ContikiRPL and TinyRPL: Happy Together

JeongGil Ko[1], Joakim Eriksson[2], Nicolas Tsiftes[2], Stephen Dawson-Haggerty[3], Andreas Terzis[1], Adam Dunkels[2] and David Culler[3]

[1] Department of Computer Science, Johns Hopkins University
[2] Swedish Institute of Computer Science (SICS)
[3] Computer Science Division, University of California, Berkeley

ABSTRACT

IP-based sensor networks provide interoperability, but experience shows that interoperability between different protocol implementations is not a binary property. Instead, subtle differences in implementation choices may affect the performance of the resulting system. We present our experiences with the Contiki and TinyOS implementations of the IPv6 stack for low-power and lossy (LLN) networks including the IETF 6LoWPAN adaptation layer and the IETF RPL protocol. Our results show two independent implementations, that have good performance on their own, can have a substandard performance in a mixed network configuration and that small implementation differences can lead to large-scale differences in behavior. This suggests that IPv6 stack implementations for LLNs need to be tested not just for correctness but also for performance.

1. INTRODUCTION

In sensor network research, interoperability has traditionally been of little importance. Research deployments have been single-purpose systems consisting of homogeneous nodes all running the same software. Likewise, software intended for sensor network research, such as the early TinyOS work and the Contiki Rime stack [4], was intended as a platform for experimental research. Therefore, interoperability was seldom seen as an important goal.

On the other hand, interoperability has always been important in commercial sensor networks, but progress towards interoperability has been slow due to the plethora of different protocols stacks, specifications, and non-standards. The Internet Protocol (IP), which has proven its interoperability and extensibility as the basis of the global Internet over the last 30 years, is seen by many as the solution to the interoperability problem in low-power wireless sensor networks [6, 11, 12]. Many central challenges in IPv6-based sensor networks are being addressed, such as implementation complexity [3], header compression [8], and routing [13].

The IPv6 protocol stack for low-power and lossy networks (LLNs) consists of the traditional IPv6 protocols but augmented with the RPL routing protocol [12, 13] and the 6LoWPAN adaptation layer [8]. The IPv6 stack is intended to provide interoperability, but given its complexity and the many implementation-specific choices in the specifications, the performance of a network running multiple independent implementations of the stack is not clear. To further complicate matters, configuration parameters such as number and interval of retransmissions are defined by different standards bodies for different layers of the stack: e.g. the IETF for the network layer and the IEEE for the MAC layer.

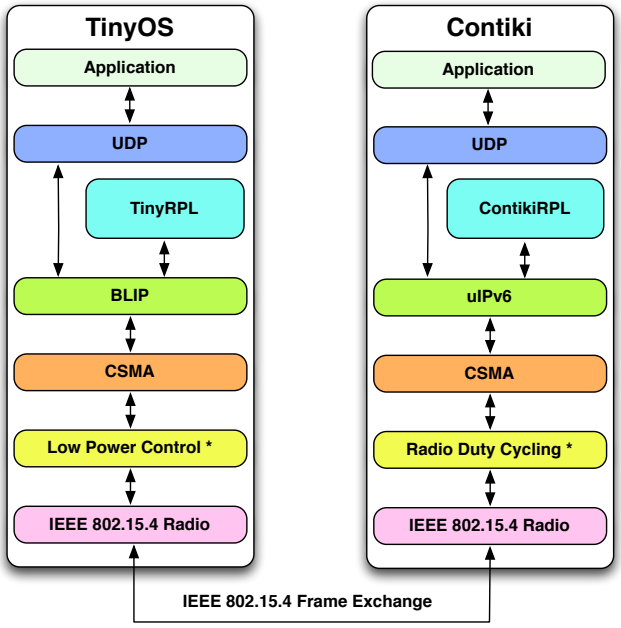


Figure 1: Making Contiki and TinyOS interoperate.

To investigate the performance of interoperability, we present and evaluate the performance of two independent IPv6 implementations for LLNs, one for Contiki and one for TinyOS, as shown in Figure 1. Both Contiki and TinyOS provide IPv6 header compression and fragmentation via the 6LoWPAN adaptation layer and many of the standard protocols in the TCP/IP protocol suite [2, 3, 5].

