A Measurement Study of TCP over RPL in Low-power and Lossy Networks

Hyung-Sin Kim, Heesu Im, Myung-Sup Lee, Jeongyeup Paek, and Saewoong Bahk

Abstract: **Low-power and lossy networks (LLNs) comprised of thousands of embedded networking devices can be used in a variety of applications, such as smart grid automated metering infrastructures (AMIs) and wireless sensor networks. Connecting these LLNs to the Internet has even greater potential, leading to the emerging concept of the Internet of Things (IoT). With the goal of integrating LLNs into IoT, the IETF has recently standardized RPL and 6LoWPAN to allow the use of IPv6 on LLNs. Although there already exist several studies on the the performance of RPL and embedded IPv6 stack in LLN, performance measurement and characterization of *TCP over RPL* in multihop LLNs is yet to be studied. In this article, we present a comprehensive experimental study on the performance of *TCP over RPL* in an embedded IPv6-based LLN running over a 30-node multihop IEEE 802.15.4 testbed network. Our results and findings are aimed at investigating how embedded TCP interoperates with common Linux TCP and underlying RPL (and vice versa), which furthers our understanding of the performance trade-offs when choosing TCP over RPL in IPv6-based LLNs.**

Index Terms: **6LoWPAN, automated metering infrastructure (AMI) network, IEEE 802.15.4, Internet of Things (IoT), IPv6, low-power lossy network (LLN), RPL, TCP.**

I. INTRODUCTION

LOW-POWER and lossy networks (LLNs) comprised of thousands of embedded networking devices can be used in a variety of applications including smart grid automated metering infrastructures (AMIs) [1]–[4], smart city management, home and building automation, wireless sensor networks, and the newly proposed concept of Industry-4.0. Recently, LLNs have started to employ open and standardized IP/IPv6-based architecture to integrate as part of the Internet. This approach enables LLN to be more interoperable, flexible and versatile, leading to the emerging concept of the Internet of Things (IoT).

Combining LLNs with the standard Internet protocol is essential for various useful applications. Taking smart grid as an example, its network requirements include unprecedented scale, multivendor interoperability, and use of low-cost communication devices [3]–[6]. Furthermore, since information networking technology is fast changing, seamless integration with a more slowly evolving smart grid requires a layered architecture to facilitate upgrade of link and physical layer technologies. This is one of the reasons why the National Institute of Standards and Technology (NIST) approved IP as a smart grid standard [7]. We believe that this trend is promising and will continue for the foreseeable future. That is, industrial LLN solutions will adopt well-tested standard Internet protocols for scalability, compatibility, security, and development cost reasons [6], [8].

Integration of embedded devices into the Internet introduces several new challenges since existing Internet technologies and protocols have not been designed for this class of devices. In fact, LLNs typically have different traffic patterns, low throughput, high packet loss, and frequent topology changes, among other characteristics that pose integration challenges. To support expanding smart grids and other LLN applications, the IETF has recently standardized protocols such as RPL [9] and 6LoWPAN [10]. They extend Internet technologies to constrained devices, aiming to move LLNs closer to IoT where end-to-end IP-based network connectivity is possible.

RPL is an IPv6 routing protocol for LLNs, designed for resource constrained embedded devices to meet the requirements of a wide range of LLN applications [9]. RPL constructs a tree-like routing topology rooted at an LLN border router (LBR), and supports bi-directional IPv6 communication. 6LoWPAN is an adaptation layer that defines encapsulation, header compression, fragmentation, and other mechanisms. It allows IPv6 to operate efficiently over IEEE 802.15.4, the representative link layer standard for LLNs [10], [11].

