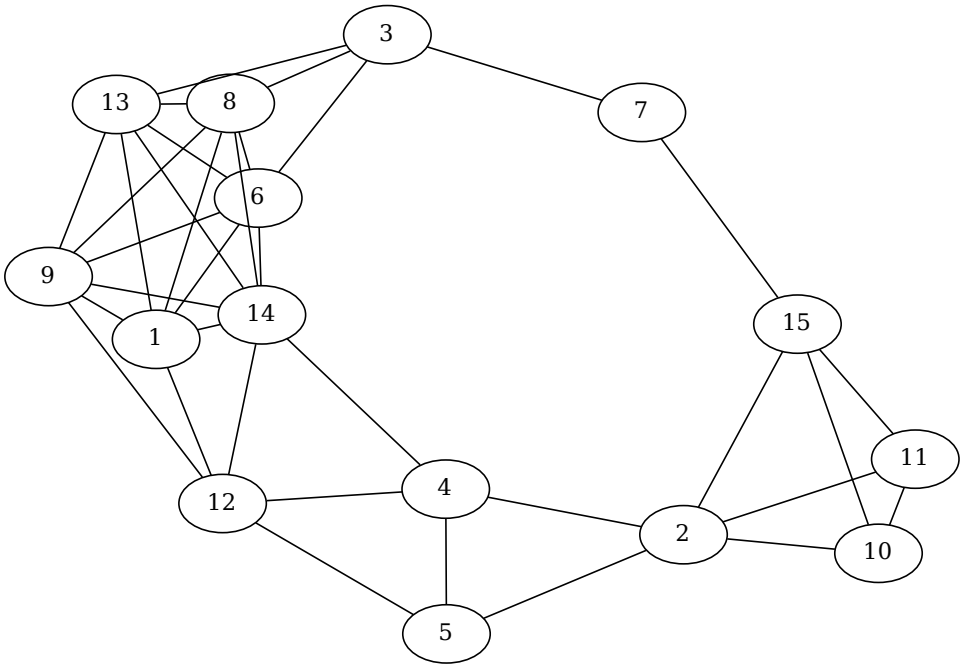

Figure 7: The random topology used for experiments.

On this topology, the same experiments described in Section 6.2 were run again in the constant case $k = 3$ which was the best performing. Figure 8 visually assesses the properties of the *Trickle-d* algorithm. The column height gives the overall transmission load (*i.e.,* the number of messages sent by each node during a single simulation) while colored rectangle surfaces depict each node transmission load. In the *Trickle-d* dynamic case on the right, the transmission load is visually more equally distributed among nodes, although the total number of messages is a little higher. However, from a lifetime point of view, the improved algorithm load balance leads to an important gain by avoiding early breakdowns, for instance in the constant case, using a bottom-up numbering, on nodes $1, 3, 4, 5, 7, 9, 10$ and 11.

Similarly to Section 6.2, Figure 9 plots ellipse representation of the transmission loads against the Jain index for this topology. The conclusions are the same, the *Trickle-d* algorithm achieves a high fairness, while its message count stays in between the constant cases $k = 2$ and $k = 4$.

6.4 Impact of the k_{init}, k_{min}, k_{max} parameters

The goal here is to provide guidelines on how to choose the parameters for a given application, depending on the desired redundancy and energy consumption. For this purpose, we study the transmission load for different sets of parameters, either in the five-nodes complete network (Figure 10) and in the fifteen-nodes random network (Figure 11).

k_{init} does not seem to influence at all (except a non-significative variation in the large k_{max} case of the complete graph where the confidence interval is also the largest). This is interesting because it means that our algorithm converges to the same results, whatever the initial conditions. This is also good news in practice, because one can imagine choosing k_{init} at random on each node.