The contributions of this paper are two. First, we are the first to demonstrate interoperability between RPL implementations for the leading sensor network operating systems, TinyOS and Contiki. Second, we show that interoperability between independent implementations of an IPv6 protocol stack for LLNs is not a binary

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

property: subtle variations in implementation choices and in system components can affect system performance in unexpected ways. Our results show that although two implementations may demonstrate interoperability and high performance in terms of packet delivery, there is a need to look deeper into the network performance as subtle variations in underlying components may affect both packet delivery and network churn. With this work, we are the first to identify these problems and to provide directions on how to address them. While we focus on the RPL routing protocol, our results show that parameters of lower-level protocols can effect performance profoundly.

Our paper is structured as follows. We introduce the RPL protocol in Section 2 and present our ContikiRPL and TinyRPL implementations in Section 3. In Section 4 we establish the baseline performance of our two implementations, demonstrating that ContikiRPL and TinyRPL both have high and comparable performance. In Section 5 we put ContikiRPL and TinyRPL together and show that although our two implementations interoperate, implementation choices can have unexpected effects on the performance of networks where two implementations are intermingled. In Section 6 we discuss our experiences and lessons learned from making RPL interoperate and in Section 7 we give directions for future work. We discuss related work in Section 8 and conclude the paper in Section 9.

2. RPL

RPL (Routing Protocol for Low-power and Lossy Networks) is the IETF proposed standard protocol for IPv6 routing over Low-power, Lossy Networks (LLNs). RPL is designed for networks with significantly higher packet loss rates than traditional IP routing protocols. RPL is optimized for the many-to-one traffic pattern where many nodes send data towards a border router or sink node, but provides a set of mechanisms for any-to-any routing as well.

A RPL network is a Destination-Oriented Directed Acyclic Graph (DODAG) rooted at a border router. All nodes in the network maintain a route to the DODAG root. DODAG construction and maintenance is primarily done using two custom ICMPv6 messages: DIO (DODAG Information Object) and DIS (DODAG Information Solicitation). The root sends a DIO message that indicates that the node has a DODAG rank of `MinHopRankIncrease`. Its neighbors take a rank that is at least `MinHopRankIncrease` larger than that of the sink (or their parent node), then transmit their own DIO messages. Nodes may also ask for a DIO message from their neighbors by transmitting a DIS message.

Because different LLNs have different properties and requirements, RPL has been designed to allow for a great deal of flexibility. In RPL, nodes chose routing parents based on a generic objective function. RPL allows individual RPL networks to chose different objective functions. A power-constrained network can choose an objective function that optimizes network power consumption and a latency-bound network can choose an objective function that optimizes latency. DIO messages include a list of objective functions that the sending node supports. To provide a baseline for interoperability, RPL includes a default objective function, called Objective Function 0 (OF0), that only seeks to optimize hop count.

3. TWO RPL IMPLEMENTATIONS

In this work, we consider two independent implementations of RPL, ContikiRPL and TinyRPL. ContikiRPL is part of the Contiki operating system and TinyRPL is implemented in TinyOS 2.x. The structure of the two systems is shown in Figure 1.

3.1 ContikiRPL

ContikiRPL implements the RPL protocol, as specified in version 18 of the RPL specification [13], and two objective functions—OF0 and the Minimum Rank Objective Function with Hysteresis (MRHOF). ContikiRPL has been successfully tested for interoperability through the IPSO Alliance interop program, where it was used on three different platforms and ran over two different link layers, IEEE 802.15.4 and the Watteco low-power power-line communication module. ContikiRPL has also been used in two sensor network deployments.

The ContikiRPL implementation separates protocol logic, message construction and parsing, and objective functions into different modules. The protocol logic module manages DODAG information, maintains a set of candidate parents and their associated information, communicates with objective function modules, and validates RPL messages at a logical level according to the RPL specification. The message construction and parsing module translates between RPL's ICMPv6 message format and ContikiRPL's own abstract data structures. Finally, the objective function modules implement an objective function API. To facilitate simple replacement of and experimentation with objective functions, their internal operation is opaque to ContikiRPL.