IoT based smart grid AMI system is the main example scenario that our work is targeting [3]. Several prior studies have investigated the feasibility and applicability of LLN and RPL for smart grid [2], [12]. Cisco's field area network (FAN) is a good example of a commercial solution for AMI system developed over LLN by the industry [1]. It consists of connected grid network management server (CG-NMS), connected grid field area router (CGR), and connected grid mesh network (CG-Mesh) software for smart grid endpoints. FAN is based on IPv6 and uses IEEE 802.15.4g/e at the PHY and MAC layer to form an LLN. Above the link layer, it uses 6LoWPAN, RPL, and IPv6 to provide internetworking support for smart metering endpoints, end-to-end UDP/TCP communication at the transport layer, and coAP simple management protocol (CSMP) for network management. Cisco's CG-Mesh system provides initial evidence that the use of IPv6 over RPL/6LoWPAN in an IEEE 802.15.4 net-

This paper is specially handled by EICs and Division Editor with the help of three anonymous reviewers in a fast manner.

Saewoong Bahk is the corresponding author.

Hyung-Sin Kim, Heesu Im, Myung-Sup Lee, and Saewoong Bahk are with the Department of Electrical Engineering, Seoul National University, Seoul, Republic of Korea. email: {hskim, hsim, mslee}@netlab.snu.ac.kr, sbahk@snu.ac.kr.

Jeongyeup Paek is with the School of Computer Science and Engineering, Chung-Ang University, Seoul, Republic of Korea. email: jpaek@cau.ac.kr.

This work was supported in part by a grant to Bio-Mimetic Robot Research Center Funded by Defense Acquisition Program Administration and by Agency for Defense Development (UD130070ID), in part by the ICT R&D program of MSIP/IITP [B0126-15-1017, Spectrum Sensing and Future Radio Communication Platforms], and in part by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (201401670001).

Digital object identifier 10.1109/JCN.2015.000111

work may be feasible in large scale LLNs. It is also part of growing industry effort that invests in LLN solutions to facilitate IoT.

There have been a number of performance evaluation studies on RPL [2], [13]–[16] and light-weight embedded IPv6 stack [17]–[19] in LLNs. However, there has not been an experimental study of TCP over RPL in a multihop[1] LLN network with IEEE 802.15.4 links. It is well known that TCP's congestion control mechanism performs poorly in wireless and lossy environments and generates high overhead. This is the main reason why TCP has not been tightly associated with LLN even though it is the dominant transport protocol in today's Internet.

However, effective support of TCP is essential for achieving interoperability of the IP stack in LLNs, both for performance and legacy compatibility reasons. In the latter, smart grid applications support encapsulation of their protocols in TCP/IP, and some protocols even define a dedicated mode of operation over the TCP/IP (e.g., IEC 60870-5-104 variation of the IEC 60870-5-101 SCADA protocol) [20]. Not only because UDP does not provide end-to-end reliability, but for compatibility reasons, TCP is likely to continue to play a significant role in LLNs as a part of IoT.

For the aforementioned reasons, it is important to experimentally study the performance of TCP over RPL and reciprocally RPL under TCP, and understand their interaction in multihop LLNs. To facilitate meaningful measurement that advances our understanding of how to engineer such systems, our study uses a controlled LLN testbed where low-power embedded devices run TCP over RPL internetworked by IPv6. Empirical results and the insights obtained from our experiments will contribute toward a better understanding of how to achieve a seamless Internet of Things over LLNs.

The contributions of this work are summarized as follows:

- We present a comprehensive measurement-based study on the TCP over RPL performance in LLN using a multihop testbed of 30 embedded sensor nodes. We perform experiments between a common Linux TCP host and multiple embedded TCP hosts, and show how design choices made in embedded TCP influences the results differently from using two common TCP hosts, and also differently depending on the flow direction. We analyze the reasons for the differences, and reveal that TCP incurs significant throughput unfairness among nodes in multihop LLNs. These results are aimed at explaining why TCP over RPL had unsatisfactory performance in smart grid domain.
- We investigate how RPL interoperates with the transport layer protocols, i.e., whether TCP influences the behavior of RPL differently from UDP. Our findings indicate that although there are no significant differences in terms of routing parent changes or RPL control overhead, RPL's inability to consider traffic load balancing may adversely affect the performance of TCP.
- We identified several implementation details problematic in the latest implementations of RPL and TCP for embedded devices, and rectified them in order to focus our work on the high-level characteristics of embedded TCP and RPL. Moreover, we summarized the distinct characteristics of embedded TCP compared to common Linux TCP.
- Based on the observed results, we introduce some research challenges to improve the performance of TCP over RPL in multihop LLNs.

