Thread/OpenThread: A Compromise in Low-Power Wireless Multihop Network Architecture for the Internet of Things

Hyung-Sin Kim, Sam Kumar, and David E. Culler

Abstract

Extending an Internet subnet by connecting resource-constrained nodes (e.g., embedded sensors and actuators) over multiple wireless hops is necessary to support the future Internet of Things (IoT). RPL, the IPv6 routing standard for low-power and lossy networks, tried to achieve this goal but has not seen wide adoption in practice. As an alternative, Thread is a recently standardized low-power network protocol for IoT, driven by the Thread group, an industry consortium led by Google/Nest. We provide a comparative analysis of the technical aspects of RPL and Thread based on their specifications, explaining why using Thread, as opposed to RPL, may make sense for the future Internet. Specifically, the fundamental differences between RPL and Thread are their respective scopes and multihop network architectures, which result in Thread's unique design and advantages over RPL. Lastly, we evaluate Thread in an indoor multihop wireless testbed using OpenThread, an official open source implementation of Thread. This work serves as the first analysis of the Thread protocol in academia.

Introduction

The scope of the Internet has continuously expanded. It is now common to form a subnet with smart things, such as wearables and speakers; the Internet of Things (IoT) is happening. The current scope of IoT, however, is mostly about wirelessly connecting powerful devices, located near border routers (e.g., smartphones or WiFi access points). The natural next step for the future IoT is to extend the wireless subnet further, to include various embedded/battery-powered sensors and actuators, flexibly deployed apart from a border router.

Enabling this vision requires an interdisciplinary effort between two regimes: low-power/lossy networks (LLNs, or wireless sensor networks, WSNs) and the Internet. This effort, which actually started more than a decade ago, has extended Internet connectivity to resource-constrained embedded devices by enabling IPv6 communication over IEEE 802.15.4 low-power wireless links in 2008 (6LoWPAN) [1]. Going further, RPL, the IPv6 routing protocol for LLNs, was standardized in 2012 [2]. RPL aims to build an IPv6 subnet of thousands of resource-constrained, battery-powered devices by connecting them over *multiple wireless hops*. Although RPL has received substantial attention and spawned numerous research works for the last six years, it has many unresolved issues that preclude its widespread adoption (except for Cisco's Connected-Grid Mesh) [3, 4]. Still, practical IoT applications mostly use high-power/single-hop WiFi for "Things" or even replace Internet connectivity with another low-power/single-hop wireless connectivity, such as Bluetooth Low Energy.

In contrast, when LLN first took off two decades ago [5], researchers expected to see practical/scalable multihop wireless systems soon. Building a reliable multihop network with unattended, duty-cycling nodes, however, has been notoriously difficult. Low-power wireless networking started to receive attention again with the megatrend of IoT, but industry still has a strong perception that multihop low-power wireless networks are *unreliable* and *do not provide enough battery lifetime* [6]. The only way to make a breakthrough is to show practical evidence that it really works.

To this end, an industrial consortium, called the Thread Group, recently standardized Thread [7], a new IPv6-based low-power mesh network, with the following ideas:

1. Given that the Internet core is already globally scalable, building a low-power *subnet* with thousands of embedded devices is not only hard but also overkill.
2. In an embedded network, specifying multiple layers together makes more sense than each layer being independent.

Therefore, unlike RPL, Thread specifies *physical through network layers* and targets *modest scalability* with hundreds of embedded devices. While relying on existing standards for other layers, Thread has its own routing protocol with a clear architectural restriction: a network partition can have at most 32 *routers*, all of which must be *always on* (i.e., no radio duty cycling). Other nodes in the partition, called *leaves*, may be duty cycled, but are always one wireless hop away from an always-on router. This architectural restriction decouples routing from low-power operation, enabling Thread to build a reliable mesh among routers and simultaneously provide

Thread is a recently standardized low-power network protocol for IoT, driven by the Thread group, an industry consortium led by Google/Nest. The authors provide a comparative analysis of the technical aspects of RPL and Thread based on their specifications, explaining why using Thread, as opposed to RPL, may make sense for the future Internet.

The authors are with UC Berkeley.

