Evaluating the Performance of RPL and 6LoWPAN in TinyOS

JeongGil Ko
Department of Computer Science
Johns Hopkins University

Stephen Dawson-Haggerty
Computer Science Division
University of California, Berkeley

Omprakash Gnawali
Computer Science Department
Stanford University

David Culler
Computer Science Division
University of California, Berkeley

Andreas Terzis
Department of Computer Science
Johns Hopkins University

ABSTRACT

Responding to the increasing interest to connect wireless sensor networks (WSN) to the Internet, the IETF has proposed standards that enable IPv6-based sensor networks. Specifically, the IETF 6LoWPAN and RoLL working groups developed standards for encapsulating IPv6 datagrams in 802.15.4 frames, neighbor discovery, and routing that allow sensor networks to exchange IPv6 datagrams with Internet hosts. However, given that these standards, especially the RPL routing protocol, are relatively new, there has not yet been a study that measures the actual performance of these proposals using real implementations. In this work, we use the BLIP and TinyRPL implementations in TinyOS 2.x to evaluate the performance of the newly proposed standards and compare them with CTP, the de-facto routing protocol standard for TinyOS. Our results indicate that the performance of TinyRPL is comparable with CTP and at the same time, TinyRPL can provide additional functionalities that traditional WSN routing protocols could not provide. We also discovered several issues, relevant to system developers and the standardization groups, which can enhance the proposed standards' performance.

1. INTRODUCTION

For more than a decade, researchers considered the Internet architecture to be ill-suited for low-power and resource-constrained WSN platforms. However, increasing interest to connect WSN-based infrastructure (e.g, [10, 11, 15]) to the Internet and results from preliminary studies indicating the feasibility of using IPv6 in WSNs [9], led the Internet Engineering Task Force (IETF) to develop IP-based protocols for low-power and lossy networks (LLNs) through the 6LoWPAN and RoLL working groups.

The research community took a leading role in developing these standards, reflecting on results from over a decade of WSN research. As a result of these efforts, the IETF 6LoWPAN working group proposed standards that include mechanisms to effectively compress IPv6 addresses and also schemes to efficiently discover nodes using compressed IPv6 addresses [7]. More recently, the IETF RoLL working group developed the RPL standard, which is a routing protocol targeting IPv6-based LLNs.

These IETF standards were not developed in the void. Instead, many ideas originally developed as part of the TinyOS collaboration (e.g., [3, 4, 12]) and other research efforts, were adopted by the 6LoWPAN and RoLL working groups. As these standards are maturing, the TinyOS community is once again putting in efforts to distribute the proposed standards through open source implementations and real-life applications. Nevertheless, adoption will be determined eventually by the performance and features of these protocols, especially for resource-constrained sensor networks.

This work is the first effort to evaluate the performance of TinyRPL, the TinyOS implementation of the IETF RPL routing protocol, in a testbed environment. Specifically, we compare the performance of the latest TinyRPL implementation with the Collection Tree Protocol (CTP), the de-facto standard data collection protocol for TinyOS 2.x [4]. Our results indicate that the performance of TinyRPL is comparable, in terms of packet delivery and protocol overhead, to the well-engineered CTP protocol. Moreover, we show that the additional features, such as RPL's ability to establish bi-directional routes, makes TinyRPL more attractive for practical wireless sensing systems. Finally, based on our experience in implementing and evaluating TinyRPL, we provide suggestions for system developers that use the RPL routing protocol. We believe that these suggestions can benefit the standards development process.

2. RPL

Chartered in 2008, the goal of the IETF's Routing over Low-power and Lossy networks (RoLL) working group was to design a routing protocol that fits the various requirements introduced by the working group's target applications described in RFCs 5548 [2], 5673 [16], 5826 [1], and 5867 [14]. In 2010, the working group introduced the IPv6 Routing Protocol for Low-power and lossy networks (RPL) [18].

