Networking Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 20, 2011

P. Levis
Stanford University
T. Clausen
LIX, Ecole Polytechnique
J. Hui
Arch Rock Corporation
O. Gnawali
Stanford University
J. Ko
Johns Hopkins University

August 19, 2010

The Trickle Algorithm
draft-ietf-roll-trickle-04

Abstract

The Trickle algorithm allows nodes in a lossy shared medium (e.g., low power and lossy networks) to exchange information in a highly robust, energy efficient, simple, and scalable manner. Dynamically adjusting transmission windows allows Trickle to spread new information on the scale of link-layer transmission times while sending only a few messages per hour when information does not change. A simple suppression mechanism and transmission point selection allows Trickle's communication rate to scale logarithmically with density. This document describes the Trickle algorithm and considerations in its use.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on February 20, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the BSD License.

Table of Contents

1. Introduction

The Trickle algorithm establishes a density-aware local communication primitive with an underlying consistency model that guides when a node transmits. When a node's data does not agree with its neighbors, that node communicates quickly to resolve the inconsistency (e.g., in milliseconds). When nodes agree, they slow their communication rate exponentially, such that nodes send packets very infrequently (e.g., a few packets per hour). Instead of flooding a network with packets, the algorithm controls the send rate so each node hears a small trickle of packets, just enough to stay consistent. Furthermore, by relying only on local communication (e.g., broadcast or local multicast), Trickle handles network re-population, is robust to network transience, loss, and disconnection, is simple to implement, and requires very little state. Current implementations use 4-11 bytes of RAM and are 50-200 lines of C code[Levis08].

While Trickle was originally designed for reprogramming protocols (where the data is the code of the program being updated), experience has shown it to be a powerful mechanism that can be applied to wide range of protocol design problems, including control traffic timing, multicast propagation, and route discovery. This flexibility stems from being able to define, on a case-by-case basis, what constitutes "agreement" or an "inconsistency;" Section Section 6.8 presents a few examples of how the algorithm can be used.

This document describes the Trickle algorithm and provides guidelines for its use. It also states requirements for protocol specifications that use Trickle. This document does not provide results on Trickle's performance or behavior, nor does it explain the algorithm's design in detail: interested readers should refer to [Levis04] and [Levis08].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Additionally, this document introduces the following terminology:

Trickle communication rate: the sum of the number of messages sent or received by the Trickle algorithm in an interval.

3. Trickle Algorithm Overview

Trickle's basic primitive is simple: every so often, a node transmits data unless it hears a few other transmissions whose data suggest its own transmission is redundant. Examples of such data include routing state, software update versions, and the last heard multicast packet. This primitive allows Trickle to scale to thousand-fold variations in network density, quickly propagate updates, distribute transmission load evenly, be robust to transient disconnections, handle network repopulations, and impose a very low maintenance overhead: one example use, routing beacons in the CTP protocol [Gnawali09], requires sending on the order of a few packets per hour yet can respond in milliseconds.

Trickle sends all messages to a local communication address. The exact address used can depend on both the underlying IP protocol as well as how the higher layer protocol uses Trickle. In IPv6, for example, it can be the link-local multicast address or another local multicast address, while in IPv4 it can be the broadcast address (255.255.255.255).

There are two possible results to a Trickle message: either every node that hears the message finds its data is consistent with their own state, or a recipient detects an inconsistency. Detection can be the result of either an out-of-date node hearing something new, or an updated node hearing something old. As long as every node communicates somehow - either receives or transmits - some node will detect the need for an update.

For example, consider a simple case where "up to date" is defined by version numbers (e.g., network configuration). If node A transmits that it has version V, but B has version V+1, then B knows that A needs an update. Similarly, if B transmits that it has version V+1, A knows that it needs an update. If B broadcasts or multicasts updates, then all of its neighbors can receive them without having to advertise their need. Some of these recipients might not even have heard A's transmission.

In this example, it does not matter who first transmits, A or B; either case will detect the inconsistency. All that matters is that some nodes communicate with one another at some nonzero rate. As long as the network is connected and there is some minimum communication rate for each node, the network will reach eventual consistency.