Digital Object Identifier:
10.1109/MCOM.2019.1800788

0163-6804/19/$25.00 © 2019 IEEE

Since RPL first came on the scene six years ago, it has received significant attention and spawned numerous research papers. It has seen little adoption, however, except for CISCO's CGmesh. What's wrong? We unveil the underlying reasons in the light of the specification document and related work in the literature.

low-power operation for battery-powered leaf nodes. To avoid incompatibilities due to differing implementation choices, Google/Nest, a leading member of the Thread Group, has released an official open source implementation of Thread, called OpenThread.

With its open source implementation, focus on practical usage, and considerable driving force from industry, Thread is worth investigating and has the potential to be applied in practice. As a recent standard, however, its technical aspects have not yet been investigated in the research community. As a stepping stone, this article explores Thread's routing aspects through *a comparative analysis with RPL* to demonstrate why it may make sense for the future IoT. We also discuss future steps to put the Internet in the *Internet* of Things. In doing so, our goal is not to conclude which protocol is better, but to introduce Thread as a new low-power IoT protocol that is interesting enough to investigate for the future IoT.

The RPL Standard

This section presents a brief overview of RPL's design goals and features.

Design Goal and Architecture

The RPL routing protocol was designed to accomplish three challenging goals together: scalability, reliability, and resource/energy efficiency. Specifically, it aims to construct an IPv6 subnet with *thousands* of resource-constrained nodes over multiple low-power wireless hops that simultaneously provides *reliable data delivery* and *several years of battery lifetime*. In addition, RPL focuses on being solely a routing protocol, *without limiting other node characteristics*; any node can be a router and/or battery-powered (i.e., easy deployment). The high goal and the flexible architecture, however, result in a strict requirement for protocol design: RPL routing should provide both reliability and energy efficiency for all of numerous nodes in the case that they are all routers and battery-powered (the typical setting in WSN research).

Design Choices and Features

To satisfy the requirements, RPL made two important assumptions and design choices in light of prior work on WSNs [8].

Upward-Focused Routing: *"Multipoint-to-point (MP2P) is a dominant traffic flow in many LLN applications."* — Internet Engineering Task Force (IETF) RFC 6550 [2]

The first assumption is that traffic in LLNs mostly goes upward, such as data gathering from embedded sensors through border routers. Therefore, while it provides bidirectional (both up/downward) routes, RPL focuses on reliable upward routes. To this end, it builds a quasi-forest routing topology, called destination-oriented directed acyclic graph (DODAG), rooted at a border router. Each node selects a parent node as the next hop of its upward route, based on the *distance vector from the border router*, while setting the downward route simply as the reverse of the upward route. The distance vector-based path cost for the upward route, called RANK, is propagated by broadcasting DODAG Information Object (DIO) messages.

Data-Reactive Route Update: *"Data traffic can be infrequent" and "Typical LLNs exhibit variations in physical connectivity that are transient and innocuous to traffic."* — IETF RFC 6550 [2]

With this assumption, making the control plane maintain a routing topology that is constantly up to date with the physical topology can waste energy. Therefore, RPL detects physical connectivity changes, which are supposed to be transient and infrequent, *reactive to data transmissions* while minimizing control packet transmissions (by using Trickle Timers [9]). Although the path cost design is decoupled from the RPL standard, the concept of data-traffic-reactive route update naturally ends up with the use of ETX (expected transmission count) as the path cost (RANK). ETX increases when more link layer retransmissions are required to deliver a data packet, indicating that the wireless link becomes unstable.

Additional Features and Not-Included Aspects: In addition to the above design choices, RPL includes the multi-instance feature to support heterogeneous traffic. Given that each RPL instance builds a separate routing topology with its own quality of service (QoS) and routing metric, the multi-instance feature enables different traffic to go through different routes to reach the same destination. As a result, a routing table can have multiple entries for a single node depending on the number of instances the node joins.

However, as a routing protocol in the layered Internet architecture, RPL does not specify neighbor management (a list of directly reachable nodes) but instead relies on an external mechanism (e.g., the IPv6 standard Neighbor Discovery, ND). RPL does not require anything for other network layers.

Shortcomings of RPL

Since RPL first came on the scene six years ago, it has received significant attention and spawned numerous research papers. It has seen little adoption, however, except for Cisco's CGmesh. What's wrong? We unveil the underlying reasons in light of the specification document and related work in the literature. For more details, we strongly recommend reading recent surveys of RPL, such as [3, 4].