RPL's high-level goal is to provide efficient routing paths for

IPSN'11, April 12–14, 2011, Chicago, Illinois.

three traffic patterns, multipoint-to-point, point-to-multipoint, and point-to-point traffic in low-power and lossy networks (LLNs). Specifically, once a RPL node obtains a proper global IPv6 address (e.g., via DHCPv6), it tries to join a Destination Oriented Directed Acyclic Graph (DODAG) by exchanging ICMPv6-based DODAG Information Solicitation (DIS) or DODAG Information Object (DIO) messages.

Using such routing control messages[1], the root of a RPL DODAG advertises itself as a parent node for other nodes in its neighborhood. Once it selects a parent, a RPL node propagates its own DIO further down the network to form its sub-DODAG. When a DIO is received and a node tries to connect to a RPL DODAG, the node starts by computing its own "rank" value which represents its relative position within the DODAG from the root. The DIO messages also include objective functions (OFs) which contain the details on how the rank value is initially computed and further maintained. Currently two different types of OFs are specified as Internet Drafts of the RoLL working group, *OF0* [17], which is simply a hop count-based metric, and the Minimum Rank Objective Function with Hysteresis(*MRHOF*) [5], which uses hysteresis while selecting the path with the smallest metric value (e.g., path ETX).

As previously said, RPL supports three traffic patterns. Intuitively, multipoint-to-point (i.e., collection) traffic can be supported with little routing state, whereby each node stores the next hop leading to the single destination, the root of the DODAG. This state can be maintained by properly constructing the DODAG using DIO message exchanges. On the other hand, for the other two types of traffic patterns (i.e., point-to-multipoint and point-to-point), RPL provisions separate "downward" routes using an additional RPL control ICMPv6 message, the so-called Destination Advertisement Object (DAO) message. DAO messages advertise routes for various destinations and prefixes within a RPL network. Upon receiving a DAO message, depending on how the network is pre-configured, nodes either locally store the route (storing mode) or forward this route information to the root of the DODAG (non-storing mode). By collecting such individual route information, a packet can travel "up" the tree until it reaches a node that has knowledge of the routing path to the packet's ultimate destination.

It is notable that the design of RPL was heavily influenced by various findings from a decade of wireless sensor network research. Specifically, RPL reduces the cost of propagating routing state by using a Trickle-based timer [13], increases the probability of packet delivery by simultaneously provisioning multiple potential routing paths [4], and uses expressive link metrics to deal with various types of applications in challenging radio channel environments [3].

The RPL standards are based on a IPv6-based addressing layer (e.g., 6LoWPAN layer). While the RPL specification does not mandate the use of the 6LoWPAN specifications, it is important to keep an efficient IPv6 stack that can fit in resource-constrained mote platforms. Therefore, we consider the use of a 6LoWPAN layer to be a de-facto requirement for WSN systems. Specifically, drafts such as the IPv6 header compression draft [7] are essential for effective management of a mote's resources (e.g., radio capacity and memory resources). Finally, for applications that require or provide global connectivity, a communication path that physically connects the low-power wireless network to other nodes on the Internet is necessary (e.g., through a PPP connection at the edge router or the DODAG root).

[1]DIO messages carry details on how to connect to a node to a DODAG and DIS messages explicitly request DIO messages.