[1] There is one known study that uses multihop in a single line topology and experiments a single stream of traffic. We distinguish our work from that since we use multihop tree topology with 30 concurrent streams of traffic. Furthermore, that work did not employ RPL routing.

II. RELATED WORK

A number of works have investigated the performance of RPL [9] on IEEE 802.15.4 network in various network configurations. Ko *et al.* experimentally evaluated the performance of RPL and 6LoWPAN using TinyRPL and BLIP implementations in TinyOS [13]. They have shown that performance is similar to CTP (collection tree protocol), the de facto data collection protocol in TinyOS, while having the benefits of an IPv6-based architecture. They also evaluated the performance of ContikiRPL and TinyRPL over uIPv6 and BLIP, respectively [21]. In wireless sensor networks where Contiki and TinyOS are the popular underlying operating systems, they showed that the two embedded IP stack implementations are interoperable but parameter selection and implementation details can have significant effect on the performance of a network consisting of both implementations. However, both of these works neglected TCP in their evaluations.

Kermajani *et al.* presented simulation results on the network convergence process of RPL over IEEE 802.15.4 multihop networks and investigated improvements and trade-offs [22]. Herberg *et al.* compared the RPL protocol with 6LoWPAN ad hoc on-demand distance vector routing (LOAD) using NS-2 simulation [15]. They showed that LOAD may incur less overhead than RPL if the traffic pattern is bi-directional. Clausen *et al.* provided a critical evaluation of RPL with respect to limitations and trade-offs, and proposed suggestions for improvements [14]. What is lacking in these works is a measurement-based evaluation of embedded TCP over RPL in a multihop LLN testbed that incorporates system properties such as wireless interference and protocol overhead.

RPL has drawn significant attention in the smart grid domain and several works have studied the applicability and performance of RPL in this context [3]. Ancillotti *et al.* presented an overview of the role of RPL for smart grid communication and studied ContikiRPL performance using Cooja simulation [2]. More recently, Ancillotti *et al.* proposed a cross-layering design for RPL, which provides enhanced link estimation and efficient management of neighbor tables [23]. They used AMI as a case study and employed Cooja emulator to evaluate their proposal. Wang *et al.* discussed the use of RPL for AMI in smart grid and compared RPL with AODV routing using NS-2 simulation [24]. Bressan *et al.* discussed the deployment of a smart monitoring system using LLNs and performed RPL simulations for a smart grid scenario [25]. Although these works provide good overview of how RPL is applicable to the smart grid, they are simulation studies and do not provide evidence of protocol behavior on real devices. Gungor *et al.* measured IEEE 802.15.4 link quality in real power grid environments and discussed associated opportunities and challenges [12]. However, this was only for the link

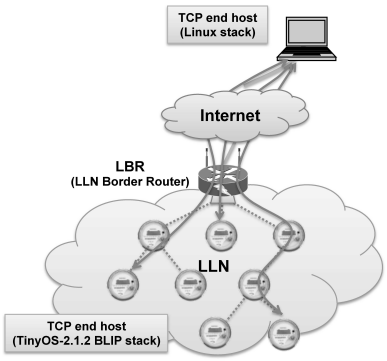


Fig. 1. Example scenario of IoT LLN connected to the Internet via LBR.

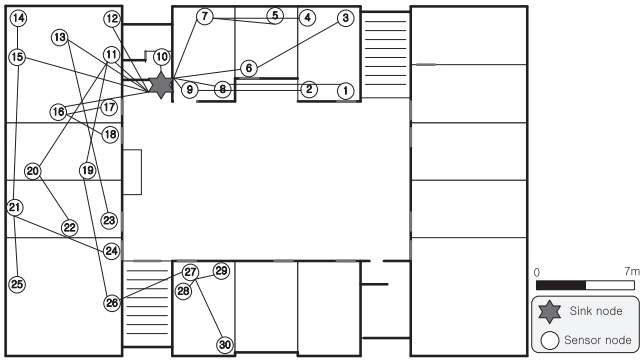


Fig. 2. Testbed topology map with snapshot of routing path given by RPL.

layer and did not discuss RPL nor TCP.