The Flip Side of the Design Choices

First of all, the two fundamental design choices of RPL turn out to be inappropriate for real IoT use cases.

Upward-Focused Routing: Although upward traffic is dominant in LLNs, *the need for downward traffic delivery in IoT use cases is nontrivial*. Downward traffic is needed, for example, for actuation commands, firmware updates, acknowledgment (ACK)-based transport layer protocols like TCP, and application-layer ACKs. In addition, some applications, such as electronic shelf labeling, generate more downward traffic than upward traffic [10].

However, *the upward-focused design of RPL makes downward routing unreliable and unscalable* [4]. Specifically, when physical wireless connectivity changes, the downward route is very slowly updated, after upward data transmission failures (ETX increase), the upward route update, and a successful control message (Destination

Advertisement Object, DAO) transmission toward the new upward route. Meanwhile, many downward data packets can be lost. Furthermore, RPL's downward routing should select either of two modes of operation (MOP): storing mode (table-driven routing) or non-storing mode (source routing), which suffer from memory overhead or routing header overhead, respectively. The overhead prevents RPL from constructing downward routes with thousands of resource-constrained nodes.

Data-Reactive Route Update: Although using data traffic for route updates significantly reduces control overhead, this incurs stability and reliability problems. Note that data packets are not for assisting the network maintenance but mainly for delivering important information to a destination. When connectivity changes, however, RPL cannot detect it before a number of data packets are sacrificed, degrading reliability.

In addition, RPL's representative routing metric, ETX, has shown many problems when wireless link conditions become challenging. In contrast to the standard's assumption, data traffic in LLNs is not necessarily infrequent. Some applications, such as structural monitoring and anemometry [11], do require frequent data reporting for meaningful analysis. In a large-scale network with thousands of nodes, nodes near the border routers should relay heavy traffic. Heavy traffic causes data packet loss due to congestion and hidden terminals. Wireless interference such as WiFi also causes nontrivial packet loss. In this case, RPL's routing topology severely churns [12]. This is because RPL tries to change routes since ETX becomes bad, but changing routes does not resolve the wireless link problems caused by wireless interference and traffic load.

Although the above design choices have clear weaknesses, there have been no outstanding alternatives that provide scalability, reliability, and energy efficiency within a flexible network architecture where all nodes can be routers and battery-powered.

Simultaneous Complexity and Ambiguity

The key idea of having a protocol standard is that all implementations following the standard should be interoperable with each other and provide reasonable performance. A standard should have fine-grained guidelines for all necessary features while excluding any redundant feature.

The RPL standard, however, has unnecessary features, such as a multi-instance/QoS metric (never investigated on embedded devices) and routing messages overlapping IPv6 ND features. This only increases implementation complexity and memory overhead without a clear benefit. A protocol for *resource-constrained* embedded devices should avoid such *over-specified* features.

Simultaneously, RPL *under-specifies* many necessary features. Specifically, although an embedded network implementation should be vertically integrated, RPL has few guidelines for inter-layer operation. Although it makes sense that a routing standard excludes ND functionality, RPL unfortunately does not have a complete suggestion for an external ND mechanism or interoperability with IPv6 ND.

The complexity and ambiguity of RPL result in various different implementations with different implementation choices and different features, degrading interoperability and performance.

Thread: A Multi-Layer Standard with Restricted Architecture

This section introduces Thread, an alternative to RPL as a low-power multihop IoT network protocol. Thread does not share RPL's design goals and architecture. Rather, compared to RPL, Thread loosens the design goals and promotes a compromise network architecture, as follows.

Energy Efficiency. *"Routers are not designed to sleep" and "Sleepy end Devices (SEDs) are host devices. They communicate only through their parent router and cannot forward messages for other devices."* — Thread 1.1.1 Specification [7]

Scalability. *"It is designed specifically for Connected Home applications," "Home networks vary from several devices to hundreds of devices communicating seamlessly," and "There can be a maximum of 32 Routers in a thread Network partition."* — Thread 1.1.1 Specification

Compared to RPL, the targeted subnet size is reduced from thousands to hundreds (including leaf nodes), and there is a clear bound on the number of routers. The goal of *modest scalability* is not just a compromise, however, but avoids overkill for a *subnet*. In addition, routers should be powered enough to be always on; duty-cycled (low-power) nodes are explicitly decoupled from routing and focus on energy efficiency. This allows routers to focus on the reliability aspect and battery-powered leaf nodes to focus on the low-power aspect. The "always-on router" restriction is reasonable in indoor environments, which have abundant outlets, as well as outdoor settings with solar power, for example.