The fact that Trickle communication can be either transmission or reception enables the Trickle algorithm to operate in sparse as well as dense networks. A single, disconnected node must transmit at the

---

Trickle communication rate. In a lossless, single-hop network of size n, the Trickle communication rate at each node equals the sum of the Trickle transmission rates across all nodes. The Trickle algorithm balances the load in such a scenario, as each node's Trickle transmission rate is 1/nth of the Trickle communication rate. Sparser networks require more transmissions per node, but utilization of the radio channel over space will not increase. This is an important property in wireless networks and other shared media, where the channel is a valuable shared resource. Additionally, reducing transmissions in dense networks conserves system energy.

4. Trickle Algorithm

This section describes the Trickle algorithm.

4.1. Parameters and Variables

A Trickle timer has three configuration parameters: the minimum interval size Imin, the maximum interval size Imax, and a redundancy constant k:

- The minimum interval size is defined in units of time (e.g., milliseconds, seconds). For example, a protocol might define the minimum interval as 100 milliseconds.

- The maximum interval size is described as a number of doublings of the minimum interval size (the base-2 log(max/min)). For example, a protocol might define the maximum interval as 16. If the minimum interval is 100ms, then the maximum interval is 100ms * 65536, 6,553.6 seconds, or approximately 109 minutes.

- The redundancy constant is a natural number (an integer greater than zero).

In addition to these three parameters, Trickle maintains three variables:

- I, the current interval size

- t, a time within the current interval, and

- c, a counter.

4.2. Algorithm Description

The Trickle algorithm has five rules:

1. When an interval begins, Trickle resets c to 0 and sets t to a random point in the interval, taken from the range [I/2, I), that is, values greater than or equal to I/2 and less than I. The interval ends at I.

2. Whenever Trickle hears a transmission that is "consistent," it increments the counter c.

3. At time t, Trickle transmits if and only if the counter c is less than the redundancy constant k.

4. When the interval I expires, Trickle doubles the interval length. If this new interval length would be longer than Imax, Trickle sets the interval length I to be Imax.

5. If Trickle hears a transmission that is "inconsistent," the Trickle timer resets. If I is greater than Imin, resetting a Trickle timer sets I to Imin and begins a new interval. If I is equal to Imin, resetting a Trickle timer does nothing. Trickle can also reset the timer in response to external "events."

The terms consistent, inconsistent and event are in quotes because their meaning depends on how a protocol uses Trickle.

5. Using Trickle

A protocol specification that uses Trickle MUST specify:

- Default values for Imin, Imax, and k. Because link layers can vary widely in their properties, the default value of Imin SHOULD be specified in terms of the worst-case latency of a link layer transmission. For example, a specification should say "the default value of Imin is 4 times the worst case link layer latency" and should not say "the default value of Imin is 500 milliseconds." Worst case latency is approximately time until the first link-layer transmission of the frame assuming an idle channel (does not include backoff, virtual carrier sense, etc.).

- What constitutes a "consistent" transmission.

- What constitutes an "inconsistent" transmission.

- What "events," if any, besides inconsistent transmissions that reset the Trickle timer.

6. Operational Considerations

It is RECOMMENDED that a protocol which uses Trickle include mechanisms to inform nodes of configuration parameters at runtime. However, it is not always possible to do so. In the cases where different nodes have different configuration parameters, Trickle may have unintended behaviors. This section outlines some of those behaviors and operational considerations as educational exercises.

6.1. Mismatched redundancy constants

If nodes do not agree on the redundancy constant k, then nodes with higher values of k will transmit more often than nodes with lower values of k. In some cases, this increased load can be independent of the density. For example, consider a network where all nodes but one have k=1, and this one node has k=2. The different node can end up transmitting on every interval: it is maintaining a Trickle communication rate of 2 with only itself. Hence, the danger of mismatched k values is uneven transmission load that can deplete the energy of some nodes.

6.2. Mismatched Imin

If nodes do not agree on Imin, then some nodes, on hearing inconsistent messages, will transmit sooner than others. These faster nodes will have their intervals grow to similar size as the slower nodes within a single slow interval time, but in that period may suppress the slower nodes. However, such suppression will end after the first slow interval, when the nodes generally agree on the interval size. Hence, mismatched Imin values are usually not a significant concern. Note that mismatched Imin values and matching Imax doubling constants will lead to mismatched Imax values.