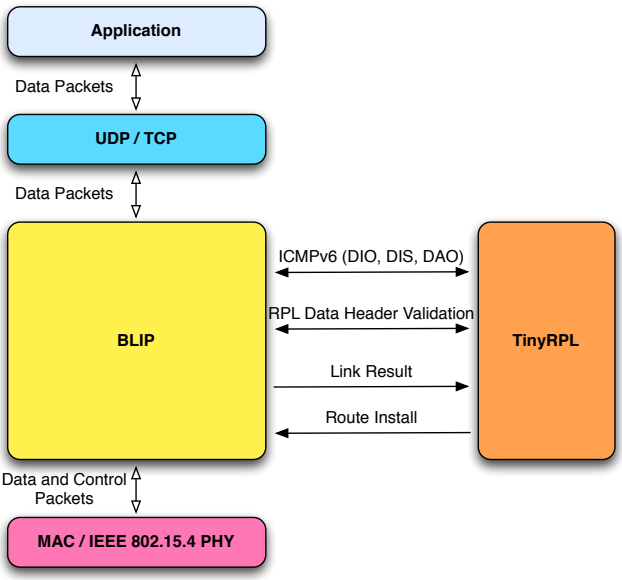

Figure 1: The 6LoWPAN/RPL software stack in TinyOS 2.x. TinyRPL implements the control plane of the RPL specifications and interacts with the packet forwarding plane implemented in BLIP.

3. RPL IN TINYOS

The RPL implementations in TinyOS is centered in two layers of the software stack, `blip`, the TinyOS 6LoWPAN stack, and `TinyRPL` the actual implementation of the RPL standard. This section introduces the interfaces and the interactions between TinyRPL and BLIP and also presents the performance of TinyRPL/BLIP.

3.1 BLIP

The Berkeley Low-power IP stack, `blip`, is the de-facto IPv6/6LoWPAN stack for TinyOS 2.x. Therefore, the RPL implementations in TinyOS interacts heavily with the interfaces that `blip` provides. `blip` implements the 6LoWPAN header compression, 6LoWPAN neighbor discovery and DHCPv6 to support the use of IPv6 in the upper layers. `blip` also provides a layered IP forwarding abstraction which allows routing protocols such as RPL to be implemented on top of the ICMP engine. In its interaction with TinyRPL, `blip` initiates the TinyRPL operations once a node is assigned a global address. Specifically, as Figure 1 shows, once TinyRPL establishes a route using the RPL-related ICMPv6 messages (e.g., when TinyRPL obtains knowledge of what the next hop node is for any desired/supported destination), the route is added to `blip`'s routing table. The forwarding engine in `blip` makes routing decisions by performing lookups in this table and also includes optional up-calls to the routing layer for each packet being forwarded; this allows the routing protocol to implement non-standard tests for packet forwarding. For instance, TinyRPL checks for an optional routing header to discover the existence of any routing loops. `blip` also manages per-interface message queues which are used to buffer outgoing packets, necessary to support bursts of outgoing packets such as those generated from sending a fragmented packet and that caused by head-of-line blocking during retransmissions.

The stack also provides a number of other components which allow implementers to work on just one layer rather than reinventing a large amount of new software. One important piece for our experiments was the implementation of the Point-to-Point protocol

Protocol	TinyRPL		CTP	
Packet Interval	5 sec	10 sec	5 sec	10 sec
PRR	99.88%	99.96%	99.94%	99.99%
Control Traffic	8.96	9.01	8.29	8.02

Table 1: Packet reception ratio (PRR) and the average number of routing control packets generated at each node per hour with varying traffic rates. We compare TinyRPL and CTP when each member of a 50-node testbed generates a packet every 5 and 10 seconds. The results are the average of two 24-hour testbed runs for each network configuration. The PRR and routing control packet overhead of TinyRPL is comparable with that of CTP.

(PPP). PPP is a framing and configuration protocol for many types of links which provide a byte-stream abstraction; it has mostly replaced the SLIP protocol. Using PPP, a TinyOS node running RPL can present itself as a network interface to another host or router. In our experiments, the node running as the RPL DODAG root also configures itself with a serial interface, which it used to route packets between the LLN running RPL and external networks such as the Internet. The PPP stack is still a work in progress, but provided satisfactory functionality for these experiments.

3.2 TinyRPL

TinyRPL is a prototype implementation of the IETF RPL routing protocol in TinyOS 2.x. As a sample implementation, TinyRPL supports the RPL draft's basic mechanisms, while omitting some of RPL's optional features, such as the security options. The current implementation provides the OF0 [17] or MRHOF [5] objective functions with the ETX metric. Nevertheless, the modular design of TinyRPL simplifies the use of additional objective functions.