Furthermore, considering the nature of an embedded network as a vertical silo, Thread specifies multiple layers (physical to network); while proposing its own routing design, Thread also specifies other networking aspects, such as duty-cycling medium access control (MAC) (L2), addressing, and commissioning. This resolves both complexity and ambiguity problems. Specifically, Thread's design choices are different from those of RPL, as shown in Table 1.

Routing Topology: Two-Tier, Asymmetric, Full Mesh

In contrast to RPL, Thread provides a two-tier routing topology considering the heterogeneous power capabilities of nodes.

The topology among routers is not a forest but a *full mesh*, given that the routers are always on. That is, each router has path cost (distance vector) not just to the border router but *to all routers*. Each router can reach any destination by selecting a next hop among multiple candidates *by itself*. In contrast to RPL, for which a downward route is passively determined by an upward route (as its reverse), Thread sets each path independently, which can cause a bidirectional route to be *asymmetric* in various cases. In this way, Thread provides reliable routes not only for upward traffic, but also for any-to-any traffic.

In contrast, a battery-powered leaf node selects a single router as its parent and focuses only on

RPL uses Trickle for DIO transmission to provide both minimal control overhead and fast route recovery, setting the steady-state interval between DIO transmissions to several minutes. Thread also uses Trickle to set the advertisement message interval.

	RPL [2]	Thread [7]
Scalability	Thousands of nodes	Hundreds of nodes (up to 32 routers)
Radio duty cycling	Out of scope	Always-on routers, duty-cycling leaf nodes (listen-after-send)
Routing topology	DODAG (quasi-forest)	Two-tiered mesh
Upward route selection	Direct	Direct
Downward route selection	Indirect (passive)	Direct
Bidirectional route symmetry	Symmetric	Asymmetric
Link cost	Out of scope (typically ETX)	Quantized SNR
Link cost in a control packet	Not included	Both incoming/outgoing link costs, from/to each router (up to 32)
Path cost	Out of scope (typically accumulated ETX)	Accumulated quantized SNR
Path cost in a control packet	Upward path only	Path toward each router (up to 32)
Physical connectivity tracking	Slow (> several minutes), mainly relying on the data plane	Fast (< 1 minute), mainly relying on the control plane
Routing table	Upward or all up/downward entries	To any router, up to 32 entries
Multi-border router support	Multiple DODAGs (one DODAG per border router)	A single network partition including all border routers
Addressing	Out of scope	16-bit address, encoding parent/child relationship
Neighbor discovery	Out of scope	As in the specification
Commission	Out of scope	As in the specification
Open implementation	TinyRPL, ContikiRPL, RIOT-RPL (academia-driven)	OpenThread (industry-driven)

Table 1. Detailed comparison between RPL and Thread.

maintaining connectivity with the parent. Without any neighbor/routing information, its routing strategy is to simply rely on its parent: it sends/receives all packets to/from the parent. This design choice, decoupling battery-powered nodes from route management, can improve battery lifetime for leaf nodes by reducing control overhead.

Routing Packet: All-in-One

A Thread router broadcasts *advertisement* messages to update the routing metric at other nodes. In contrast to RPL's DIO message containing only one path cost to the border router, Thread's advertisement message *has link costs and path costs to all 32 routers*, which significantly restricts control packet overhead while forming a full mesh. Furthermore, given that the advertisement message provides link cost (not only path cost), the message supports both routing and neighbor management; *Thread tightly integrates routing with ND*, in contrast to RPL, which focuses on routing. To contain all 32 entries in one packet, Thread represents bidirectional link cost, each of incoming and outgoing link costs with 2 bits and path costs with 4 bits, resulting in a 1-byte payload for each entry. In addition, Thread recognizes each entry's address with its location in the advertisement message payload (i.e., address is its index), resulting in no additional space for addressing.

In contrast, a battery-powered leaf node does not send/receive advertisement messages, focusing solely on low-power operation.