6.3. Mismatched Imax

If nodes do not agree on Imax, then this can cause long-term problems with transmission load. Nodes with small Imax values will transmit faster, suppressing those with larger Imax values. The nodes with larger Imax values, always suppressed, will never transmit. In the base case, when the network is consistent, this can cause long-term inequities in energy cost.

6.4. Mismatched definitions

If nodes do not agree on what constitutes a consistent or inconsistent transmission, then Trickle may fail to operate properly. For example, if a receiver thinks a transmission is consistent, but the transmitter (if in the receivers situation) would have thought it inconsistent, then the receiver will not respond properly and inform the transmitter. This can lead the network to not reach a consistent state. For this reason, unlike the configuration constants k, Imin, and Imax, consistency definitions MUST be clearly stated in the protocol and SHOULD NOT be configured at runtime.

6.5. Specifying the constant k

There are some edge cases where a protocol may wish to use Trickle with its suppression disabled (k is set to infinity). In general, this approach is highly dangerous and it is NOT RECOMMENDED. Disabling suppression means that every node will always send on every interval, and can lead to congestion in dense networks. This approach is especially dangerous if many nodes reset their intervals at the same time. In general, it is much more desirable to set k to a high value (e.g., 5 or 10) than infinity. Typical values for k are 1-5: these achieve a good balance between redundancy and low cost[Levis08].

Nevertheless, there are situations where a protocol may wish to turn off Trickle suppression. Because k is a natural number (Section 4.1), k=0 has no useful meaning. If a protocol allows k to be dynamically configured, a value of 0 remains unused. For ease of debugging and packet inspection, having the parameter describe (k-1) can be confusing. Instead, it is RECOMMENDED that protocols which require turning off suppression reserve k=0 to mean k=infinity.

6.6. Relationship between k and Imin

Finally, a protocol SHOULD set k and Imin such that Imin is at least two to three as long as it takes to transmit k packets. Otherwise, if more than k nodes reset their intervals to Imin, the resulting communication will lead to congestion and significant packet loss. Experimental results have shown that packet losses from congestion reduce Trickle's efficiency [Levis04].

6.7. Tweaks and improvements to Trickle

Trickle is based on a small number of simple, tightly integrated mechanisms that are highly robust to challenging network environments. In our experiences using Trickle, attempts to tweak its behavior are typically not worth the cost. As written, the

algorithm is already highly efficient: further reductions in transmissions or response time come at the cost of failures in edge cases. Based on our experiences, we urge protocol designers to suppress the instinct to tweak or improve Trickle without a great deal of experimental evidence that the change does not violate its assumptions and break the algorithm in edge cases.

This warning in mind, Trickle is far from perfect. For example, Trickle suppression typically leads sparser nodes to transmit more than denser ones; it is far from the optimal computation of a minimum cover. However, in dynamic network environments such as wireless and low-power, lossy networks, the coordination needed to compute the optimal set of transmissions is typically much greater than the benefits it provides. One of the benefits of Trickle is that it is so simple to implement and requires so little state yet operates so efficiently. Efforts to improve it should be weighed against the cost of increased complexity.

6.8. Uses of Trickle

The Trickle algorithm has been used in a variety of protocols, both in operational as well as academic settings. Giving a brief overview of some of these uses provides useful examples of how and when it can be used. These examples should not be considered exhaustive.

Reliable flooding/dissemination: A protocol uses Trickle to periodically advertise the most recent data it has received, typically through a version number. An inconsistency is when a node hears a newer version number or receives new data. A consistency is when a node hears an older or equal version number. When hearing an older version number, rather than reset its own Trickle timer, it sends an update. Nodes with old version numbers that receive the update will then reset their own timers, leading to fast propagation of the new data. Examples of this use include multicast[I-D.hui-6man-trickle-mcast], network configuration[Lin08][Dang09], and installing new application programs[Hui04][Levis04].