4. EVALUATION

Next, we present the results of our RPL evaluation using measurements collected from a medium-size testbed. The same testbed was used to run CTP and we compare the performance of the two protocols.

4.1 Methodology

In this work, we evaluate the performance of TinyRPL with the path-ETX metric and MRHOF using various routing performance metrics and compare the measurements with the de-facto standard routing protocol in TinyOS 2.x, the Collection Tree Protocol (CTP) [4].

We present results collected from a testbed of 51 TelosB motes distributed over two floors of an office building. One of the 51 nodes was configured as the root of the routing tree while the remaining 50 nodes operated as packet generating sources that formed a routing tree using either TinyRPL or CTP. When the network boots up and a global address is issued by `blip`, the preassigned root node starts the TinyRPL operations by sending DIO packets (routing beacon packets for CTP[2]). After joining the routing tree, source nodes start to generate data packets destined to the root node. In our experiments, packets were generated at each source node with a fixed interval of 5 seconds or 10 seconds for two separate sets of tests. Note that in order to make a fair comparison with CTP, we temporarily disabled the downstream routing options

[2]CTP does not use the `blip` layer; thus, no global IPv6 address is assigned.

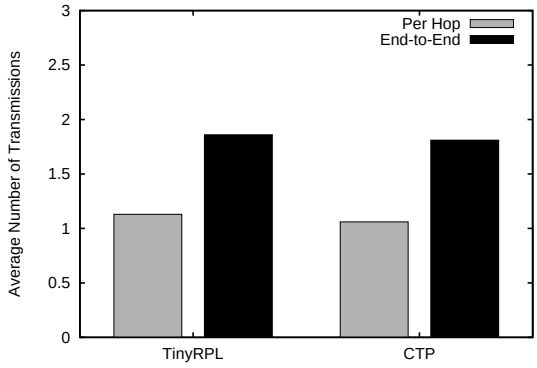

Figure 2: Average number of packet transmissions required at each hop (gray bars) and over the entire end-to-end path (black bars) for TinyRPL and CTP. The results indicate that TinyRPL and CTP both select links with very good quality and also that the path that CTP selects are slightly more efficient than the path that TinyRPL selects.

from TinyRPL; thus, no DAO packets were sent. Also note that the packets generated when using TinyRPL were UDP packets.

4.2 Packet Reception Ratio

The first result that we present in Table 1 is a comparison of the average end-to-end packet reception ratio (PRR) between TinyRPL and CTP computed over two runs for a duration of 24 hours each. The PRR values in Table 1 indicate that the PRR for TinyRPL is comparable with that of CTP and always higher than 99.8%.

Given the similar high PRR values of TinyRPL and CTP, another important metric to consider is the amount of overhead necessary to maintain the routing tree. It is always desirable to maintain well performing routes with minimal routing control overhead. Furthermore, since routing control packets are typically transmitted using multicast addresses and lead to longer radio uptimes in duty-cycled networks, minimizing their quantity is even more desirable for low-power and lossy networks. As discussed above, in RPL, the routing control packets are ICMPv6-based messages (e.g., DIO, DIS, and DAO messages) and likewise, CTP transmits routing beacons for the same purpose as DIOs. While the DIS packets are sent periodically from a RPL node until the first parent node is selected[3], a Trickle timer is used in both CTP and TinyRPL to schedule the transmission of DIOs and CTP routing beacons[4] [13]. The Trickle timer allows the beacon intervals to exponentially increase when the network conditions are stable and quickly decrease to the minimum interval when noticeable changes in the network conditions are detected; thus, using a Trickle timer effectively minimizes the amount of routing overhead when the network is stable.