Route Update: Back to Control Plane

RPL uses Trickle [9] for DIO transmission to provide both minimal control overhead and fast route recovery, setting the steady-state interval between DIO transmissions to several minutes. Thread also uses Trickle to set the advertisement message interval. Given that routers in Thread are not low-power, however, it sets the interval to 32 s in steady state, *much shorter* than RPL. With such frequent advertisements, Thread can periodically refresh the entire mesh route relying on control packets rather than (infrequent) data transmissions. By using control packets to detect changes in physical connectivity, Thread is closer to classic routing than RPL.

Each (low-power) leaf node periodically wakes up and checks the reachability of its parent by sending data request (keepalive) messages.

Path Cost: Accumulated SNR

Although ETX, the inverse of packet reception ratio (PRR), is a very intuitive metric that can be obtained from data transmission experience without any additional overhead, it is subject to the issues described earlier, and is additionally unreliable in LLNs. Given that PRR does not decrease linearly with received signal strength indicator (RSSI) but suddenly drops from > 90 percent to < 10 percent at a certain RSSI threshold (e.g., –87 dBm) [13], ETX can finely distinguish link quality around that threshold. However, *it cannot distinguish a very robust link from possibly fragile links*. For example, when choosing an upward route, two candidate links may both have a current ETX of one 1 (the best quality), but one link may have an RSSI of –50 dBm and the other –80 dBm. Although the –80 dBm candidate link currently has good PRR, it is likely to become bad (below –87 dBm) in the future due to fluctuations in link quality. The –50 dBm candidate link is substantially more reliable, but the ETX information cannot tell the difference.

In contrast, Thread uses signal-to-noise ration (SNR) (link margin) for path cost one-way link quality and link cost. Given that SNR is well known to highly fluctuate, it is averaged and quantized for stability, as shown in Table 2. Thread takes the maximum of incoming/outgoing link cost to represent the cost of a bidirectional link, resolving link asymmetry. The path cost is the link cost accumulated along a route, increasing with hop count. This SNR metric-based routing provides robustness while possibly increasing hop count (forwarding overhead), for example, by selecting a −50 dBm node rather than a −80 dBm node, which is a reasonable trade-off since routers are free from energy constraints. Another advantage is that the routing topology can be stable even when the link layer suffers from data packet loss due to wireless congestion and/or interference.

A leaf node only holds the link cost for its parent, with no routing metric; it focuses on the direct link, not route length.

Next Hop Selection: Table- and Address-Driven

RPL provides non-storing and storing modes for next hop selection of downward traffic, both of which have scalability issues: packet and memory overhead, respectively.

In contrast, Thread uses the routing table to find a route only among routers, which bounds memory overhead to at most 32 entries. To provide a route to a leaf node, Thread uses both the routing table and an addressing technique. Given that a leaf node exchanges packets with others only through its parent router, the goal is to find a route from the source to the parent router. To this end, Thread encodes the parent router's address in a leaf node's address: among the 16-bit IEEE 802.15.4 short address field, the first 6 bits are used for the parent router's address and the other 10 bits are used for a leaf node's identifier. In this way, when a node is willing to send a packet to a leaf node, it extracts the parent router information from the leaf node's address and finds a route toward the parent router based on the routing table.

Duty-Cycling: Listen-After-Send

As a multi-layer standard, Thread specifies a duty-cycling MAC: *listen-after-send* in the IEEE 802.15.4 standard. Taking advantage of always-on routers, the duty-cycling protocol focuses on communication between duty-cycled leaf nodes and always-on routers; there is no direct communication between leaf nodes. Unlike most duty-cycling protocols, which assume all nodes are duty-cycled, Thread's leaf node does not have to check if the parent router is awake, enabling simple operation. Specifically, when a leaf node has a packet to send to the parent, it simply wakes up, sends, and sleeps. Downstream packets destined to a leaf node are queued at that node's parent. To receive packets, the leaf node periodically wakes up, sends a data request packet to its parent, and listens for a short interval. Upon receiving the data request packet, the parent node sends queued packets to the leaf node. This data request packet is also used for checking connectivity with the parent. The duty-cycling mechanism enables control of leaf node energy consumption via the sleep interval [14].

Link margin (SNR)	Link quality	Link cost
> 20 dB	3	1
> 10 dB	2	2
> 2 dB	1	4
≤ 2 dB	0	Infinite

Table 2. Link margin conversion to one-way link quality in Thread.