ContikiRPL does not make forwarding decisions per packet, but sets up forwarding tables for uIPv6 and leaves the actual packet forwarding to uIPv6. Outgoing IPv6 packets flow from the uIPv6 layer to the 6LoWPAN layer for header compression and fragmentation. The 6LoWPAN layer sends outgoing packets to the MAC layer. The default Contiki MAC layer is a CSMA/CA mechanism that places outgoing packets on a queue. Packets from the queue are transmitted in order through the radio duty cycling (RDC) layer. The RDC layer in turn transmits the packets through the radio link layer. The MAC layer will retransmit packets until it sees a link-layer acknowledgment from the receiver. If a collision occurs, the MAC layer does a linear back-off and retransmits the packet. Outgoing packets have a configurable threshold for the maximum number of transmissions.

An external neighbor information module provides link cost estimation updates through a callback function. Within one sec of such an update, ContikiRPL recomputes the path cost to the sink via the updated link, and checks with the selected objective function whether to switch the preferred parent. The link cost reflects the per-neighbor ETX metric, which is calculated using an exponentially-weighted moving average function over the number of link-layer transmissions with $\alpha = 0.2$. The ETX is used to compute the rank for the MRHOF objective function and in parent selection for the OF0 objective function. New neighbor table entries have an initial ETX estimate of 5. The ContikiRPL neighbor eviction policy is to keep neighbors that have good ETX estimates and low ranks.

3.2 TinyRPL

The RPL implementation in TinyOS, TinyRPL, is a prototype implementation of the IETF RPL draft version 18 [13] with the goal of providing the TinyOS community a software component for building IP-based sensor network systems [7]. It provides all the basic features of the RPL draft while omitting some of the optional functionalities (e.g., security) [13]. TinyRPL relies heavily on the interfaces provided by `blip`, the implementation of the IPv6 stack (e.g., 6LoWPAN specifications) in TinyOS.

Similar to ContikiRPL, TinyRPL installs its routes separately at `blip`'s packet forwarding module once the DODAG is constructed using DIO messages. Since the routes to any known destination (including the default route) are installed in the forwarding module,

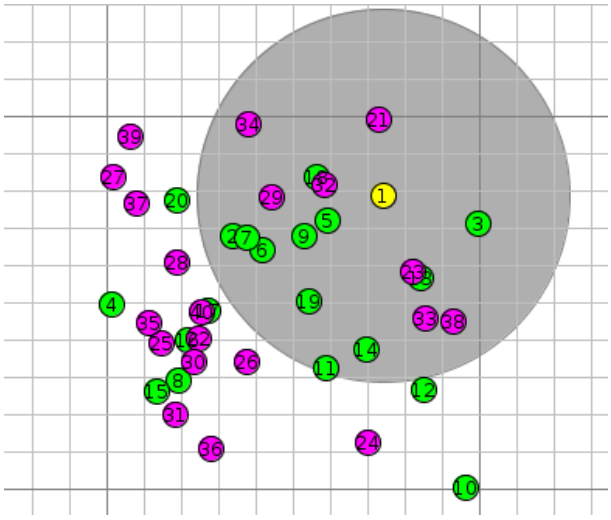


Figure 2: The simulation topology consists of 40 nodes, 39 sender nodes and one sink. The figure shows a sample topology in which ContikiRPL nodes (lighter nodes) and TinyRPL nodes (darker nodes) coexist. We show the communication range of the sink (node 1).

upper-level protocols can directly send their messages through the forwarding module in `blip`.

The parent node selection and rank calculation policies are controlled using either Objective Function 0 (OF0) or MRHOF. For both objective functions (OFs), a new node is added to the parent set (i.e., the set of nodes with lower rank values in a node's single hop neighborhood) with a local expected number of transmissions (ETX) value of 3.5. For OF0, which is essentially a hop-count-based parent selection metric, the local ETX measurements are used as a tie-breaker for potential parents with the same hop count. When MRHOF is used, the local ETX measurements and the parents' path-ETX values (included in the metric container option of the DIO) are used to compute a node's end-to-end path ETX to the root of the DODAG. Due to TinyRPL's modular design, additional OFs can also be supported easily.