In addition to the PRR values mentioned previously, Table 1 also shows that on average, 8.96 DIO packets were generated at each node per hour when TinyRPL source nodes were generating packets at a 5 second interval. Each CTP node generated 8.29 control packets per hour. The slightly higher routing control overhead for TinyRPL can be explained in two ways. The first reason, which is rather simple, is that the events that trigger a Trickle timer reset

[3]Few DIO packets are exchanged during the initial part of our experiments and we disable DAO packets.

[4]We match the Trickle timer parameters of TinyRPL to those of CTP (e.g., minimum interval as 0.128 seconds and maximum interval as 512 seconds).

are set differently in the two implementations. Moreover, TinyRPL nodes experienced slightly higher churn compared to CTP. While CTP experienced an average number of 0.241 parent changes for each node every hour, TinyRPL experienced a slightly higher 0.252 parent changes per node per hour. By making more frequent parent changes, the network was less stable, which, as a result, created more control packets. In any case, we conjecture that the small differences in the overhead measurements is caused by the use of different link quality estimation techniques that impact the parent selection process. While CTP uses the 4-bit link estimator [3], which combines link quality information from multiple layers in the networking stack, TinyRPL relies on the Minimum Rank Objective Function with Hysteresis (MRHOF) [5] with the path-ETX metric.

4.3 Routing Protocol Overhead

Another type of overhead related to the efficiency of a routing protocol is the overhead of attaching and transporting a routing header to the data packets. In RPL and CTP, these routing headers are used to detect route inconsistencies during data transmissions. For CTP, each data packet includes an 8-byte routing header that consists of information on the origin of the data packet along with information used for detecting routing loops. In TinyRPL, an 8-byte (including the IPv6 extension header size [8]) optional routing header, which contains the information needed for detecting loops, is attached to outgoing datagrams. Also, when using TinyRPL, the datagram's origin information, which the CTP routing header also contains, is attached at the BLIP layer using a 7-byte 6LoWPAN header (when 16-bit addresses are used). In summary, each data packet in TinyRPL contains up to 15 bytes of overhead to achieve the same functionalities of the CTP routing header. While the amount of overhead is higher for TinyRPL, we argue that this is only a small difference and also given that the RPL routing headers are optional [8], this difference can always be reduced. Overall, by combining these numbers with the results from Table 1, we can conclude that the per-packet overhead for TinyRPL and CTP are also comparable.

Finally, one of the main goals of a routing protocol for low-power and lossy networks, or any other wireless network for that matter, is selecting high quality paths that can minimize the aggregate number of transmissions, which directly translates to the energy consumed during packet transmissions and radio idle listening. For this, Figure 2 presents the total number of transmissions necessary to successfully deliver a packet to its final destination; the same figure also plots the number of transmission attempts made at each hop for both TinyRPL and CTP. The per-hop bars indicate that the quality of the links that TinyRPL and CTP selected were similar and that in both cases, these links had very high delivery ratios (e.g., per link ETX of 1.13 and 1.06 for TinyRPL and CTP respectively). While the per-hop link quality was close to identical, the two end-to-end bars indicate that the MRHOF-ETX-based TinyRPL selected slightly longer routes when compared to the 4-bit link estimator-based CTP (e.g., 1.81 for CTP vs. 1.86 for TinyRPL). Nevertheless, the numbers are very close, indicating that a MRHOF combined with path-ETX-based link selection mechanism was effective in selecting efficient end-to-end routing paths.

4.4 Downstream Routing

The results presented until now demonstrate that, for collection traffic, the performance of TinyRPL is comparable to CTP. In turn, these results suggest that an IPv6-based collection protocol provides high PRR and low routing overhead. However, what makes TinyRPL even more attractive than CTP are the additional features that the RPL specification offers. Major benefits that separate RPL from CTP include RPL's support for various types of traffic patterns (see Section 2) and its ability to directly connect to Internet nodes by exchanging packets with global IPv6 addresses.