Given that RPL can be combined with any link layer protocol, such as Time-Synchronized Channel Hopping (TSCH) in the IEEE 802.15.4e standard, it should be investigated whether the specific link-layer protocol of Thread is good enough in real environments.

Multiple Border Routers

In practical use cases, any node can fail for various reasons; LLNs should avoid relying heavily on a single node. Therefore, even when a single border router can connect all nodes, deploying multiple border routers is necessary to avoid a single point of failure. Given that RPL's DODAG is rooted at one border router, however, RPL needs to form multiple DODAGs to include multiple border routers. In addition, RPL allows a node to join only one DODAG (per instance), resulting in two disjoint DODAGs rather than a single topology fully utilizing all potential connectivity. Although the RPL standard allows multiple border routers to coordinate to form one DODAG with a single "virtual" root, it provides no specific guidelines to realize this feature, so no RPL implementation actually supports it.

Because Thread forms a full mesh instead of DODAGs, it can include all nodes and multiple border routers in a single routing topology. This significantly reduces the practical management burden.

OpenThread: A Complete, Official, and Open Implementation of Thread

By design, Thread solves many problems in RPL. In LLNs, however, the devil has always been in the implementation details. In particular, RPL has suffered from incomplete implementations [4]. To address the issue, Google/Nest, a leading member of the Thread Group, has open-sourced an official/complete implementation of Thread, called OpenThread (https://openthread.io). OpenThread is a complete network implementation including all network layers. It has an event-driven kernel to run by itself, but can also be ported on an embedded operating system.

To evaluate OpenThread's performance, we ported OpenThread on RIOT-OS, an open source embedded operating system, and used the Hamilton embedded platform [15]. We first confirm energy consumption of leaf nodes in two cases:

1. Periodic sensing and sending to the parent
2. Periodic reception from the parent, each with 10 s between packets and good link quality

In each case, a Hamilton device acting as a leaf node consumes 30 μA and 21 μA, respectively, resulting in lifetimes of 5.3 and 7.6 years, respectively, with a 1400 mAh battery.

Unlike most duty-cycling protocols, which assume all nodes are duty-cycled, Thread's leaf node does not have to check if the parent router is awake, enabling simple operation. Specifically, when a leaf node has a packet to send to the parent, it simply wakes up, sends, and sleeps.

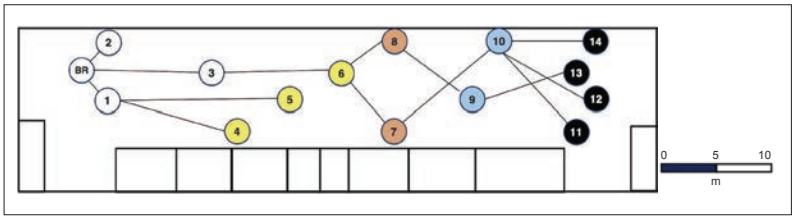

Figure 1. Testbed topology with a snapshot of 5-hop upward routing paths chosen by OpenThread when using transmission power of −8 dBm. Nodes with the same hop count have the same color.

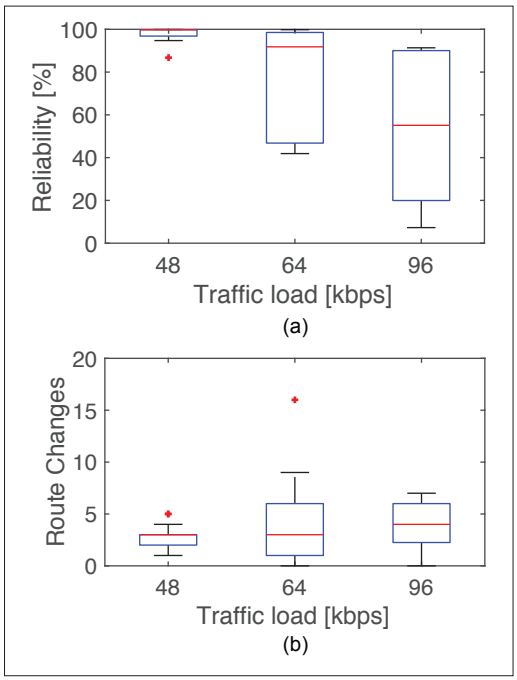

Figure 2. Performance of OpenThread in the testbed as traffic load increases.