TinyRPL updates the ETX values after each unicast transmission. The radio stack provides information on how many transmission attempts were made to (un)successfully send a unicast packet (e.g., receiving a proper acknowledgment would indicate that the packet was successfully delivered). Using this report, TinyRPL updates its ETX to a node using an exponentially weighted moving average (EWMA) of $\alpha = 0.5$. Whenever a new ETX is set for a node in the parent set, TinyRPL re-evaluates the elements in its potential parent set to select the node with the best metric to be its desired parent in the DODAG.

While TinyRPL does not maintain a message queue on its own, `blip` includes a message queue that buffers outgoing messages. This buffer is useful in cases where a packet needs to be retransmitted or if a burst of packets arrive at the node for further forwarding in the DODAG.

4. EVALUATION

We evaluate ContikiRPL and TinyRPL in two steps. We first study the performance of the two implementations in isolation, proving that both implementations achieve high and comparable performance. We then proceed to running the two systems together and study the resulting performance. The results show that two interoperable implementations can demonstrate surprising performance artifacts, despite the fact that they perform well in isolation.

To be able to control all parameters for our evaluation, we use the Contiki simulation environment. The Contiki simulation environment consists of two systems: the MSPsim node-level emulator and the Cooja network simulator. MSPsim is a cycle-accurate emulator of the Tmote Sky mote with a bit-level accurate emulation of the CC2420 radio transceiver. Cooja is a network-level simulator that enables bit-level accurate timing between different MSPsim nodes.

Cooja allows for a range of radio models and for this paper we use a simple unit disk graph model with a Bernoulli loss model where the loss probability is proportional to the square of the distance between nodes. The loss probability at the edge of the transmission range is configurable. Throughout this paper, we use three different path loss configurations: one with no path loss; one with 50% loss at the edge of each transmission range, resulting in an average link reception rate of 78%; and one with a 100% loss at the edge of each transmission range, resulting in an average link reception rate of 56%. We are aware that this loss model does not accurately capture the typical loss behavior found in real-world deployments. We use it here not to emulate reality, but only to create a network dynamism for our RPL implementations. Running our implementations in a testbed, to study their behavior in face of realistic wireless losses, is left as future work.

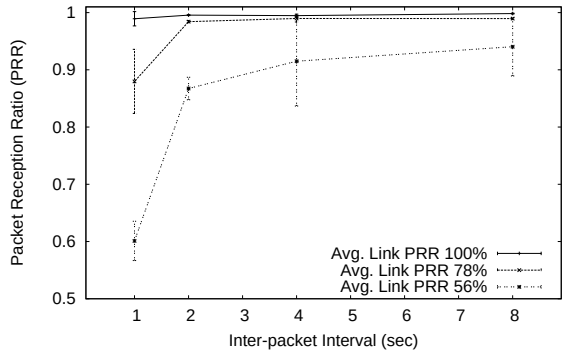
We first investigate the performance of ContikiRPL and TinyRPL in isolation. We use the topology presented in Figure 2 consisting of either all ContikiRPL nodes or all TinyRPL nodes. With each network configuration, we perform tests with different traffic loads where each non-root node sends one packet every 1, 2, 4, or 8 seconds. Figures 3(a) and 3(b) shows the overall average PRR of the ContikiRPL and TinyRPL networks along with the standard deviation for each test case computed over 10 separate runs. The figures show that TinyRPL and ContikiRPL both show high PRR values (for any channel condition) when the inter-packet interval (IPI) is either 4 or 8 seconds and this PRR degrades gracefully with the increase in traffic rate.

Figures 3(a) and 3(b) not only show that the two implementations operate well independently but they provide hints on the upper-bound performance that we can expect from a mixed network of TinyRPL and ContikiRPL nodes. Specifically, for the ranges where both implementations performed well (i.e., IPI values of 4 and 8), the expected result would be in the high-90% range for the perfect channel case and the case where the average link quality is 78%. Also, when the channel is at its lossiest configuration, we should expect a PRR $\sim$94%. We use these baseline performance values to validate the performance of the heterogeneous networks that we present in the following section.