Duquennoy *et al.* presented TCP experiment results on LLN on top of the novel *'burst forwarding'* scheme that they propose to enhance the data throughput [26]. However, it has a few major differences from our work. First, this work runs TCP between two PC hosts using PC–LLN–PC topology. Thus, both the TCP sender and the receiver were common full-scale TCP, not embedded TCP, tunneled over LLN using their proposed burst forwarding scheme. Furthermore, the LLN setup was in a single line topology testing a single stream of data. Lastly, there is no mention about what routing protocol is used in the LLN or how the routing topology was constructed. In contrast, our work uses embedded TCP on one side of the connection, uses multihop tree topology with possibly multiple children nodes per parent, and tests 30 streams of data (one stream per node) simultaneously. Furthermore, our intention is to investigate the behavior and influence (or lack thereof) between RPL and embedded TCP when they are used together.

Additionally, the works in [27] and [28] have used TCP/IP in LLNs for web services. Altmann *et al.* investigated the use of embedded web services in smart metering applications (including single node TCP experiment for web) [29]. These works, among a few, are examples of TCP being used in LLN. However, they used IPv4 or tested only a single node, and did not evaluate TCP over IPv6/RPL/6LoWPAN in a multihop LLN. Recently, Dunkels *et al.* evaluated the performance of low-power IPv6 for IoT using the Contiki OS [19]. They discussed the performance of RPL, ContikiMAC, and RESTful data acquisition with HTTP over TCP. However, TCP evaluation was performed only through simulations with seven nodes in a linear topology and single stream of data. In contrast, we evaluate TCP over RPL performance in a 30 node multihop LLN testbed with 30 concurrent streams of data.

III. METHODOLOGY

Consider an IoT LLN system as depicted in Fig. 1 where thousands of LLN endpoints form a mesh network rooted at an LBR. The LBR connects the LLN to a server in wide area network (WAN) which can be the Internet or private IP-based intranet [30]. LLN endpoints communicate with each other through IEEE 802.15.4 links and use RPL to construct routes towards the LBR. At the network and transport layers, endpoints use TCP/IPv6 to communicate with the server. Traffic flows in both directions, upstream from LLN endpoints to the server and downstream from the server to LLN endpoints.

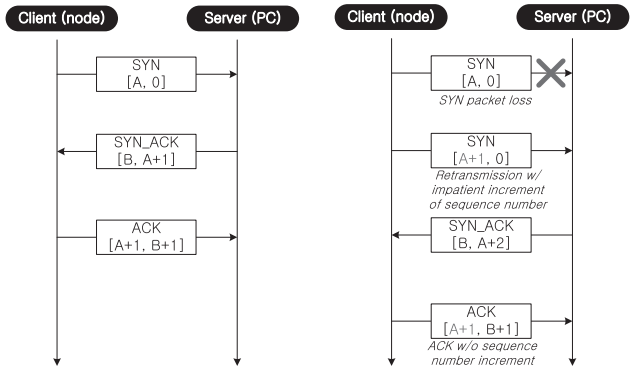
A. Experimental Setup

We configured a testbed environment as depicted in Fig. 2 where 30 LLN endpoints, one LBR (marked with the star), and one server are deployed in an office environment [31]. The server is a Linux desktop PC which uses a common Linux TCP/IP stack. The LBR uses the *ppprouter* stack in TinyOS, and exchanges IPv6 packets with the server through UART and with LLN nodes through IEEE 802.15.4 links. Each LLN node is a TelosB [32] clone device with an MSP430 microcontroller and a CC2420 radio, and uses an embedded TCP/IP (TinyOS BLIP) stack. Each node uses a transmission power of -22 dBm with antenna gain of 5dB which forms a 5-hop testbed network with RPL. Each node employs CSMA and a FIFO transmit queue size of 10 packets. An important point to note is that this study measures data delivery performance between embedded TCP endhosts and a common Linux TCP endhost which can be located anywhere on the Internet.