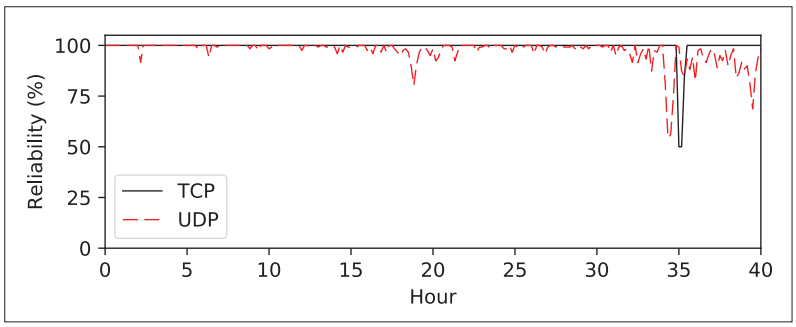

Figure 3. Reliability of TCP and UDP over OpenThread for 40 hours.

To evaluate routing, we constructed a multihop testbed with 15 Hamiltons (including the border router) in an office (Fig. 1). Each Hamilton acts as a router and sends packets periodically to the border router using its IEEE 802.15.4 radio supporting a data rate of 250 kb/s. Figure 2 plots OpenThread performance according to the total input traffic load across all nodes. Figure 2a shows that OpenThread does experience more packet loss as traffic load increases. Figure 2b, however, shows that even with severe packet loss, OpenThread maintains the routing topology and does not intensify the problem. OpenThread provides the opportunity to systematically investigate many aspects of Thread, such as link/path cost, load balancing, commissioning, and timely propagation of network information.

What Else Is Missing?

Neither Thread nor RPL explicitly specifies the transport- or application-layer protocols used on top of IPv6. RPL, with its focus on upward routes, favors protocols that require a unidirectional flow of packets leaving the LLN, such as UDP-based protocols. In contrast, Thread provides good support for both upward and downward routes. Therefore, it has promise to support protocols with *bidirectional packet flow*. For example, TCP could be used to collect data from sensor readings, with data packets sent in one direction and TCP ACKs in the other direction. Using TCP in LLNs is interesting because it would improve interoperability with the existing Internet services, which primarily use TCP/IP [11].

As a proof of concept, we ran TCP and UDP over OpenThread in the same testbed (Fig. 1) simultaneously for 40 hours. Nodes 11, 12, 13, and 14 generate a message every 10 s. Nodes 11 and 14 send the reading over UDP, and nodes 12 and 13 use TCP. Note that in our testbed environment, human activities with various wireless devices during daytime incur significant wireless interference. Figure 3 depicts the reliability of each protocol for the duration of the experiment, as the average of both nodes running each protocol (TCP or UDP). Our results demonstrate that whereas UDP does not provide reliable data transport, TCP increases reliability to nearly 100 percent over multihop, lossy wireless links. TCP's reliability drops to 50 percent briefly at Hour 35 because Node 13 lost connectivity. This may have been due to a routing failure in OpenThread; the current OpenThread implementation does occasionally experience disconnection in harsh wireless environments.

In addition, although Thread provides a fairly stable routing topology in the presence of link errors, it should also reduce these errors *as a multi-layer standard*. Given that wireless interference, such as Bluetooth and WiFi, is significant, Thread's link layer protocol should be more robust. Applying TSCH could be an option.

Conclusion

We introduce Thread, showing its potential to expand Internet connectivity to resource-constrained embedded devices over multiple wireless hops. We do so via a comparative analysis with RPL, the de facto IoT routing standard. Thread has significant driving force, with an active industrial consortium of more than 100 members (the Thread Group), and with commercial Thread-enabled products on the market from companies such as Google/Nest and NXP. OpenThread opens the door for researchers to investigate and even improve Thread's technical design. Although we have shown the potential of Thread, thorough experimental studies are needed to reveal its behavior and performance in detail. We ask the research community to actively participate in this exciting move toward the future IoT.

Importantly, the RPL "standard" is flexible (ambiguous at the same time) and leaves many things for "implementation" choices. Many spe-

cific design choices of Thread, such as its two-tier architecture, control message-based route update, path cost, and multiple border routers, can possibly be implemented in the context of RPL without violating its standard, although implementation complexity could be problematic. Investigation of Thread and OpenThread may uncover the best choices for a RPL implementation.