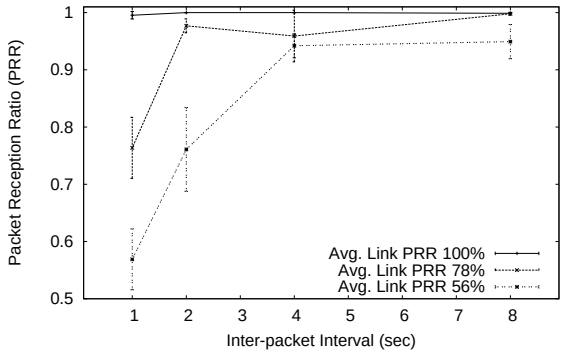
5. INTEROPERABILITY IS NOT ENOUGH

Having established the baseline performance of both ContikiRPL and TinyRPL, we now turn our attention to studying the performance when the two implementations interoperate. We use the Contiki simulation environment with the 40-node topology shown in Figure 2. To investigate how the two implementations work together, we vary the ratio of ContikiRPL and TinyRPL nodes from having only ContikiRPL nodes to having only TinyRPL nodes. Our results show that although two implementations interoperate, interoperability alone is not enough. Two interoperable implementations can display a suboptimal performance despite demonstrating excellent performance on their own.

We first study the packet reception rate of networks where ContikiRPL and TinyRPL are intermingled. For this experiment, we



(a) Packet reception ratio for TinyRPL as a function of the inter-packet interval.



(b) Packet reception ratio for ContikiRPL as a function of the inter-packet interval.

Figure 3: Packet Reception Ratios for TinyRPL and ContikiRPL. Statistics are collected from ten Cooja simulations on the topology shown in Figure 2.

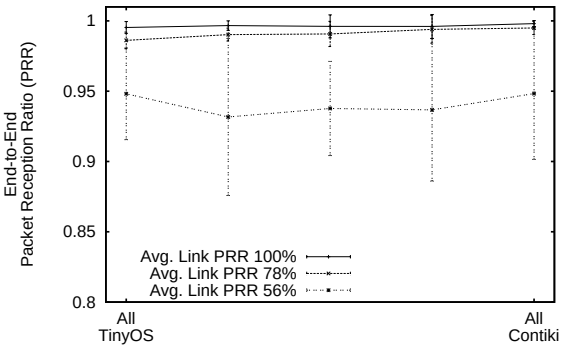


Figure 4: Perfect interoperability between ContikiRPL and TinyRPL. The packet reception rate (PRR) for the 40-node topology with a varied distribution of ContikiRPL and TinyRPL nodes on the x-axis. To the left, there is only TinyRPL nodes and to the right, there are only ContikiRPL nodes. ContikiRPL and TinyRPL were configured with the same MAC-layer parameters. The PRR is consistent for all configurations.

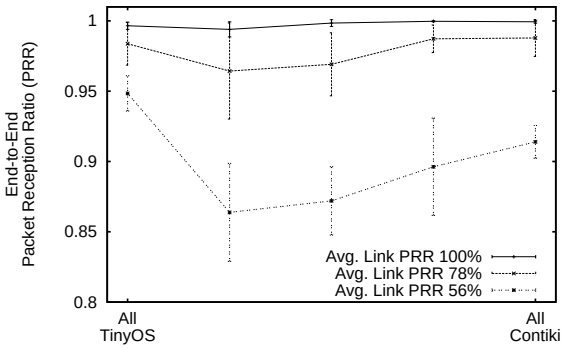


Figure 5: Average PRR when the MAC-layer parameters of ContikiRPL were in their original settings (e.g., retransmission timeout = 128 ms and transmission queue size = 4 compared to the retransmission timeout of 50 ms and queue size of 10 in TinyRPL). The performance of the pure networks remains high, but the performance of the mixed networks is significantly reduced. This is particularly pronounced when link loss is high.

match the configuration parameters in ContikiRPL and TinyRPL to each other. We configured the same MAC-layer retransmission timeout (50 ms) and the same MAC-layer queue length (10 packets). Each sending node transmits a packet every 8 seconds, which is a reasonable albeit somewhat high transmission rate.

