Internet Engineering Task Force (IETF)
Request for Comments: 6206
Category: Standards Track
ISSN: 2070-1721

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

March 2011

The Trickle Algorithm

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 allow 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 is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6206.

Copyright Notice

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 a 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 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 regarding 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.

Trickle transmission rate: the sum of the number of messages sent 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 re-populations, and impose a very low maintenance overhead: one example use, routing beacons in the Collection Tree Protocol (CTP) [Gnawali09], requires sending on the order of a few packets per hour, yet CTP can respond to topology changes in milliseconds.

Trickle sends all messages to a local communication address. The exact address used can depend on 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 that the message data is consistent with its 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 have even heard A’s transmission. In this example, it does not matter who first transmits -- A or B; the inconsistency will be detected in either case.

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 the utilization of a given broadcast domain (e.g., radio channel over space, shared medium) 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 runs for a defined interval and has three configuration parameters: the minimum interval size Imin, the maximum interval size Imax, and a redundancy constant k:

- The minimum interval size, Imin, 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, Imax, is described as a number of doublings of the minimum interval size (the base-2 log(max/min)). For example, a protocol might define Imax as 16. If the minimum interval is 100 ms, then the amount of time specified by Imax is 100 ms * 65,536, i.e., 6,553.6 seconds or approximately 109 minutes.

- The redundancy constant, k, 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 six rules:

1. When the algorithm starts execution, it sets I to a value in the range of [Imin, Imax] -- that is, greater than or equal to Imin and less than or equal to Imax. The algorithm then begins the first interval.

2. 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.

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

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

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

6. If Trickle hears a transmission that is "inconsistent" and I is greater than Imin, it resets the Trickle timer. To reset the timer, Trickle sets I to Imin and starts a new interval as in step 2. If I is equal to Imin when Trickle hears an "inconsistent" transmission, Trickle does nothing. Trickle can also reset its timer in response to external "events".

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

The only time the Trickle algorithm transmits is at step 4 of the above algorithm. This means there is an inherent delay between detecting an inconsistency (shrinking I to Imin) and responding to that inconsistency (transmitting at time t in the new interval). This is intentional. Immediately responding to detecting an inconsistency can cause a broadcast storm, where many nodes respond at once and in a synchronized fashion. By making responses follow the Trickle algorithm (with the minimal interval size), a protocol can benefit from Trickle’s suppression mechanism and scale across a huge range of node densities.

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 the 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 -- reset the Trickle timer.

- What information a node transmits in Trickle messages.

- What actions outside the algorithm the protocol takes, if any, when it detects an inconsistency.

6. Operational Considerations

It is RECOMMENDED that a protocol that 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 in a low-power network.

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 a size similar to that of 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 maximum interval lengths.

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 receiver’s 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; this 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 rather than k can be confusing. Instead, it is RECOMMENDED that protocols that 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 times 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.

With 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, 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 occurs when a node hears a newer version number or receives new data. A consistency occurs when a node hears an older or equal version number. When hearing an older version number, rather than reset its own Trickle timer, the node 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 [Hui08a], network configuration [Lin08] [Dang09], and installing new application programs [Hui04] [Levis04].

Routing control traffic: A protocol uses Trickle to control when it sends beacons that contain routing state. An inconsistency occurs 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 occurs 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 the IPv6 routing protocol for low-power and lossy networks (RPL) [RPL], CTP [Gnawali09], and some current commercial IPv6 routing layers [Hui08b].

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 Ulrich Herberg, which greatly improved the document.

8. Security Considerations

As it is an algorithm, Trickle itself does not have any specific security considerations. However, two security concerns can arise when Trickle is used in a protocol. The first is that an adversary can force nodes to send many more packets than needed by forcing Trickle timer resets. In low-power networks, this increase in traffic can harm system lifetime. The second concern is that an adversary can prevent nodes from reaching consistency.

Protocols can prevent adversarial Trickle resets by carefully selecting what can cause a reset and protecting these events and messages with proper security mechanisms. For example, if a node can reset nearby Trickle timers by sending a certain packet, this packet should be authenticated such that an adversary cannot forge one.

An adversary can possibly prevent nodes from reaching consistency by suppressing transmissions with "consistent" messages. For example, imagine node A detects an inconsistency and resets its Trickle timer. If an adversary can prevent A from sending messages that inform nearby nodes of the inconsistency in order to repair it, then A may remain inconsistent indefinitely. Depending on the security model of the network, authenticated messages or a transitive notion of consistency can prevent this problem. For example, let us suppose an adversary wishes to suppress A from notifying neighbors of an inconsistency. To do so, it must send messages that are consistent with A. These messages are by definition inconsistent with those of A’s neighbors. Correspondingly, an adversary cannot simultaneously prevent A from notifying neighbors and not notify the neighbors itself (recall that Trickle operates on shared, broadcast media). Note that this means Trickle should filter unicast messages.

9. References

9.1. Normative References

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

9.2. 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://portal.acm.org/citation.cfm?id=1506781>.

[Gnawali09] Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., and P. Levis, "Collection Tree Protocol", Proceedings of the 7th ACM Conference on Embedded Networked Sensor 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 Sensor Systems, SenSys 2004, November 2004, <http://portal.acm.org/citation.cfm?id=1031506>.

[Hui08a] Hui, J., "An Extended Internet Architecture for Low-Power Wireless Networks - Design and Implementation", UC Berkeley Technical Report EECS-2008-116, September 2008, <http://www.eecs.berkeley.edu/Pubs/>.

[Hui08b] Hui, J. and D. Culler, "IP is dead, long live IP for wireless sensor networks", Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems, SenSys 2008, November 2008, <http://portal.acm.org/citation.cfm?id=1460412.1460415>.

[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, Vol. 51 No. 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>.

[RPL] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Clausen, T., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., and JP. Vasseur, "RPL: IPv6 Routing Protocol for Low power and Lossy Networks", Work in Progress, March 2011.

Authors' Addresses

Philip Levis
Stanford University
358 Gates Hall
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 2nd 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