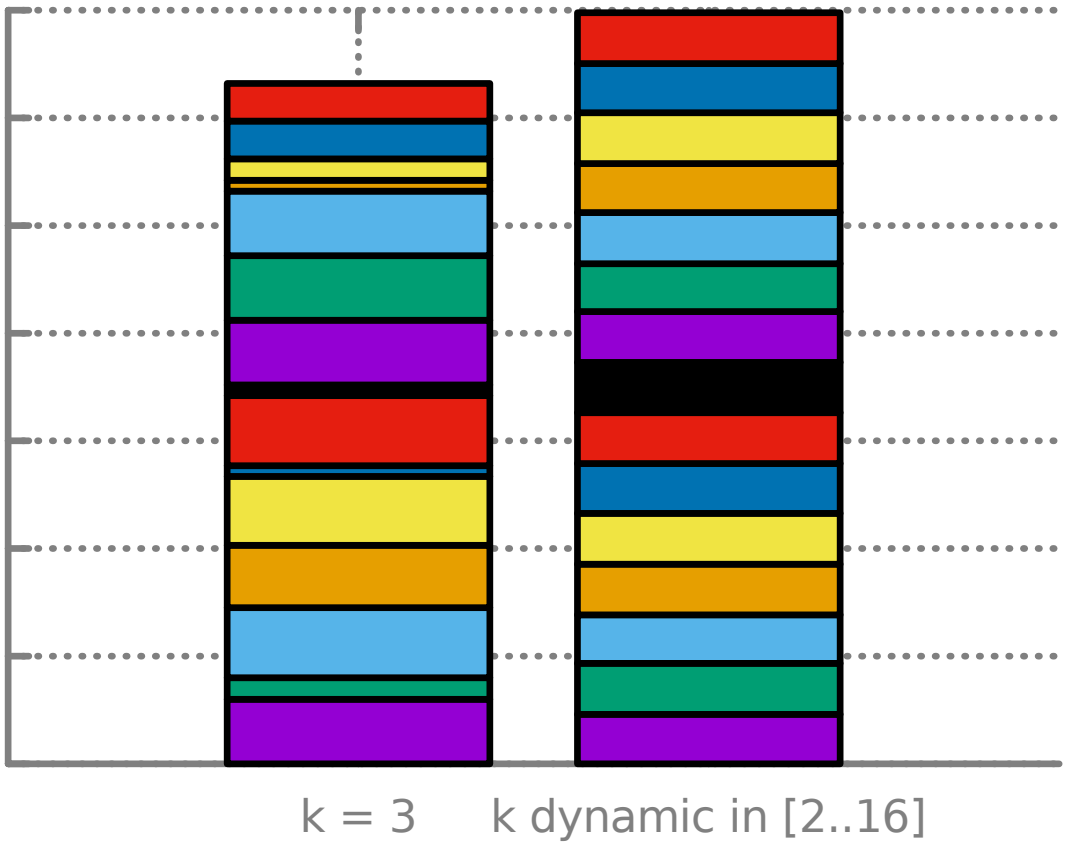

Figure 8: Typical message count per nodes repartition for $k = 3$ and k dynamically adjusted. Each node is represented by a color and the rectangle surface is proportional to its transmission load.

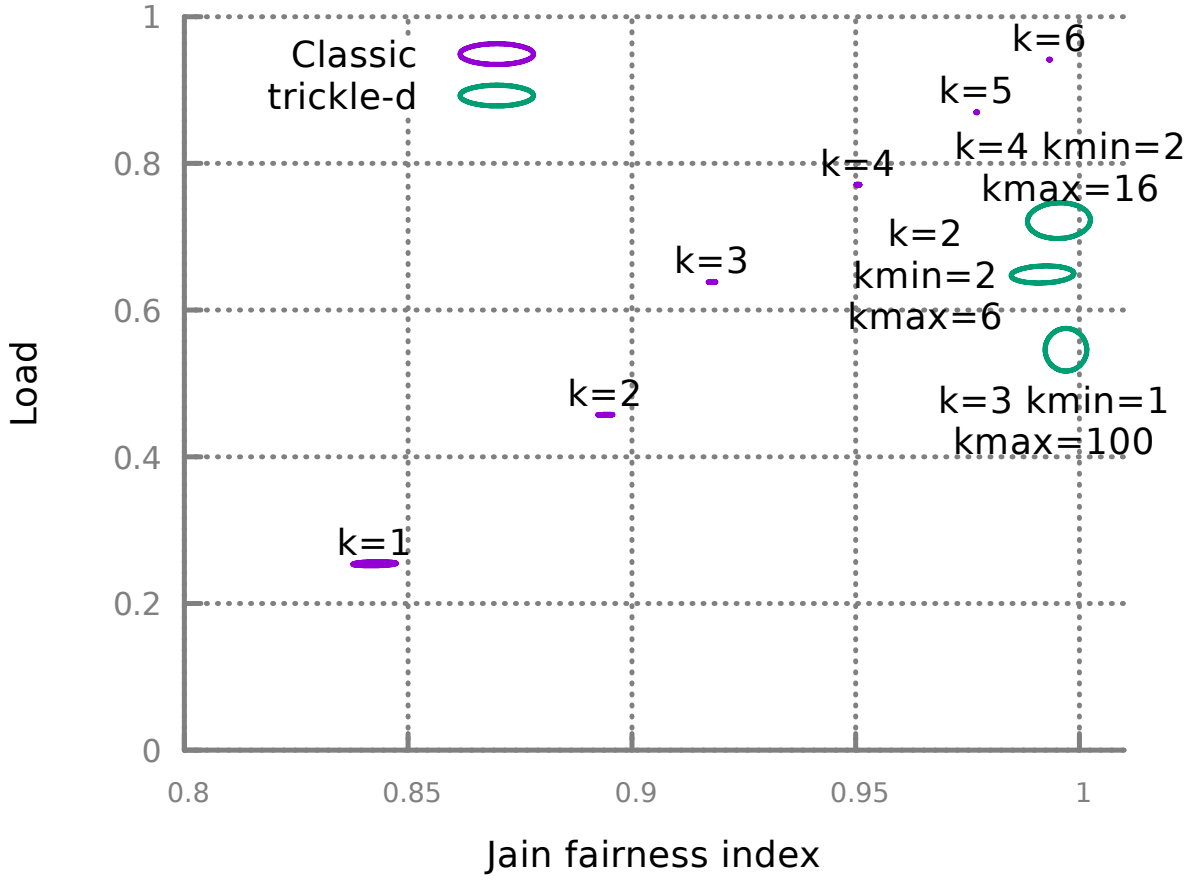

Figure 9: Trickle and *Trickle-d* algorithms comparison in a synchronized random network of fifteen nodes. Each axis represents a performance metric: Jain fairness against transmission load.