	Response Ratio	Latency
Edge router	99.99%	132.23 ms
RPL node	98.51%	191.01 ms

Table 2: The response ratio (i.e., round-trip PRR) of actuation messages and their response latencies when using the PPP module provided with BLIP. Actuation messages are sent from a remote PC to a RPL network bridged to another PC using the PPP module. The edge router in the RPL network receives the messages from the PPP connection then forwards them to the target node using the downwards routes constructed by TinyRPL's storing mode option. Once the action is performed at the target node, the node responds to the remote PC indicating that the process was successful. The results show that the bi-directional paths that TinyRPL establishes enables LLN nodes to successfully exchange messages with nodes in the Internet. By using a reliable transport layer protocol, a RPL network can enable sensors to effectively communicate with a host in the Internet to exchange actuation messages.

To demonstrate this benefit, we present results from a simple application that uses actuation messages. For this, we set up a RPL network which is bridged to a PC using the PPP connections presented in Section 3.1. We then send actuation messages to the RPL nodes in this network asking them to toggle their LEDs. These actuation commands originate from a remote PC with a periodic interval of 2 seconds for 100 times to each reachable RPL node (e.g., 29 nodes + 1 edge router node). The RPL network establishes downwards routes, which are needed to forward these requests to the target RPL node, using the storing mode option described in the RPL specification. Upon receiving the actuation request and performing the requested operations, the RPL nodes send back a response indicating that the task was successfully completed. We present the response ratio (i.e., round-trip PRR) of the actuation message exchanges and their latency in Table 2. Our results indicate that the remote PC receives 98.51% of the actuation response messages that it requests, implying that the bi-directional links that are established by TinyRPL can successfully forward packets traveling in both directions. From a practical standpoint, these results indicate that when TinyRPL is combined with a reliable transport layer protocol it can offer the connectivity that a real-life application (e.g., home automation system) would expect.

5. EXPERIENCES AND DISCUSSION

Next, we present some of the points that we, as the implementers of the IETF proposed standards, feel that would help a system developer or even the standardization groups to improve the performance RPL-based networks.

Flexibility of the RPL Specifications.

Our experience with implementing TinyRPL suggests that the RPL specification provides a level of freedom to implementers, even to a point where the performance of the implementation can be heavily affected.

One area that warrants more detailed description is the set of events that trigger a DIO Trickle timer reset. The RPL specification indicates that a RPL implementation must reset the Trickle

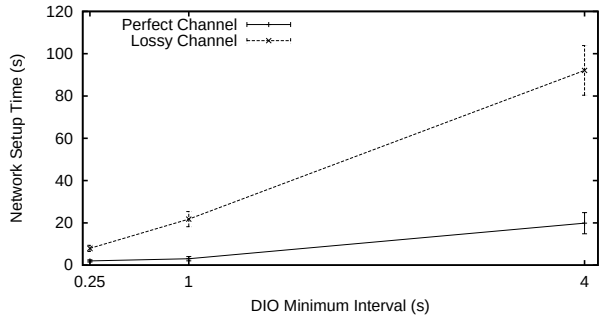

Figure 3: Cooja simulation results on the network setup delay for constructing the initial DODAG using different minimum intervals for DIO transmissions. Despite the topology having a small average hop count of 2.37, large values for the minimum interval of the DIO controlling Trickle timer can result in long network setup times when the network is lossy.

timer when a DIS message is received (with some restrictions [18]) and when an inconsistency or loop is detected. Additionally, depending on the implementation, a developer can define additional events that reset the DIO Trickle timer. Also separate implementations may come up with a diverse set of cases on how multicast DIS messages are used (which in turn will trigger a reset for the Trickle timer as well). Based upon our experience, the Trickle timer intervals and the number of their resets heavily affect the amount of overhead spent to maintain the DODAG. Without a firm definition, different RPL implementations can obtain extremely different results both in terms of routing overhead and PRR; therefore, we suggest that future amendments to the standard, further clarify the trigger conditions.