Using the above hardware and software setup, we make all 30 nodes generate 10–60 data packets per minute (ppm) *per node* concurrently. That is traffic load of 5–30 data packets per second for the LBR[2]. Although typical LLN applications generate low rate data, workloads produced by large scale applications such as smart grid can lead to congested resources. For example, in a network consisting of 5,000 nodes as in Cisco's CG-Mesh deployment, each node generates a packet every 10–15 min, which corresponds to the similar traffic load at the bottleneck LBR in our test environments. We focus on the performance of routing and transport layers, and thus conduct the experiments during night time to avoid interference from daily activities of office occupants (i.e., static environments).

Finally, each LLN node is connected to a PC via USB and sends log messages to the PC through UART back-channel. We gather the log messages from each PC through ethernet back-channel, obtaining various performance measurements and real-time operation status. The UART and ethernet back-channels are only for debugging and statistics gathering, and are not used for data communication between nodes.

[2]This is close to the limit considering the multihop effect where a relay node cannot send and receive at the same time, and the nodes two-hops apart cannot successfully deliver packets at the same time due to collision.



(a) (b)

Fig. 3. Handshake process for connection setup in BLIP TCP: (a) Normal operation and (b) abnormal operation when *SYN* packet is lost.

B. Identifying and Rectifying problems in BLIP and TinyRPL

TinyOS, the embedded software in our experiments, is a popular open source OS for LLN endpoints [33]. BLIP and TinyRPL are respectively an IPv6 stack (including TCP and 6LoWPAN) and an implementation of RPL in TinyOS [21]. Preliminary experiments with BLIP and TinyRPL resulted in performance that was significantly worse than anticipated based on prior work (see Section II). After further investigation of the IPv6 stack in TinyOS 2.1.2 (latest version), we found that at least three implementation details were problematic.

The first two problems are in the TCP implementation of BLIP. TCP in BLIP has a periodic timer of 0.5 seconds which controls the (re)transmission at the TCP sender. Transmission and retransmission are both processed via the same function which is triggered by the transmission timer expiration and increments the retransmission counter in the TCP. This results in BLIP TCP increasing the retransmission counter not only for retransmissions but also for first transmissions.

However, BLIP TCP does not reinitialize the retransmission counter even when an ACK is received, which prompts the sender to respond as if there were repeated transmission failures before the current transmission attempt. The sender closes the end-to-end connection when the retransmission counter reaches 6, and thus the BLIP TCP *always closes* the connection every six transmissions (including retransmissions) after opening the connection. We fixed the problem by re-initializing the retransmission counter whenever an ACK is received for a transmitted data packet.

The second problem is in the connection establishment process of TCP in BLIP. It uses the same handshaking process as regular TCP for setting up a connection between two hosts as depicted in Fig. 3(a). A client initiates the connection by transmitting a *SYN* packet to the server with a predefined sequence number A and *ACK* number 0. After receiving the *SYNACK* from the server, which has a random sequence number B and *ACK* number $A + 1$, the client transmits an *ACK* for the *SYNACK* with sequence number $A + 1$ and *ACK* number $B + 1$, and finishes the handshake process.

BLIP TCP, however, increases the sequence number by one (i.e., A to $A + 1$) at the client when transmitting the *SYN* packet (before receiving a *SYNACK*). The hasty increment of the sequence number causes a problem when *SYN* packet delivery fails as illustrated in Fig. 3(b). The client retransmits the *SYN* packet with the incremented sequence number $A + 1$ and receives the *SYNACK* with *ACK* number $A + 2$. Since the client has already finished increasing the sequence number, it sends the *ACK* to the server without updating it into $A + 2$, which results in connection failure due to server misbehavior. We empirically observed that *SYN* loss is not negligible in LLNs, especially when a client node has a large hop distance from the LBR. To resolve this issue of sequence number mismatch, we decrease the sequence number to its previous value when a client node retransmits the *SYN* packet.

The third problem is in TinyRPL. In RPL, each node sends a destination advertisement object (DAO) message towards the root periodically[3] and also when its upstream route has changed. TinyRPL implements the "storing mode" of RPL, and thus each node sets up a downstream route to the DAO sender and adds it to the routing table whenever receiving a DAO message. Each entry in the downstream routing table is removed when no DAO is received from the destination of the entry for a certain timeout period (20 minutes by default). To this end, TinyRPL uses a timeout counter called RemoveTimer for each downstream route entry.