For both topologies, if k_{max} is large enough, decreasing k_{min} decreases the message count. This is an interesting information that tells k_{min} should be small to allow a lot of communications suppression during the execution. In practice one can imagine setting $k_{min} = 1$.

For both topologies, the message count increases with increasing k_{max} from 6 to 16. This is an expected behavior since a high value of k causes nodes to transmit more often. However, when decreasing k_{min}, increasing k_{max} seems not to be significant.

6.5 Trickle-D: Algorithm modification following the results

We introduce our result trends from Section 6.4 in the *Trickle-D* algorithm described in Algorithm 2. There is no longer application nor topology dependent parameters. The control loop is also applied when Trickle suppresses a transmission, assuring much smoother modification of the k value.

6.6 Markov model of the Trickle-D algorithm

In this section, we provide a discrete-time Markov chain model of Algorithm 2. Let the state of the chain $S = \{1, 2, \ldots, 16\}$ denote the current value of k. Let y denote the number of neighbors of node i. In order to find the transition probabilities of the chain, we compute the k increment that is equal to the difference between the number of received packets and the constant number of neighbors of a node. The probability to stay in the current state is equal to the probability that this difference is zero i.e., that node i received y packets during the interval. Hence, the chain will increment/decrement its next interval state value following computed difference. Therefore, we are interested in finding the

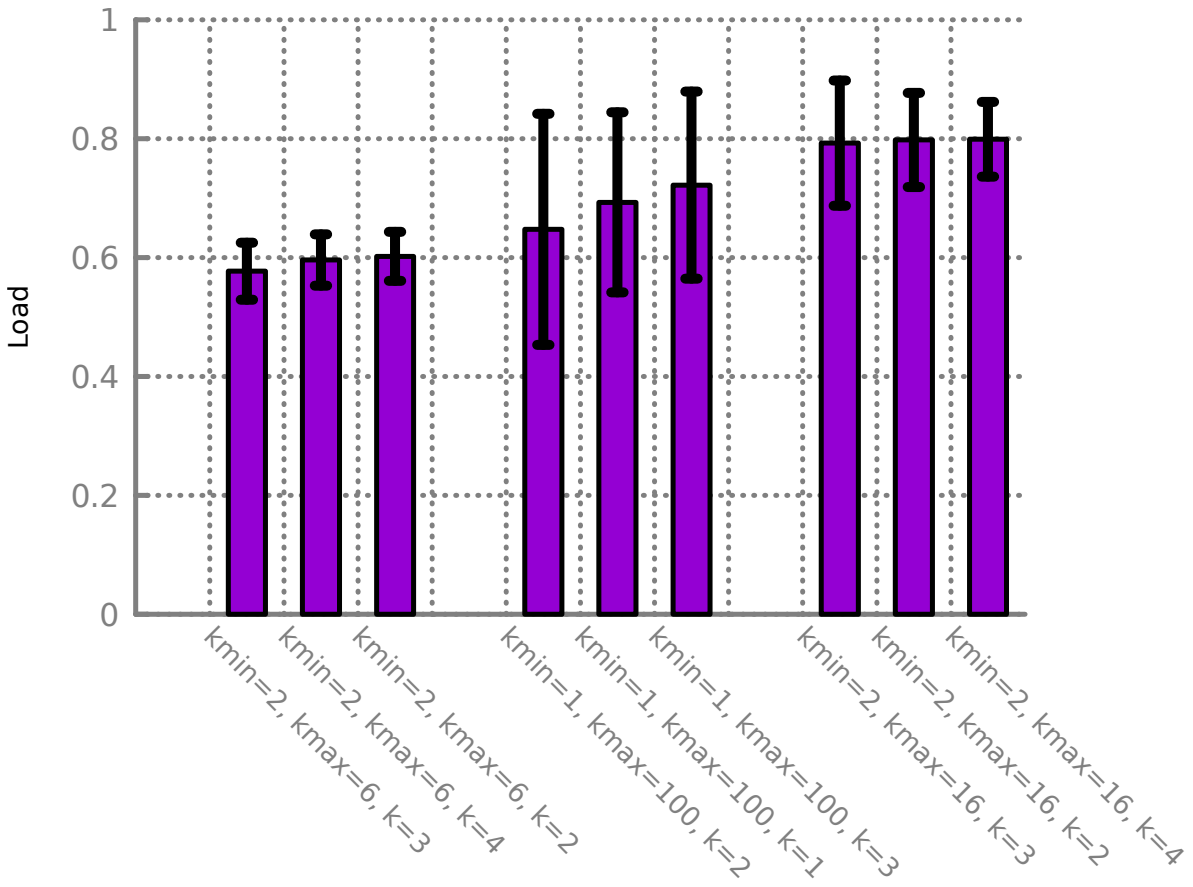

Figure 10: Influence of the parameters of the *Trickle-d* algorithm on the transmission load in a complete graph of five unsynchronized nodes.