ACKNOWLEDGMENT

This research is partly supported by the Department of Energy Grant No. DE-EE0007685, California Energy Commission, Intel Corporation, and the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1752814. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We are thankful to Jonathan Hui, principal software engineer at Google/Nest, for his valuable feedback.

REFERENCES

[1] J. W. Hui and D. E. Culler, "Extending IP to Low-Power, Wireless Personal Area Networks," *IEEE Internet Computing*, no. 4, 2008, pp. 37–45.
[2] T. Winter *et al.*, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," IETF RFC 6550, Mar. 2012.
[3] O. Iova *et al.*, "Rpl: The Routing Standard for the Internet of Things... or Is It?" *IEEE Commun. Mag.*, vol. 54, no. 12, Dec. 2016, pp. 16–22.
[4] H.-S. Kim *et al.*, "Challenging the ipv6 Routing Protocol for Low-Power and Lossy Networks (RPL): A Survey," *IEEE Commun. Surveys & Tutorials*, vol. 19, no. 4, 2017.
[5] D. Estrin *et al.*, "Next Century Challenges: Scalable Coordination in Sensor Networks," *Proc. 5th Annual ACM/IEEE Int'l. Conf. Mobile Computing and Networking*, 1999, pp. 263–70.
[6] C. A. Boano *et al.*, "IoTbench: Towards A Benchmark for Low-Power Wireless Networking," *1st Wksp. Benchmarking Cyber-Physical Networks and Systems*, 2018.
[7] I. Thread Group, "Thread 1.1.1 Specification," Feb. 2017.
[8] O. Gnawali *et al.*, "Collection Tree Protocol," *Proc. 7th ACM Conf. Embedded Networked Sensor Systems*, 2009, pp. 1–14.
[9] P. Levis *et al.*, "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," *Proc. USENIX/ACM NSDI*, 2004.
[10] H.-S. Kim, J. Ko, and S. Bahk, "Smarter Markets for Smarter Life: Applications, Challenges, and Deployment Experiences," *IEEE Commun. Mag.*, vol. 55, no. 5, May 2017, pp. 34–41.
[11] S. Kumar *et al.*, "Tcplp: System Design and Analysis of Full-Scale TCP in Low-Power Networks." arXiv preprint arXiv:1811.02721, 2018.
[12] D. Han and O. Gnawali, "Performance of RPL Under Wireless Interference," *IEEE Commun. Mag.*, vol. 51, no. 12, Dec. 2013, pp. 137–43.
[13] K. Srinivasan *et al.*, "An Empirical Study of Low-Power Wireless," *ACM Trans. Sensor Networks*, vol. 6, no. 2, 2010, p. 16.
[14] T. Schmid *et al.*, "Disentangling Wireless Sensing from Mesh Networking," *Proc. 6th ACM Wksp. Hot Topics in Embedded Networked Sensors*, 2010, p. 3.
[15] H.-S. Kim *et al.*, "System Architecture Directions for Postsoc/32-Bit Networked Sensors," *Proc. ACM SenSys*, 2018.

BIOGRAPHIES

HYUNG-SIN KIM (hs.kim@berkeley.edu) is a postdoctoral researcher of electrical engineering and computer science (EECS) at the University of California (UC) Berkeley and part of the RISE Lab and the Building Energy Transportation Systems (BETS) group. His research interests include development of networked embedded systems for the Internet of Things.

SAM KUMAR (samkumar@berkeley.edu) is a Ph.D. student in EECS at UC Berkeley. He is part of the RISE Lab, and also part of the BETS group. His research interests include systems, networking, IoT, and security.

DAVID E. CULLER [F] (culler@berkeley.edu) is the Friesen Professor of EECS at UC Berkeley. He is a member of the National Academy of Engineering and a Fellow of the ACM. He was co-founder and CTO of Arch Rock Corporation and serves on several corporate technical advisory boards.

OpenThread opens the door for researchers to investigate and even improve Thread's technical design. Although we have shown the potential of Thread, thorough experimental studies are needed to reveal its behavior and performance in detail. We ask the research community to actively participate in this exciting move toward the future IoT.