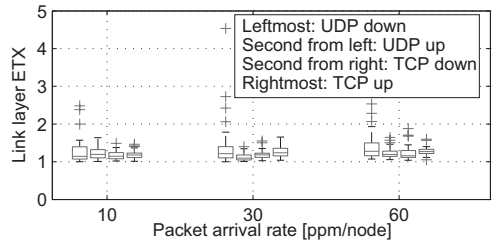
However, it does not reinitialize RemoveTimer of each entry even when a corresponding DAO message is received. Thus, a new downstream route entry has a fixed lifetime, and a node suffers from the absence of a downstream route between the timeout removal and the reception of next DAO message. We were alerted to this problem in experiments by observing very poor downstream delivery performance. We fixed the problem by reinitializing RemoveTimer at every reception of an updated DAO message. The experiments and measurement results reported below were obtained after correcting the three problems in order to focus our work on the high-level characteristics of embedded TCP and RPL rather than their current implementation.

As a final note, we would like to emphasize that these problems have not been reported before in the literature (even though these implementations are 2 years or older and there have been numerous related works using these implementations). Prior works have not found these problems since they have tested TCP and RPL in either small networks (e.g. single hop or single line topology) where packet losses and timeouts were infrequent, or in simulation studies only. Thus, it is one of our contributions to identify and report the problems in a widely used open source OS in LLN.

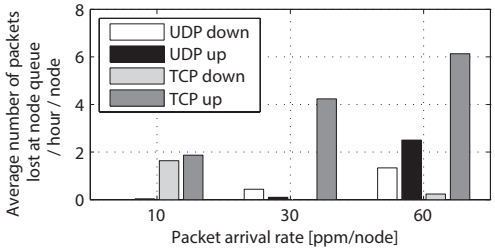
IV. UNDERSTANDING THE DIFFERENCES OF EMBEDDED LIGHTWEIGHT TCP

BLIP TCP is a lightweight version of the common Linux TCP that respects the hardware limitations of embedded sensor nodes under the assumption of low traffic load. Although it is equipped with the fundamental functions of the standard TCP for interconnection and interoperability, BLIP TCP has three major differences compared to a common implementation, which are discussed below.

[3]Depending on the implementation, it can be pseudo-periodic. The RPL standard RFC6550 does not mandate the transmission timing of DAO messages.

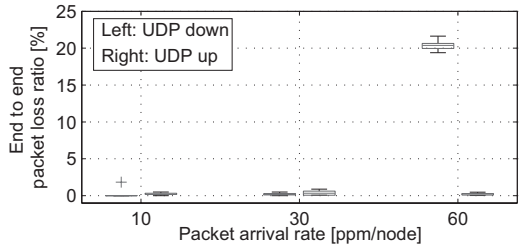


(a)

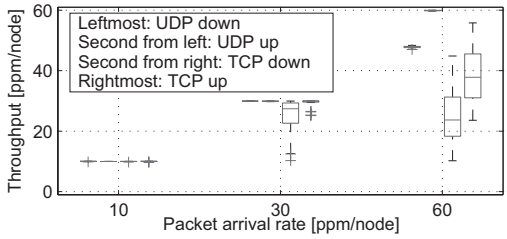


(b)

Fig. 4. IPv6 and link layer performance of each node: (a) Link layer ETX and (b) packets lost at queue.



(a)



(b)

Fig. 5. Application layer performance of each node: (a) End-to-end loss rate using UDP and (b) achieved throughput.

1) Limited congestion control: One of the main functions of TCP is congestion control which has evolved over three decades and produced several flavors (e.g. CUBIC [34] and CTCP [35]). BLIP TCP has basic congestion control mechanisms such as slow-start, congestion avoidance, and fast retransmission. However, it cannot use them except for fast retransmission since it transmits all data in the buffer once the transmission timer fires. This transmission strategy has been designed based on the assumption that LLN is used in low rate traffic environments, which is no longer valid in large scale applications such as smart grids due to large number of nodes.

