Performance Evaluation 85–86 (2015) 19–32

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

A data propagation model for wireless gossiping

T.M.M. Meyfroyt [a,*], S.C. Borst [a], O.J. Boxma [a], D. Denteneer [b]

[a] Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
[b] Philips Research, HTC 34, 5656 AE Eindhoven, The Netherlands

ARTICLE INFO

Article history:
Received 23 July 2014
Received in revised form 22 October 2014
Accepted 4 January 2015
Available online 10 January 2015

Keywords:
Analytical model
Markov renewal process
Wireless communication
Gossip protocol
End-to-end delay
Trickle algorithm

ABSTRACT

Wireless sensor networks require communication protocols for efficiently propagating data in a distributed fashion. The Trickle algorithm is a popular protocol serving as the basis for many of the current standard communication protocols. In this paper we develop a mathematical model describing how Trickle propagates new data across a network consisting of nodes placed on a line. The model is analyzed and asymptotic results on the hop count and end-to-end delay distributions in terms of the Trickle parameters and network density are given. Additionally, we show that by only a small extension of the Trickle algorithm the expected end-to-end delay can be greatly decreased. Lastly, we demonstrate how one can derive the exact hop count and end-to-end delay distributions for small network sizes.

1. Introduction

In recent years wireless sensor networks have been quickly growing in popularity. In these networks inexpensive autonomous sensor units, called nodes, gather data, which they exchange with each other through wireless transmissions. Wireless sensor networks have various applications, for example in health care, area- and industrial monitoring [1].

Typically, these networks are communication, computation, memory and energy constrained and therefore require appropriate distributed communication protocols. The requirements for a good communication protocol are threefold. First, it should be able to quickly disseminate and collect data within the network. Second, this should be done as efficiently as possible, meaning that the number of transmissions in the network should be as low as possible. Third, communication should be reliable; new data should eventually be received by all the nodes in the network. Several communication protocols have been proposed in recent years for this purpose, see for example [2–6].

In [4] the Trickle algorithm has been proposed in order to effectively and efficiently distribute and maintain information in such networks. Trickle relies on a "polite gossip" policy to quickly propagate updates, while minimizing the number of redundant transmissions. Because of its broad applicability, Trickle has been documented in its own IETF RFC 6206 [7]. Moreover, it has been standardized as part of the IPv6 Routing Protocol for Low power and lossy networks [8] and the Multicast Protocol for Low power and lossy networks [9].

Because the Trickle algorithm has become a standard and is being widely used, it is crucial to understand how its parameters affect QoS measures like energy usage and end-to-end delay. However, not much work has yet been done in this

* Corresponding author.
E-mail addresses: t.m.m.meyfroyt@tue.nl (T.M.M. Meyfroyt), s.c.borst@tue.nl (S.C. Borst), o.j.boxma@tue.nl (O.J. Boxma), dee.denteneer@philips.com (D. Denteneer).

http://dx.doi.org/10.1016/j.peva.2015.01.001
0166-5316/

regard. Most of the papers that evaluate Trickle try to give guidelines on how to set the Trickle parameters using simulations [4,10–12].

The goal of this paper is to develop and analyze an analytical model describing how Trickle disseminates updates throughout a network. Our results serve as a first step towards understanding how Trickle's parameters influence its performance; however, the main focus of this paper will be on the mathematical analysis. Additionally, our models are relevant for the analysis of protocols that build upon Trickle, such as Deluge [13] and Melete [6].

1.1. Key contributions of the paper

As key contributions of this paper, we thoroughly analyze a Markov renewal process which models a Trickle propagation event in networks of nodes arranged on a line. We show how the hop count and end-to-end delay distributions depend on the Trickle parameters and network density, as the size of the network grows large. These insights help us to better understand the impact of Trickle's parameters on its performance and how to tune these parameters. Furthermore, we show that by a simple extension of the Trickle algorithm, the expected end-to-end delay can be significantly decreased. Finally, we show how to calculate the generating functions of the hop count and end-to-end delay distributions for any network size.

1.2. Related work

Some analytical results concerning the message count of the Trickle algorithm are provided in [12,14,15]. First, in [12] qualitative results are provided on the scalability of the algorithm. Specifically, it is conjectured that in single-hop networks the number of transmissions per time interval is bounded regardless of the network size, if a listen-only period is used. When no listen-only period is used, the message count scales as $\mathcal{O}(\sqrt{n})$, where n is the size of the network. These claims are proven in [15] and tight upper bounds and accompanying growth factors for the message count are provided. Additionally, distributions of times between consecutive transmissions for large single-hop networks are derived and the concept of a listen-only period is generalized. Moreover, approximations for the message count in multi-hop networks are provided. Lastly, in [14], a different, slightly more accurate, approximation is given for the message count in multi-hop networks with specific parameter settings.

Analytical results on the speed at which the Trickle algorithm can propagate new data throughout a network are even fewer. In [16] the authors provide a method for deriving the Laplace transform of the distribution function of the end-to-end delay for any network topology. However, the method is computationally involved, limiting its practical use, and does not provide much insight. In this paper we provide more easily computable expressions for the Laplace transforms of the hop count and end-to-end delay in a line network.

1.3. Organization of the paper

The remainder of this paper is organized as follows. In Section 2 we give a detailed description of the Trickle algorithm and propose a simple extension. We then analyze how fast the algorithm can propagate updates across a network consisting of nodes placed on a line in Section 3. Additionally, we derive the limiting distributions for the hop count and end-to-end delay as the size of the network grows large. These results are compared with simulations and we show that with our modified algorithm the expected end-to-end delay can be significantly decreased. This is followed in Section 4 by a derivation of the generating functions of the hop count and end-to-end delay for any network size. Finally, we present our conclusions in Section 5.

2. The Trickle algorithm

We now provide a detailed description of the original Trickle algorithm [4] before introducing a small extension, which helps improve the algorithm's performance. The Trickle algorithm has two main goals. First, whenever a new update enters the network, it must be propagated quickly. Secondly, when there are no updates, communication overhead has to be kept to a minimum.

The Trickle algorithm achieves this by using a "polite gossip" policy. Nodes divide time into intervals of varying length. During each interval a node will broadcast its current information, if it has not heard other nodes transmit the same information during that interval, in order to check if its information is up to date. If it has recently heard another node transmit the same information it currently has, it will stay quiet, assuming there is no new information to be received. Additionally, it will increase the length of its intervals, decreasing its broadcasting rate. Whenever a node receives an update or hears old information, it will reduce its interval size, increasing its broadcasting rate, in order to quickly resolve the inconsistency. This way inconsistencies are detected and resolved fast, while keeping the number of transmissions low.