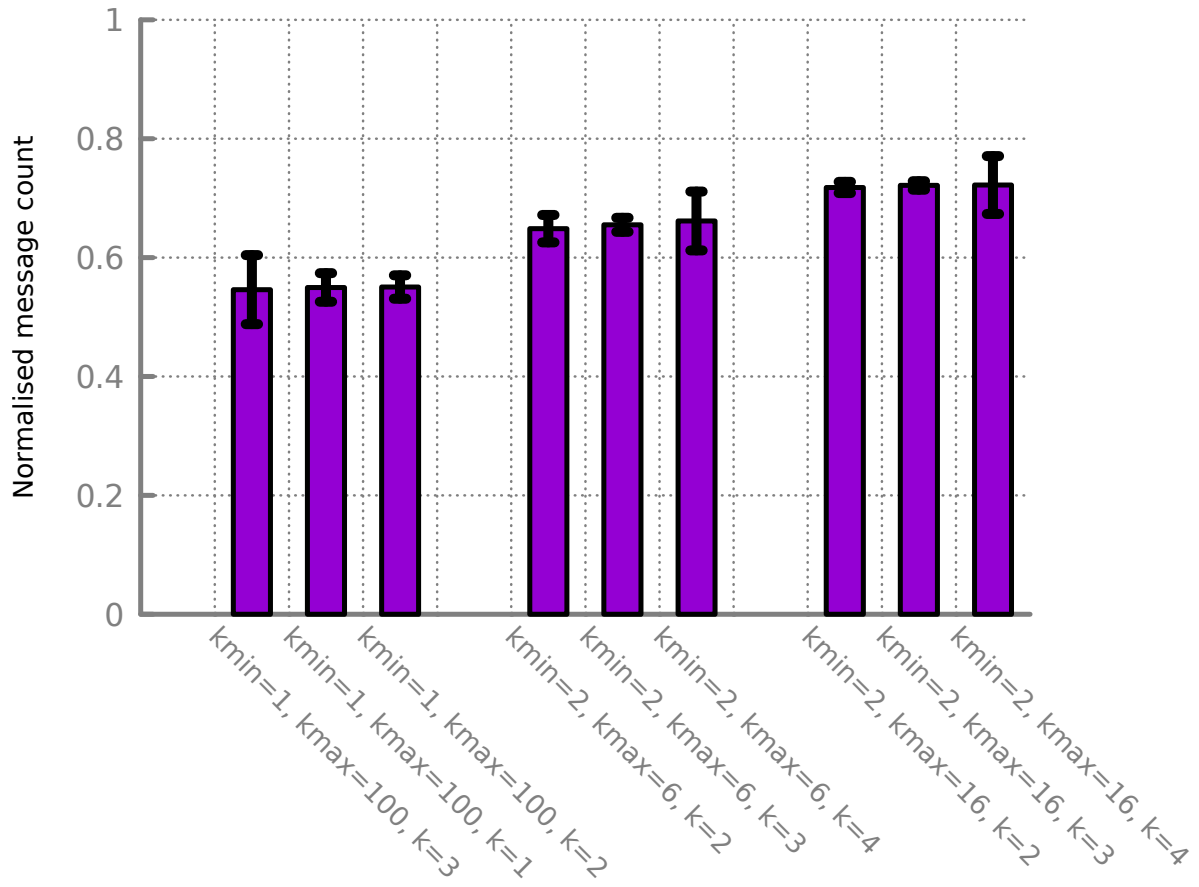

Figure 11: Influence of the parameters of the *Trickle-d* algorithm on the transmission load in a random graph of fifteen synchronized nodes.

$n_{RX} = 0$ //received messages counter//;
k = random()%16 //any value within results range//;
when receiving a packet
  n_{RX}++;
end
when sending a packet according to Trickle
  $n_{RX} = 0$;
  $k_{base} = k$;
end
when sending a packet or suppressing a transmission according to Trickle
  $k = k_{base}$ + (n_{RX} - neighbors#()) ;
  k = min(k, 16) //large enough value// ;
  k = max(k, 1) //smallest value// ;
end

Algorithm 2: Algorithm *Trickle-D* without any parameters. The k fixed range is determined from section 6.4 results.

probability that exactly j transmissions occur before the timer of node i expires, when the decision to transmit or suppress a transmission takes place. Let N_T be a random variable that denotes the number of transmissions that occurred. To find the probability $P(N_T = j)$, we resort to the model of Coladon *et al.* [4] that calculates the average probability of transmission for a given node in an arbitrary topology by setting up a system of N equations with N unknowns that can be numerically resolved, where N is the total number of nodes in a network.

