Poster Abstract: Synchronizing Trickle Intervals

Joakim Eriksson
Swedish Institute of Computer Science
Kista, Stockholm, Sweden
joakime@sics.se

Omprakash Gnawali
Computer Science Department
University of Houston
gnawali@cs.uh.edu

***Abstract*—Routing protocols for low-power wireless networks, such as CTP and RPL, use explicit control traffic between nodes to find neighbors, construct and maintain routing paths. To conserve power and reduce network congestion, the control traffic frequency must be kept low. Reducing control traffic will not only reduce power consumption, but also affect the quality of the routing paths constructed by the routing protocols, in terms both of delivery ratio and power consumption. Several existing protocols use control traffic reduction mechanisms such as Trickle. Trickle uses two mechanisms to achieve efficient routing: (1) rate adaption where the nodes send control traffic less often when the network is stable, and (2) suppression where a node avoids sending control traffic if the information has already been recently sent by neighboring nodes. In this work, we present scenarios that cause these mechanisms to result in pathology in control packet timing. We also present a modification to Trickle to address the pathology.**

I. Introduction

Routing protocols for low-power wireless networks send control traffic for topology maintenance. Examples of such control traffic are neighbor reachability information [5] and routing beacons with path cost [1], [2]. Control traffic must reach all neighbors and is therefore sent using broadcast. In duty cycled low-power wireless networks, however, broadcast is significantly more power consuming and requires significantly more bandwidth than unicast transmissions.

Because of the expensive broadcast transmissions, routing protocols for low-power wireless face a fundamental trade-off: by sending more control traffic, the quality of the routing graph may be increased, but excessive control traffic significantly reduces network lifetime and also results in network congestion.

Many routing protocols use mechanisms such as Trickle [3] to minimize the number of control packets in the network while maintaining certain agility to network dynamics. Routing protocols such as CTP [2] and RPL [4], [5] use Trickle to time their control packets: decrease the control traffic interval when the routing cost change little or the network is not dynamic. They also suppress transmissions of control traffic when other nodes have already transmitted the same message.

The problem has an inherent complexity caused by the intricate interactions between the behavior of the radio medium, duty cycling mechanisms, the control traffic rate, and the data traffic. For example, although a routing protocol may create better routing paths by rapid dissemination of large number of control packets, the control traffic will cause a significant congestion that interferes with the data traffic, causing lower application performance.

In this work, we describe the pathologies caused by the use of Trickle as mechanism for keeping control traffic at a low rate at stable topologies while maintaining a fast adaptation to changes. We also offer possible solutions to address these pathologies.

II. The Trickle Dissemination Protocol

Trickle was originally designed for data dissemination in a multi-hop network. The data could be software version, or similar slow-changing data which is an ideal case for Trickle with its increasing intervals and suppression of already seen data. Later, Trickle was also used for timing the routing beacons. The Trickle protocol is specified by the pseudo-code in figure 1.

τ expires	$\rightarrow$	$\tau = \min(\tau \times 2, \tau_{max}), c = 0, t = [\frac{\tau}{2}, \tau]$
t expires $\wedge$ $c < k$	$\rightarrow$	transmit M_i
receive M_i	$\rightarrow$	$c = c + 1$
receive $M_{j>i}$	$\rightarrow$	$\tau = \tau_{start}, c = 0, t = [\frac{\tau}{2} - \tau]$

Fig. 1. The rules of the Trickle mechanism. When the maintenance interval (τ) expires, τ is doubled if not already at its maximum (τ_{max}). The redundancy counter c is also set to zero. When t expires, a metadata message is sent if c is less than k. When an incoming metadata message is received, c is increased if the message is consistent with the current metadata; otherwise the timers and intervals are reset.

Trickle disseminates new information fast, sends very little information when there is no new information available for dissemination, and does load balancing between the nodes.

III. Trickle Tricked

When all nodes have synchronized intervals, i.e., Trickle timers are approximately aligned across the nodes, Trickle handles propagation of meta-data efficiently and load balances the propagation between the nodes in the network. Unfortunately, the load balancing fails when the nodes start to lose synchronization with each other. In a network with bottleneck topology and desynchronized Trickle timer, information propagation can be slow.

One case where Trickle load balancing fails is when there is a large network of synchronized nodes and a new node joins by listening to the network beacons. The first beacon will be sent very close to half the interval which will cause this node to be unsynchronized with the other nodes. Figure 3

shows the consequence of having an unsynchronized node in a single-hop network consisting of 10 nodes.