We varied the distribution of ContikiRPL and TinyRPL nodes in steps of ten nodes: the first setup used 39 ContikiRPL source nodes and 0 TinyRPL source nodes, the next had 29 ContikiRPL nodes and 10 TinyRPL nodes, and so on. For each distribution, we made one configuration with a ContikiRPL sink and one configuration with a TinyRPL sink. We ran each network configuration ten times to get a dispersion metric. We measured the packet reception rate at the sink and we present the results in Figure 4. The figure shows that packet reception rates are consistent across all configurations.

Based on the baseline results from Section 4, the result shown in Figure 4 is the ideal result for interoperability testing: the two implementations perform well on their own, but also when intermingled. These results were, however, not the results we saw when we first tested the interoperability of our two implementations.

In our initial experiments, we used the default Contiki MAC-layer parameters and the default TinyOS MAC-layer parameters. By default, Contiki uses a MAC-layer retransmission timeout base time of 128 ms whereas TinyOS uses a timeout base time of 50 ms. The default Contiki MAC-layer transmission queue is 4 whereas TinyOS uses 10. Using these parameters showed to have unexpected results on system performance. We reran the same simulations as above but with the default Contiki settings.

Figure 5 shows the packet reception rate when Contiki's default MAC parameters were used. In both extreme cases—only TinyRPL and only ContikiRPL nodes—system performance is as good as in Figure 4. But when there is a mixture of ContikiRPL and TinyRPL, performance degrades, particularly when link losses are high. We have not fully been able to explain this phenomenon, but our initial hypothesis is that the slower MAC-layer retransmissions and shorter queue causes ContikiRPL nodes to create an inadvertent back pressure mechanism that does not work well when faster TinyRPL nodes are part of the network.

Finally, to gain insight into the performance of interoperability,

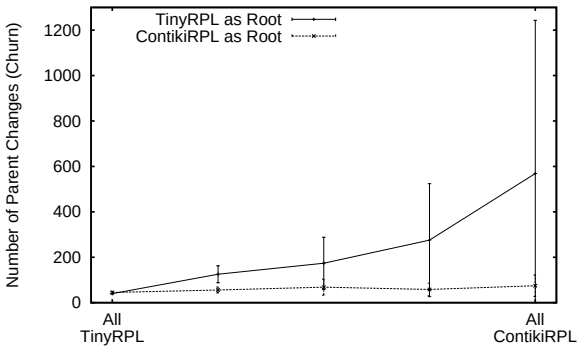


Figure 6: Churn measurements for tests with a TinyRPL sink and ContikiRPL sink. The number of parent switches are stable for most cases but as Contiki nodes start to make up a majority of the network with a TinyRPL sink, the churn starts to fluctuate significantly.

there is a need to look beyond the packet reception rates seen by the sink. Although the PRR shown in Figure 4 is high, there is a dynamism in the network that cannot be seen by only inspecting the packets received at the sink. By counting the number of times nodes switch parents in the two RPL implementations, we can calculate the churn rate of the network. The average churn rates of the simulation in Figure 4 is shown in Figure 6. We see that churn is low for networks with a ContikiRPL sink and for networks with only TinyRPL nodes. On the other hand, a network with ContikiRPL senders and a TinyRPL sink exhibit a pathological behavior where churn rates rise sharply. Although the churn rate does not impact the PRR in our experiments, it may increase the control traffic overhead and can cause packet reordering. We have yet to find the root cause of this behavior, but show these results here to demonstrate that developers need to look deeper than packet reception rates to achieve interoperability.

6. LESSONS LEARNED

The results presented thus far give the impression that interoperability between ContikiRPL and TinyRPL was immediate and painless. This was not the case. After ironing out the initial quirks, such as making sure that ContikiRPL and TinyRPL used the same 6LoWPAN header compression mechanism, the same 802.15.4 address encodings, and the same set of RPL features, we were able to run networks with TinyRPL and ContikiRPL. But when we begun setting up larger networks with more demanding loss patterns, we quickly found that each implementation would trigger bugs and unexpected behavior in the other implementation.