In the same context, we note that the loss of DIO messages and their rapidly increasing Trickle timers had a large impact on the network setup time and also the nodes' parent selection processes. As an example, we show network setup latency results obtained using our TinyRPL binary in the Cooja simulator with a random topology of 40 nodes (average hop count of the topology = 2.37). We perform experiments for a network with perfect channel conditions and also a network with lossy links. To simulate a lossy network, we configured a network where the average PRR of all possible links in the network was $\sim$56%. In this setting, we tested for cases where the DIO Trickle timer's minimum interval was 0.25 second, 1 second, and 4 seconds. For each test case, we compute the latency required to setup an initial DODAG that connects all the nodes in the network (e.g., time that the final node connects itself to the DODAG minus the time of the first DIO transmission).

Figure 3 shows the average of these values and their standard deviations computed over five runs for the two different channel conditions. Results indicate that despite having a topology with a small average hop count of 2.37 (i.e., shallow topology), the increase in the minimum interval for the DIO transmission Trickle timer results in a long setup latency when the channel conditions are lossy. We can also expect that if the number of hops in the DODAG's topology increases, this network setup latency can increase even further. While this may be less of an issue in LLNs that have the goal of long lifetimes with sparse traffic patterns, it can become critical in scenarios where the network needs to be deployed rapidly and carries high volumes of traffic (e.g., emergency response scenarios). Recently, as a first step to address such issues, the `Recommendations for Efficient Implementation of RPL` draft was proposed to discuss and clarify several topics on implementation decisions [6]. This document should be further expanded to address the issues raised above.

Downwards Route Provisioning.

RPL requires the network to select between the storing and the non-storing mode to provision downwards routes. The storing mode option effectively reduces communication overhead since it eliminates the additional routing header that specifies the routing path that a packet should travel to reach the target destination (typically attached at the DODAG root). On the other hand, using the storing mode requires a node to sacrifice more of its local memory to store routes, thereby leaving developers with a trade-off.

While making detailed estimations on the tradeoff is possible, if local memory spaces permit, it is always meaningful to reduce bandwidth usage as much as possible. In our experiments with downstream routing in storing mode, we noticed that the current TinyRPL/BLIP implementations on TelosB motes support up to $\sim$30 target destinations. This argues that to support routes to all nodes in the RPL network, the network's size must be limited to $\sim$30 nodes. This of course can be further optimized by using compressed addresses and constructing routing tables based on IPv6 prefixes rather than individual IPv6 addresses. In any case, we advise that the size of the implementations themselves can be a way of determining which downstream route provisioning mode to use when designing a RPL-based WSN system.

Fragmented packet forwarding.

Another implementation decision bearing further investigation is the forwarding path for large, fragmented packets. A strictly layered approach requires the entire IPv6 packet to be reconstructed at each hop, so that routing decisions are made with the entire packet on hand; this is the approach taken by the current implementations of `blip`. A disadvantage of this approach is that it increases memory pressure on forwarding nodes: they must allocate a reconstruction buffer for each packet being forwarded, which may remain allocated until a timeout (in the case of packet drops) occurs. An alternative approach is to decompress only the IPv6 header contained in the first fragment, and make a routing decision. The forwarder can then cache the next hop determination for succeeding fragments at the link-layer and dispatch incoming fragments directly to the output queue. This approach results in both reduced latency and reduced memory pressure, and has been successfully used in previous versions of `blip`.

6. SUMMARY

This work evaluates the performance of the IETF RPL routing protocol using an implementation in TinyOS 2.x. Our implementation of RPL, TinyRPL is combined with BLIP, the 6LoWPAN stack in TinyOS to provide efficient routing performance, in terms of PRR and protocol overhead, that is comparable with CTP, the de-facto routing protocol in TinyOS. Furthermore, we show that, by leveraging RPL's capabilities of performing bi-directional routing and forwarding IPv6 data frames, TinyRPL/BLIP can successfully communicate with devices in the greater Internet. Our experience with implementing and evaluating RPL has allowed us to provide some suggestions that can benefit future system developers as well as improve the quality of the proposed standards.