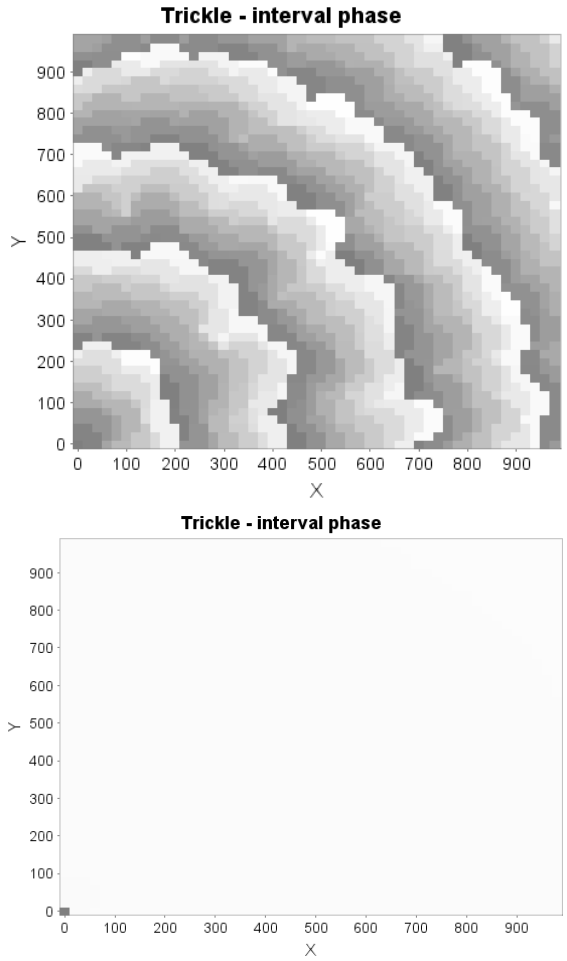

Fig. 2. Results from a simulation in a 2500-node network. On the top, we run the unmodified Trickle. We use the node in the lower left corner as reference to visualize the timer phase. White corresponds to nodes that have the same interval phase as the reference node. The darker the color, the more out of synch the interval. On the bottom figure, we run the modified Trickle that resynchronizes its interval.

IV. Trickle with resynchronization of intervals

The desynchronization in Trickle can be eliminated by adding a synchronization mechanism that resynchronize the time intervals in Trickle. This mechanism makes all nodes execute their Trickle listen and transmission intervals at the same time which improves the efficiency and also removes any lack of load balancing that may appear in a network. The algorithm described in Figure 4 achieves this goal.

The efficiency of the modified Trickle at a steady state when the intervals have been synchronized will approach "perfect" (e.g. k packet per interval in a network with full reachability) and in the worst case, assuming that the synchronization fails completely, it will be as efficient as Trickle with an interval of $\tau - w_s \frac{1}{4}\tau$. By resynchronizing, the Trickle protocol becomes both more efficient and better load balanced which reduces the propagation delays in bottleneck topologies. Figure 2 compares the synchronization properties of the original and the modified Trickle. We find that the modified Trickle was able to synchronize the schedules of the nodes across the network while the original Trickle was not.

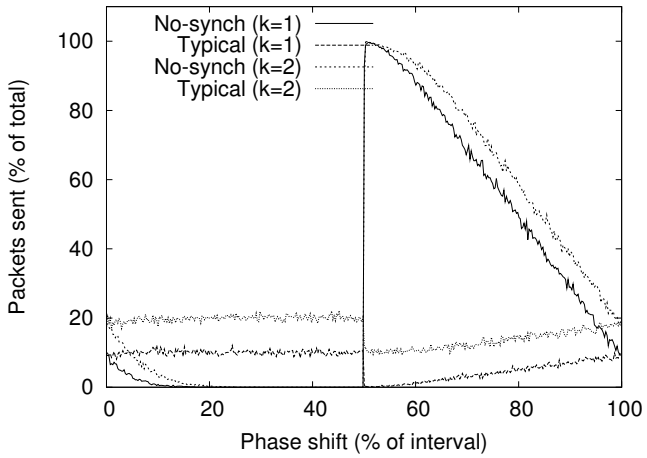

Fig. 3. Results from a simulation of a 11-node network. The network is setup with 10 synchronized nodes and one node that is out of phase with the other 10. k is set to 2. The phase is shifted from 0% to 100% of the maximum interval τ_{max}. The load balancing of Trickle breaks quite dramatically around 50% phase shift. Some nodes end up sending packets almost every interval while one did not perform any transmission.

τ_s expires	$\rightarrow$	$\tau = \min(2\tau, \tau_{max}), \tau_s = \tau - \delta_s,$ $c, \delta_s = 0, t = [\frac{\tau_s}{2}, \tau_s]$
t expires, $c < k$	$\rightarrow$	transmit M_i
receive M_i	$\rightarrow$	$c = c + 1$
receive M_i, $T \in [\frac{\tau}{4}, \frac{\tau}{2}]$	$\rightarrow$	$\delta_s = w_s(T - \frac{\tau}{2})$
receive $M_{j>i}$	$\rightarrow$	$\tau = \tau_{start}, c = 0, t = [\tau/2 - \tau]$

Fig. 4. Trickle modified for resynchronization of intervals. The idea is to start the next interval earlier if this node got a Trickle message during its second half of its listen only period.

V. Acknowledgements

We would like to thank Philip Levis and others in the IETF ROLL working group for the active discussions related to Trickle and its use in RPL.

References

[1] D. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the International Conference on Mobile Computing and Networking (ACM MobiCom)*, pages 134–146, San Diego, CA, USA, 2003. ACM.

[2] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, Berkeley, CA, USA, 2009.

[3] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the USENIX Symposium on Networked Systems Design & Implementation (NSDI)*, March 2004.

[4] J.P. Vasseur and A. Dunkels. *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann, 2010.

[5] T. Winter (Ed.), P. Thubert (Ed.), and RPL Author Team. RPL: IPv6 Routing Protocol for Low power and Lossy Networks. RFC 6550.