Based on our experiences with ContikiRPL and TinyRPL, we learned a number of important lessons.

Maintain a lowest common denominator. We initially hoped to run ContikiRPL and TinyRPL with all their bells and whistles, but quickly found that we needed to agree on a common minimal set of mechanisms to run. The most prominent effect of this was the choice of RPL route selection objective function. Although both ContikiRPL and TinyRPL implemented both the proposed OF0 and MRHOF objective functions, our two MRHOF implementations did not fully agree on the format of messages interchanged between the objective functions. Thus we chose to use OF0 only. But since the development of both ContikiRPL and TinyRPL had focused on MRHOF, the OF0 code had been unmaintained for some time. Because of this, we were bitten by a set of bugs in our OF0 code. It is therefore important not only that implementations provide a lowest common denominator but also that this part of the code is maintained and regularly tested.

Leverage simulation. An interoperability tester has a choice of performing the interoperability testing in a testbed or in a simulator. A testbed-based approach may achieve more realistic results, but does not allow control of all parameters. Moreover, simulation allows deep visibility into the behavior of the network that a testbed cannot easily provide.

Throughout this work, we relied on network simulation for investigating both the interoperability and the performance of our implementations. Simulation was crucial both to be able to quickly run many network setups and to allow visibility into the network. To test each parameter setting, we ran 300 simulated networks in the Contiki simulator for 15 minutes each. Running on a simulation cluster with three desktop PCs, 300 network setups were simulated in roughly two hours. On a testbed, each experiment would have taken us over three days to run.

The visibility provided by simulation allowed us to find and diagnose serious problems that we found during interoperability testing. An unforeseen interaction between ContikiRPL and TinyRPL, could in some rare cases trigger a bug that caused a non-sink node to advertise a rank of 0, causing other nodes to direct their traffic to it, which would drop all incoming packets. Our simulation logs would only reveal that the PRR would decrease to less than 20%. But by running the same network setup, with the same random seed, we could deterministically replay the scenario in the simulator, where we could inspect each packet sent in the network and thereby diagnose and resolve the problem. In a testbed, it would be impossible to deterministically repeat the exact sequence of events leading up to this rare case.

We used the Contiki simulation environment, which emulates the node hardware and therefore naturally allows different operating systems to be mixed in the same simulated network. This ability was essential for us, not only for demonstrating interoperability, but also for finding and diagnosing the problems we saw.

Simulate with accurate timing. In our first interoperability experiments with ContikiRPL and TinyRPL, the performance of a mixture of ContikiRPL and TinyRPL nodes was surprisingly low. Our investigation showed that part of this performance degradation was due to a difference in MAC-layer retransmission timing between the two systems. ContikiRPL had a retransmission timeout that was twice as long as TinyRPL. We were able to find this problem because the Contiki simulation environment allows simulation at a fine-grained timing level: packet transmission and reception is simulated at the bit-level.

Embrace the underlying layers. System behavior and performance is affected by all layers of the system. In the case of RPL, the routing choices made by RPL are affected by the underlying layers: the MAC layer, the radio duty cycling layer, and the radio link layer. The retransmission policy of the MAC layer will affect how quickly RPL can identify poor links. The queue length of the MAC layer's transmission queue will affect the total packet reception rate of the system. The impact of these parameters may not be reflected when running a network with only one implementation of the protocol, but will become pronounced only when different implementations are intermixed. In our case (cf. Fig.5) configuration parameters and design decisions from the underlying layers can affect the performance of the RPL layer in unexpected ways.

7. NEXT STEPS

While this paper is one of the earliest efforts to examine the fea-

sibility of RPL-based sensor network deployments interoperating, there are still more issues to examine before heterogeneous RPL networks can be practically deployed. We have not yet tested the interoperability of the downstream routing capabilities that RPL provides. Since this bi-directional routing capability is a major benefit of using RPL, making sure that the downwards traffic can interoperate properly is very important.