Routing control traffic: A protocol uses Trickle to control when it sends beacons which contain routing state. An inconsistency is when the routing topology changes in a way that could lead to loops or significant stretch: examples include when the routing layer detects a routing loop or when a node's routing cost changes significantly. Consistency is when the routing topology is operating well and is delivering packets successfully. Using the Trickle algorithm in this way allows a routing protocol to react very quickly to problems (Imin is small) but send very few beacons when the topology is stable. Examples of this use include RPL[I-D.ietf-roll-rpl], CTP[Gnawali09],

and some current commericial IPv6 routing layers[Hui08].

7. Acknowledgements

The authors would like to acknowledge the guidance and input provided by the ROLL chairs, David Culler and JP Vasseur.

The authors would also like to acknowledge the helpful comments of Yoav Ben-Yehezkel, Alexandru Petrescu, and Urlich Herberg, which greatly improved the document.

8. IANA Considerations

This document has no IANA considerations.

9. Security Considerations

This document has no security considerations.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

10.2. Informative References

[Dang09] Dang, T., Bulusu, N., Feng, W., and S. Park, "DHV: A Code Consistency Maintenance Protocol for Multi-hop Wireless Networks", Wireless Sensor Networks: 6th European Conference Proceedings EWSN 2009 Cork, February 2009, <http://books.google.com/books?id=3fb5eePdkBgC&pg=PA327&lpg=PA327>.

[Gnawali09] Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., and P. Levis, "Collection Tree Protocol", Proceedings of the 7th ACM Conference on Embedded Networked Systems SenSys 2009, November 2009, <http://portal.acm.org/citation.cfm?id=1644038.1644040>.

[Hui04] Hui, J. and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale",

Proceedings of the 2nd ACM Conference on Embedded Networked Systems SenSys 2004, November 2004, <http://portal.acm.org/citation.cfm?id=1031506>.

[Hui08] Hui, J. and D. Culler, "IP is dead, long live IP for wireless sensor networks", Proceedings of the 6th ACM Conference on Embedded Networked Systems SenSys 2008, November 2008, <http://portal.acm.org/citation.cfm?id=1460412.1460415>.

[I-D.hui-6man-trickle-mcast] Hui, J. and R. Kelsey, "Multicast Forwarding Using Trickle", draft-hui-6man-trickle-mcast-00 (work in progress), July 2010.

[I-D.ietf-roll-rpl] Winter, T., Thubert, P., and R. Team, "RPL: IPv6 Routing Protocol for Low power and Lossy Networks", draft-ietf-roll-rpl-11 (work in progress), July 2010.

[Levis04] Levis, P., Patel, N., Culler, D., and S. Shenker, "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks"", Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation NSDI 2004, March 2004,

<http://portal.acm.org/citation.cfm?id=1251177>.

[Levis08] Levis, P., Brewer, E., Culler, D., Gay, D., Madden, S., Patel, N., Polastre, J., Shenker, S., Szewczyk, R., and A. Woo, "The Emergence of a Networking Primitive in Wireless Sensor Networks", Communications of the ACM, v.51 n.7, July 2008, <http://portal.acm.org/citation.cfm?id=1364804>.

[Lin08] Lin, K. and P. Levis, "Data Discovery and Dissemination with DIP", Proceedings of the 7th international conference on Information processing in sensor networks IPSN 2008, April 2008, <http://portal.acm.org/citation.cfm?id=1371607.1372753>.

Authors' Addresses

Philip Levis
Stanford University
358 Gates Hall, Stanford University
Stanford, CA 94305
USA

Phone: +1 650 725 9064
Email: pal@cs.stanford.edu

Thomas Heide Clausen
LIX, Ecole Polytechnique

Phone: +33 6 6058 9349
Email: T.Clausen@computer.org

Jonathan Hui
Arch Rock Corporation
501 Snd St., Suite 410
San Francisco, CA 94107
USA

Email: jhui@archrock.com

Omprakash Gnawali
Stanford University
S255 Clark Center, 318 Campus Drive
Stanford, CA 94305
USA

Phone: +1 650 725 6086
Email: gnawali@cs.stanford.edu

JeongGil Ko
Johns Hopkins University
3400 N. Charles St., 224 New Engineering Building
Baltimore, MD 21218
USA

Phone: +1 410 516 4312
Email: jgko@cs.jhu.edu