However, the model of Coladon *et al.* [4] assumes that during a node's interval, each neighbor selects exactly one transmission time, which is only true in synchronized networks. We therefore extend the derivation in [4] in order to account for the fact that more than y packets can be received by a node with y neighbors. Consider a node with a single neighbor – depending on the phase φ, 0, 1 or 2 transmission times may occur during its interval. Assuming φ follows a uniform distribution in $[0, 1]$, we can compute the average probabilities $\widehat{P}_0$, $\widehat{P}_1$, $\widehat{P}_2$ that 0, 1, or 2 transmission times occur during the node's interval:

$$\widehat{P}_0 = \widehat{P}_2 = \int_0^1 \varphi(1-\varphi)\mathrm{d}\varphi = \frac{1}{6}, \tag{11}$$

and

$$\widehat{P}_1 = \int_0^1 \varphi^2 + (1-\varphi)^2 \,\mathrm{d}\varphi = \frac{2}{3}. \tag{12}$$

Consequently, if a node has y neighbors, the probability P, that l transmission times occur during its interval, can be computed. Let Y_N be a random variable that denotes the number of transmission times that occur during a node's interval and T be the set of different possibilities:

$$T = \{(x_1, ..., x_y) | x_i \in \{0, 1, 2\}, \forall 1 \leq i \leq y, \sum_{i=1}^{y} x_i = l\}$$

Probability $P(Y_N = l)$ is then:

$$P(Y_N = l) = \sum_{B \in T} \prod_{i=1}^{y} \widehat{P}_{B_i}, l \in [0, 2y], B_i \in \{0, 1, 2\}. \tag{13}$$

In order to derive the average probability of transmission of node i, $P_{tx}[i]$, taking into account that number of transmission times can be in $[0, 2y]$, we define the conditional probability $P_{tx}[i|Y_N = l]$. This probability can be calculated by replacing y_i with l in the derivation of $P_{tx}[i]$ in [4]. Then, $P_{tx}[i]$ can be found by conditioning on Y_N:

$$P_{tx}[i] = \sum_{l=0}^{2y} P_{tx}[i|Y_N = l] P(Y_N = l) \tag{14}$$

Similarly to [4], let T be the selected transmission time of node i, $T \in [\frac{1}{2}I,\ I]$. Then, let Y_T denote the number of selected transmission times before T. Y_T can be shown, as in [4], to follow a binomial distribution with parameters y and $\frac{T}{I}$. The probability that exactly m transmissions occur before the timer of node i expires can then be found to be:

$$P(N_T = m) = \sum_{l=m}^{2y} P(Y_N = l) \sum_{n=m}^{l} P(Y_T = n) \times \frac{1}{\binom{y}{n}} \sum_{B \in \Re} \gamma(m, n, B), \quad (15)$$

where $\gamma(m, n, B)$ is the probability that m nodes of the set $B = \{1, \ldots, n\}$ transmit before node i. From [4], it is calculated as:

$$\gamma(m, n, B) = \sum_{v=(i_1,\ldots,i_m) \in A} [\prod_{q=1}^{m} P_{tx}[v[q]] \times \prod_{\substack{l=1, \\ l \neq i_1,\ldots,i_m}}^{n} (1 - P_{tx}[l])].$$

For details on the calculation of $\gamma(m, n, B)$ and $P_{tx}[i]$, the reader is referred to [4]. Taking into account that the number of potentially received packets by a node is in $[0, 2y]$, the transition probabilities p_{ij} of the Markov chain are then:

$$p_{ij} = \begin{cases} \sum_{l=i-1}^{y} P(N_T = y - l), & (i-1) \leq y, j = 1 \\ P(N_T = y - (i-j)), |i-j| \leq y, j \neq 1, j \neq 16 \\ 1 - \sum_{j=1}^{15} p_{ij}, & \forall i, j = 16 \\ 0, & elsewhere, \end{cases}$$

where $i \in [1, 16]$ and $j \in [1, 16]$.

6.7 Evaluation on a testbed

In order to evaluate the performance of our *Trickle-D* algorithm in realistic conditions, we used several IoT-lab real platforms. We created three topologies using 15, 30 and 50 nodes[8]. For each node we modified the wireless network power range to get different numbers of neighbors. The topologies were created to have dense and sparse parts. Node degrees range from 1 to 22.

We compared our proposal against the "classic" version of the Trickle algorithm using several values of the k parameter, as well as against the broadcast suppression solution proposed by Meyfroyt *et al.* [20], in which case we vary the corresponding c parameter (see [20]). Each presented result is an average of five runs. We compare these results with our protocols: *Trickle-d* with several parameter sets and *Trickle-D*.

