The Trickle Algorithm
draft-levis-roll-trickle-00

Abstract

   The Trickle algorithm allows wireless nodes 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 nechanism and transmission point
   selection allows Trickle's communication rate to scale
   logarithmically with density. This document describes Trickle and
   considerations in its use.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

The Trickle algorithm is designed for wireless networks. It
establishes a density-aware local broadcast with an underlying
consistency model that guides when a node communicates. When a
node's data does not agree with its neighbors, it communicates
quickly to resolve the inconsistency. When nodes agree, they slow
their communicationrate exponentially, such that in a stable state
nodes send at most 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 broadcasts,
Trickle handles network re-population, is robust to network
transience, loss, and disconnection, and requires very little state
(implementations use 4-11 bytes).

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. For example, routing protocols
such as RPL use Trickle to ensure that nodes in a given neighborhood
have consistent, loop-free routes. When the topology is consistent,
nodes occasionally gossip to check that they still agree, and when
the topology changes they gossip more frequently, until they reach
consistency again.

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
[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].

3. Trickle Algorithm Overview

Trickle's basic primitive is simple: every so often, a mote transmits
code metadata if it has not heard a few other motes transmit the same
thing. This 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 maintenance overhead on the order of a few packets per hour.

Trickle sends all messages to the local broadcast address. There are two possible results to a Trickle broadcast: either every mote that hears the message is up to date, or a recipient detects the need for an update. Detection can be the result of either an out-of-date mote hearing someone has new code, or an updated mote hearing someone has old code. As long as every mote communicates somehow - either receives or transmits - the need for an update will be detected.

For example, consider a simple case where "up to date" is defined by version numbers (e.g., network configuration). If node A broadcasts that it has version V, but B has version V+1, then B knows that A needs an update. Similarly, if B broadcasts that it has V+1, A knows that it needs an update. If B broadcasts 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 communication can be either transmission or reception enables Trickle to operate in sparse as well as dense networks. A single, disconnected node must transmit at the communication rate. In a lossless, single-hop network of size n, the sum of transmissions over the network is the communication rate, so for each node it is 1/n. Sparser networks require more transmissions per mote, but utilization of the radio channel over space will not increase. This is an important property in wireless networks, 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:

o 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.

o 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.

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

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

o I, the current interval size

o t, a time within the current interval, and

o 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).

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

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

4. When an interval 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" and I is
    greater than Imin, Trickle resets I to Imin, and begins a new
    interval.

The terms consistent and inconsistent are in quotes because their
meaning depends on the use of Trickle.

5. Using Trickle

   A protocol specification that uses Trickle should specify:

   o 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 the time until the first
      link-layer transmission of the frame assuming an idle channel
      (does not include backoff, virtual carrier sense, etc.).

   o What constitutes a "consistent" transmission.

   o What constitutes an "inconsistent" transmission.

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 as an educational exercise.

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 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.

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 will
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 should be clearly stated in the
protocol and should not be configured at runtime.

7. Acknowledgements

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

[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>.

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