In this paper, we use the OF0 objective function, the simplest of the objective function that RPL networks should support. More complex objective functions such as MRHOF may provide further interesting insights given that these OFs require decisions with more implementation-specific input. Finally, another interesting direction to explore is the interoperability of different low-power MAC layers (e.g., Low Power Control and Radio Duty Cycling layers in Figure 1). In this work both implementations were configured so that no radio duty cycling was employed. However, given that it is most likely that RPL network deployments will incorporate some sort of low power radio control, we plan to take the path of exploring how different low power MAC protocols can interoperate.

8. RELATED WORK

Interoperability has always been a cornerstone in protocol standardization within the IETF: only protocol specifications with interoperable implementations can become standards. Experience has shown that implementation aspects have a significant impact on network performance. Subtle differences in implementations of TCP, the most widely used transport protocol on the Internet, were shown by Paxson to have a deep impact on network performance [9]. This prompted work that subsequently resulted in the publication of RFC2525 [10], outlining potential TCP interoperability problems. Our aim with this work is to identify potential interoperability issues in RPL to increase the stability and interoperability of future implementations.

In the sensor networks domain, mechanisms like network types [1] and architectures such as Chameleon [4] can provide interoperability at the protocol header level, but not at the protocol logic level. Our experience shows that interoperability goes deeper than the protocol header level and that subtle variations at multiple system layers affect the performance of interoperable implementations.

9. CONCLUSIONS

Interoperability is of utmost importance as sensor networks go commercial, but interoperability has typically not been considered by the sensor network research community. We present two interoperable implementations of the IETF RPL IPv6 routing protocol for the two leading sensor network operating systems: Contiki and TinyOS. We demonstrate interoperability between our two implementations, but also that interoperability is not binary: subtle differences in lower layers may affect system performance in unexpected ways. This suggests that future implementations of RPL should not only be tested for interoperability, but also for performance.

Lessons learned include that it is important to maintain the least common denominator for interoperability, that network simulation is an efficient tool for interoperability testing but that the simulator must support fine-grained timing, and that interoperability must not only focus on one layer, but must also consider underlying layers. We hope that our work will be the first step towards fully interoperable sensor networks.

10. REFERENCES

[1] K. K. Chang and D. Gay. Language support for interoperable messaging in sensor networks. In *Proceedings of the 2005 workshop on Software and compilers for embedded systems*, pages 1–9, Dallas, Texas, 2005.

[2] S. Dawson-Haggerty. Design, implementation, and evaluation of an embedded IPv6 stack. Master's thesis, UC Berkeley, 2010.

[3] A. Dunkels. Full TCP/IP for 8-bit architectures. In *Proceedings of The International Conference on Mobile Systems, Applications, and Services (MobiSys)*, San Francisco, CA, USA, May 2003.

[4] A. Dunkels, F. Österlind, and Z. He. An adaptive communication architecture for wireless sensor networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, Sydney, Australia, November 2007.

[5] M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels. Making Sensor Networks IPv6 Ready. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, Raleigh, North Carolina, USA, November 2008.

[6] J. Hui and D. Culler. IP is Dead, Long Live IP for Wireless Sensor Networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, Raleigh, North Carolina, USA, November 2008.

[7] J. Ko, S. Dawson-Haggerty, J. Hui, P. Levis, D. Culler, and A. Terzis. Connecting low-power and lossy networks to the internet. *IEEE Communications Magazine*, 49, April 2011.

[8] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. Internet proposed standard RFC 4944, September 2007.

[9] V. Paxson. Automated packet trace analysis of TCP implementations. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications (ACM SIGCOMM)*, Cannes, France, 1997.

[10] V. Paxson, M. Allman, S. Dawson, W. Fenner, J. Griner, I. Heavens, K. Lahey, J. Semke, and B. Volz. Known TCP Implementation Problems. RFC 2525 (Informational), March 1999.

[11] B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao. Tiny web services: design and implementation of interoperable and evolvable sensor networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, pages 253–266, Raleigh, NC, USA, 2008.

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

[13] T. Winter (Ed.), P. Thubert (Ed.), and RPL Author Team. RPL: IPv6 Routing Protocol for Low power and Lossy Networks. Internet Draft draft-ietf-roll-rpl-18, work in progress.