We present the fairness and transmission load results in Fig. 12. Both versions of our proposed algorithm, with and without parameters, achieve Jain fairness indexes above 0.99, while keeping the transmission load as low as the best of comparable other solutions. These experiments in real conditions show that the *Trickle-D* algorithm is agnostic of the evaluated topology with always very good fairness and low load, which is not the case with any of the other solutions. Compared to the state-of-the-art algorithm of Meyfroyt *et al.* [20] (case $\alpha = 0.5$), our solution emits 17.7% less messages while offering fairness index above 0.99. Moreover, compared to the constant k classical Trickle (case $k = 12$), our solution emits 37.2% less messages.

In order to analyse the behavior of the proposed solution with respect to node degrees, Fig. 13 shows average k value variations of five runs, as a function of the node degree, in experiments with a

[8] "M3 Open Nodes"[10], based on an ARM Cortex M3 micro-controller and an ATMEL radio.

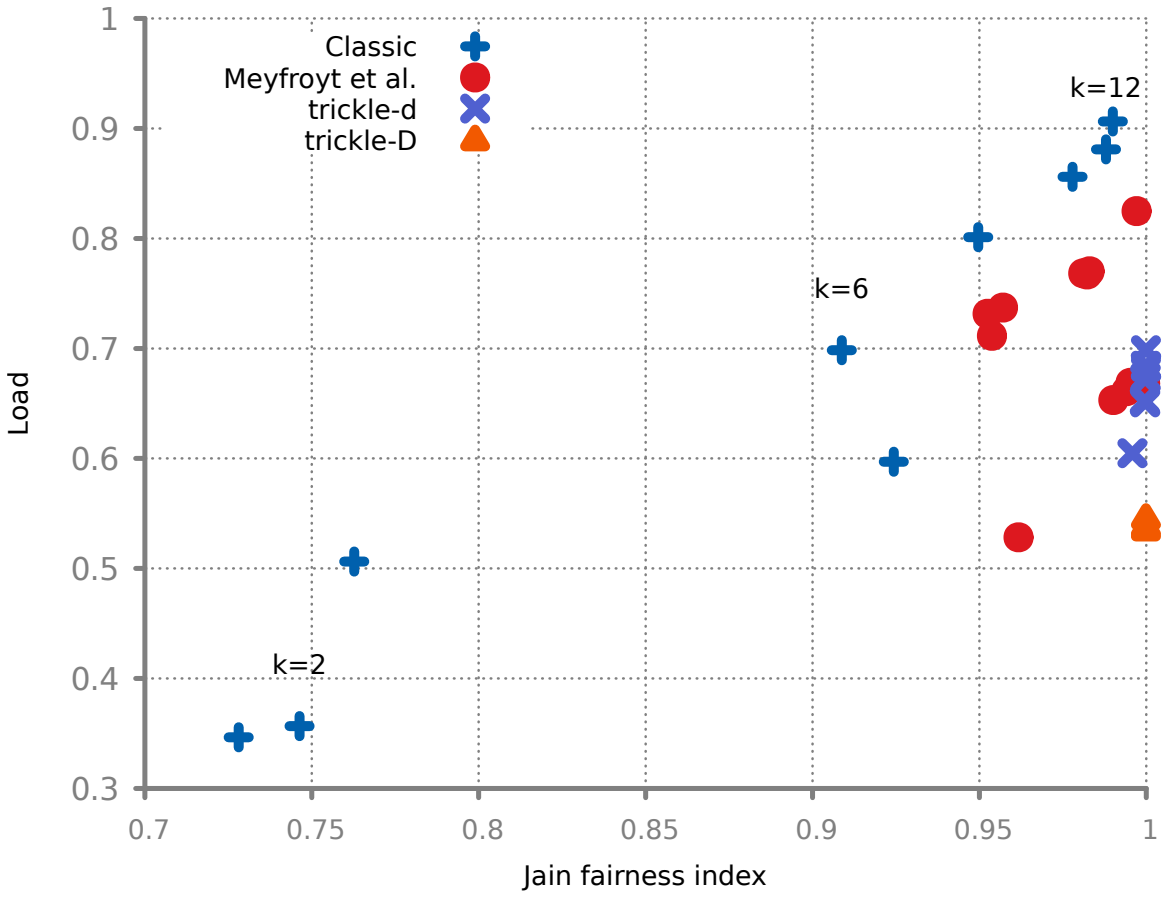

Figure 12: Fairness and load results for Trickle tests on three IoT-lab platforms with 15, 30 and 50 nodes. "Classic" shows several k values; "Meyfroyt" several α values; "Trickle-d" several parameter sets and "Trickle-D".