Acknowledgments

We thank the anonymous reviewers for their insightful comments that helped us improve the quality of this paper. We would like

to acknowledge Albert Goto for managing the testbed for our application evaluations in Section 4.4. This work was funded by NSF Awards #0435454, #0454432, #0855191, CPS-0931843, and the Stanford Army High Performance Computing Research Center Grant W911NF-07-2-0027.

7. REFERENCES

[1] A. Brandt, J. Buron, and G. Porcu. Home Automation Routing Requirements in Low-Power and Lossy Networks. RFC 5826, April 2010.

[2] M. Dohler, T. Watteyne, T. Winter, and D. Barthel. Routing Requirements for Urban Low-Power and Lossy Networks. RFC 5548, May 2009.

[3] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, and Philip Levis. Four-Bit Wireless Link Estimation. In *Proceedings of the sixth workshop on Hot Topics in Networks (HotNets)*, November 2007.

[4] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection Tree Protocol. In *Proceedings of the 7^{th} ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 1–14, Nov 2009.

[5] O. Gnawali and P. Levis. The minimum rank objective function with hysteresis. Internet Draft (Work in Progress), IETF, 2010.

[6] O. Gnawali and P. Levis. Recommendations for Efficient Implementation of RPL. Internet Draft (Work in Progress), IETF, 2011.

[7] J. Hui and P. Thubert. Compression Format for IPv6 Datagrams in 6LoWPAN Networks. Internet Draft, IETF, 2010.

[8] J. Hui and JP. Vasseur. RPL Option for Carrying RPL Information in Data-Plane Datagrams. Internet Draft (Work in Progress), IETF, 2010.

[9] Jonathan W. Hui and David E. Culler. IP is dead, long live IP for wireless sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 15–28, New York, NY, USA, 2008. ACM.

[10] X. Jiang, S. Dawson-Haggerty, P. Dutta, and D. Culler. Design and implementation of a high-fidelity AC metering network. In *Proceedings of the International Conference on Information Processing in Sensor Networks*, pages 253–264, 2009.

[11] JeongGil Ko, JongHyun Lim, Yin Chen, Razvan Musaloiu-E., Andreas Terzis, Gerald Masson, Tia Gao, Walt Destler, Leo Selavo, and Richard Dutton. MEDiSN: Medical Emergency Detection in Sensor Networks. *ACM Transactions on Embedded Computing Systems (TECS), Special Issue on Wireless Health Systems*, 10(1):1–29, 2010.

[12] Philip Levis, Eric Brewer, David Culler, David Gay, Samuel Madden, Neil Patel, Joe Polastre, Scott Shenker, Robert Szewczyk, and Alec Woo. The emergence of a networking primitive in wireless sensor networks. *Commun. ACM*, 51(7):99–106, 2008.

[13] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A Self-regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In *Proceedings of NSDI*, March 2004.

[14] J. Martocci, P. De Mil, N. Riou, and W. Vermeylen. Building Automation Routing Requirements in Low-Power and Lossy Networks. RFC 5867, June 2010.

[15] Jeongyeup Paek, Krishna Chintalapudi, John Cafferey, Ramesh Govindan, and Sami Masri. A wireless sensor network for structural health monitoring: Performance and experience. In *Proceedings of the Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, May 2005.

[16] K. Pister, P. Thubert, S. Dwars, and T. Phinney. Industrial Routing Requirements in Low-Power and Lossy Networks. RFC 5673, October 2009.

[17] P. Thubert. RPL Objective Function 0. Internet Draft (Work in Progress), IETF, 2010.

[18] T. Winter, P. Thubert, and RPL Author Team. RPL: IPv6 Routing Protocol for Low power and Lossy Networks. Internet Draft (Work in Progress), IETF, 2010.