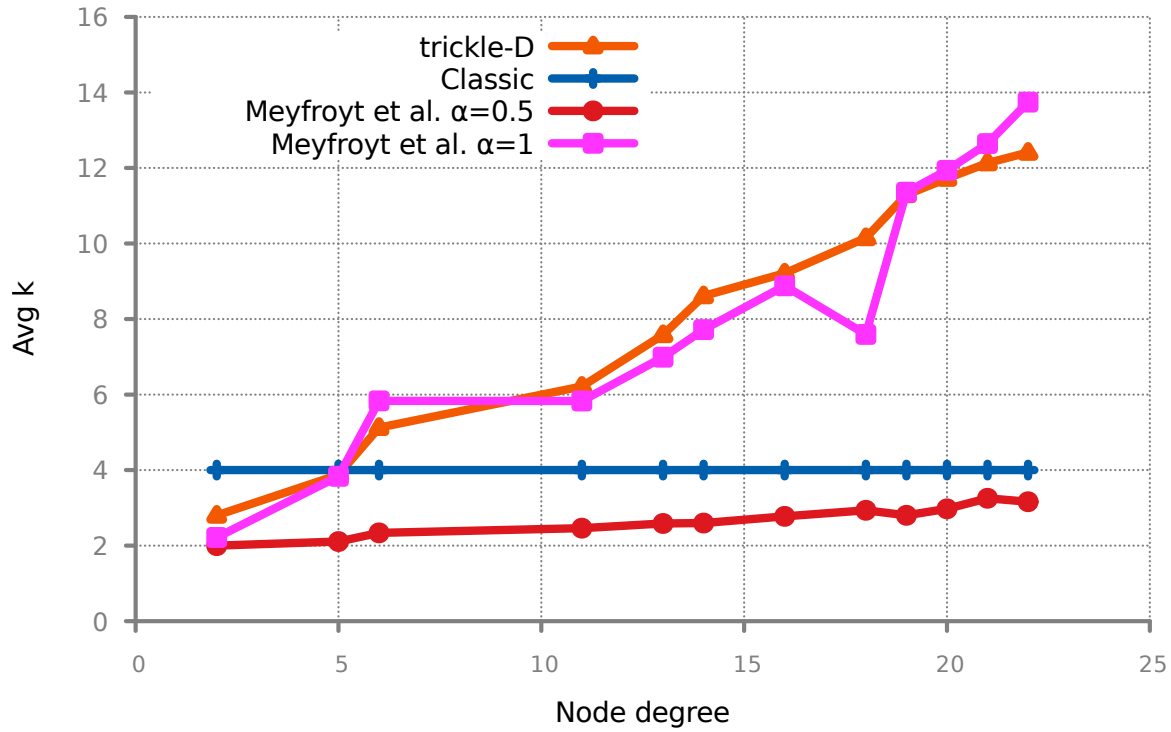

Figure 13: Average k value variation for nodes with different neighbors number in a topology composed of thirty nodes.

network of thirty nodes. "Classic" Trickle keeps the same value, fixed here with $k = 4$, for every node regardless of its degree. Both solutions proposed in [20] and our *Trickle-D* in Section 6.5 increase k as the node degree increases. The *Trickle-D* progression is similar to [20] with $\alpha = 1$. With $\alpha = 0.5$, [20] progresses very slowly. However, our *Trickle-D* solution that intrinsically takes into account both the number of received messages and the number of neighbors, adjusts the k value more smoothly. Moreover the regular k progression obtained with *Trickle-D* as well as its very good load balance does not need external knowledge nor any manual parameter adaptation. Note that all the experiments performed on the testbed account for unsynchronized nodes, as we do not control interval synchronization.

7 Conclusion

We analyzed the issue of unfairness in the Trickle algorithm, both analytically and experimentally. We introduced a control algorithm to improve fairness, that is based on received messages count and dynamically adapts the redundancy parameter k.

The algorithm has been implemented in Contiki, and evaluated on different experimental conditions in the Cooja emulator. Based on the feedback from the emulator, we derived a simplified version of the algorithm called *Trickle-D* that we evaluated in realistic conditions using a large scale experimental testbed.

The results show that *Trickle-D* achieves high fairness while keeping a low overhead compared with the original Trickle algorithm and state-of-the-art related work. Thus, the energy consumption can be balanced and the network lifetime maximized.

Our proposed algorithm acts on the redundancy parameter k but there are other parameters in Trickle that could be used to change its behavior. Further work will aim at the function of deciding whether to transmit or not. It is based on the number c of received messages acting as a threshold: transmit when $c < k$, do not transmit when $c \geq k$. We will investigate node transmission with a given probability dependent on c and k. For instance, this probability could be linear with respect to $(k - c)$ and could also take into account different node priorities. Thus, even in a dense network, each node would still have transmitting possibilities.

Last, our results indicate long-term tendencies, but say nothing about the control impact on k convergence speed, cyclic behavior or potential instabilities. In particular, most wireless networks are dynamic, either because the nodes themselves move, or because the radio propagation conditions change over time. In this case, convergence speed and dynamic matters and our dynamic approach may lead to even much better improvements against the static ones.

References

[1] Cedric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibolletand, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, and Thomas Watteyne. FIT IoT-LAB: A Large Scale Open Experimental IoT Testbed. In *IEEE World Forum on Internet of Things (WF-IoT)*, Milan, IT, Dec 2015.

[2] Gilles Berger Sabbatel, Andrzej Duda, Olivier Gaudoin, Martin Heusse, and Franck Rousseau. Fairness and its Impact on Delay in 802.11 Networks. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, volume 5, pages 2967–2973, Dallas, United States, 2004.

[3] Titouan Coladon. Modeling the Trickle Algorithm and Increasing its Fairness with Variable Redundancy Constants. Master's thesis, Grenoble Alps University, June 2014.

[4] Titouan Coladon, Mališa Vučinić, and Bernard Tourancheau. Multiple Redundancy Constants with Trickle. In *IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1951–1956, 2015.

[5] Adam Dunkels. The ContikiMAC Radio Duty Cycling Protocol. Technical report, Swedish Institute of Computer Science, 2011.

[6] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *IEEE International Conference on Local Computer Networks (LCN)*, pages 455–462, 2004.

[7] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection Tree Protocol. In *ACM SenSys*, pages 1–14, New York, NY, USA, 2009.

[8] Jonathan W. Hui and David Culler. The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale. In *ACM SenSys*, pages 81–94, New York, NY, USA, 2004.

[9] Jonathan W. Hui and Richard Kelsey. Multicast Protocol for Low-Power and Lossy Networks (MPL). RFC 7731 (Standards Track), February 2014.

[10] FIT IoT-lab. M3 open node - data sheet.

[11] Raj Jain, Dah-Ming Chiu, and William Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems. Technical Report 301, Digital Equipment Corporation, Eastern Research Lab, Hudson, MA 01749, USA, Sep 1998.

[12] Baptiste Jonglez. Improving the Fairness of the Trickle Algorithm. Master's thesis, Ecole Normale Supérieure de Lyon, June 2015.

[13] Hamidreza Kermajani, Carles Gomez, and Mostafa Hesami Arshad. Modeling the Message Count of the Trickle Algorithm in a Steady-State, Static Wireless Sensor Network. *IEEE Communications Letters*, 16(12):1960–1963, 2012.

[14] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In *Proceedings of the 1st USENIX/ACM Symp. on Networked Systems Design and Implementation*, 2004.

[15] Philip A. Levis, Thomas Clausen, Jonathan W. Hui, Omprakash Gnawali, and JeongGil Ko. The Trickle Algorithm. RFC 6206 (Proposed Standard), March 2011.

[16] Yi Liu, Chau Yuen, Xianghui Cao, Naveed Ul Hassan, and Jiming Chen. Design of a Scalable Hybrid MAC Protocol for Heterogeneous M2M Networks. *IEEE Internet of Things Journal*, 1(1):99–111, 2014.

[17] Maxima. Maxima, a computer algebra system. version 5.34.1, 2014.

[18] Thomas Meyfroyt. Modeling and analyzing the Trickle algorithm. *Master's Thesis, Eindhoven University of Technology, The Netherlands*, 2013.

[19] Thomas Meyfroyt, Sem Borst, Onno Boxma, and Dee Denteneer. On the scalability and message count of Trickle-based broadcasting schemes. *Queueing Systems*, pages 1–28, 2015.

[20] Thomas Meyfroyt, Milosh Stolikj, and Johan Lukkien. Adaptive broadcast suppression for Trickle-based protocols. In *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–9, June 2015.

[21] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-Level Sensor Network Simulation with COOJA. In *IEEE Conference on Local Computer Networks (LCN)*, pages 641–648, 2006.

[22] Leila Ben Saad and Bernard Tourancheau. Sinks Mobility Strategy in IPv6-Based WSNs for Network Lifetime Improvement. In *IEEE-IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, 2011.

[23] Carlo Vallati and Enzo Mingozzi. Trickle-F: Fair broadcast suppression to improve energy-efficient route formation with the RPL routing protocol. In *IEEE Sustainable Internet and ICT for Sustainability (SustainIT)*, pages 1–9, 2013.

[24] JP. Vasseur. Terms Used in Routing for Low-Power and Lossy Networks. RFC 7102 (Informational), January 2014.

[25] Tim Winter, Pascal Thubert, Anders Brandt, Jonathan W. Hui, Richard Kelsey, Philip A. Levis, Kris Pister, Rene Struik, JP. Vasseur, and Roger K. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550 (Proposed Standard), March 2012.

[26] Yang Yu, Loren J. Rittle, Vartika Bhandari, and Jason B. LeBrun. Supporting Concurrent Applications in Wireless Sensor Networks. In *ACM SenSys*, pages 139–152, New York, NY, USA, 2006.