2) No receive buffering: Linux TCP has sufficient receive buffer to guarantee in-order packet delivery. However, BLIP TCP does not have it due to memory limitation in low cost platforms, and sets the receive window $rwnd$ to one maximum segment size (MSS). The limited $rwnd$ incurs low downstream throughput in two ways. Since the number of TCP packets that can be transmitted without being acknowledged is $min(cwnd, rwnd)$ where $cwnd$ denotes the congestion window, a Linux TCP host can transmit at most one TCP packet at a time to a BLIP TCP host. Furthermore, Linux TCP host continuously fragments a payload to minimum size (8 bytes) since it detects that the receive buffer of embedded TCP host is full, incurring more transmission overhead.

3) Fixed RTO: Another main feature of TCP is reliable data transmission based on end-to-end retransmissions. Linux TCP provides two different retransmission schemes: retransmission after RTO and fast retransmission. A TCP sender triggers fast retransmission when it receives 3 duplicate ACKs, and RTO retransmission when it does not receive an ACK until the RTO timer expires. Linux TCP adapts the RTO by incorporating network state (i.e., $\mathrm{RTO} = \mathrm{RTT} + 4{\cdot}\mathrm{RTTDEV}$) and doubles it when timeout actually happens (i.e., binary exponential backoff) such that retransmissions happen less aggressively in congested environments. Although BLIP TCP provides fast retransmission similarly to Linux TCP, it uses a constant RTO value (3 seconds by default), resulting in constant rate retransmissions. This may aggravate congestion under heavy load while under-utilizing a network under light load. In Section V, we will show that throughput performance of the TCP upstream significantly varies with the RTO value.

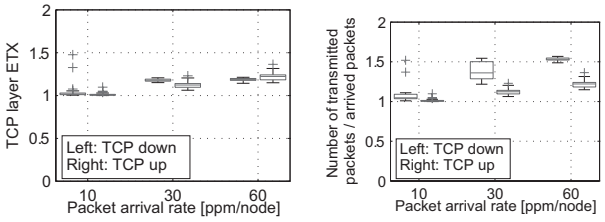
Other implementation – *uIP*: Although we have not experimented with the counterpart TCP/IPv6 implementation in the Contiki OS, called *uIP*, we believe its implementation follows a similar approach, if not identical, as gleaned by the description in [18]. Specifically, it is stated that *uIP* uses a single global buffer for holding packets and advertises a very small receiver window such that only a single un-acknowledged TCP segment is in flight per connection. For outgoing traffic, *uIP* does not queue data for retransmissions. Instead, the application is responsible for reproducing the data when a retransmission is necessary, checking the number of available bytes in the send window, and adjusting the number of bytes to send accordingly. If no buffer space is available, the application has to defer transmission and wait. Since only one in-flight TCP segment per connection is allowed, there is no congestion control mechanism necessary. All of these descriptions are identical to BLIP.

V. MEASUREMENT RESULTS

In this section, we first present our measurement results obtained from running *TCP over RPL* on a multihop LLN testbed under various scenarios, and then, introduce some research challenges for performance enhancement.

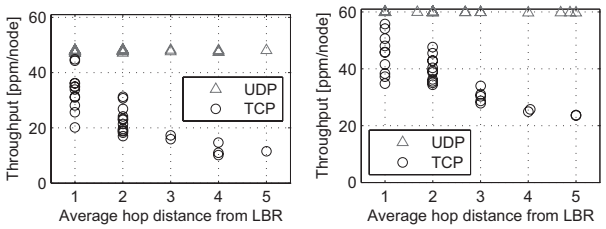
A. IPv6 and Link Layer Performance

We first present link and IPv6 layer performance when using RPL in our testbed environment. Fig. 4(a) plots link layer ETX of each node with varying packet arrival rate. We can see that link layer ETX is sufficiently low for all nodes regardless of



(a) (b)

Fig. 6. Packet arrival rate vs. Normalized TCP transmission overhead: (a) TCP layer ETX and (b) total TCP overhead.



(a) (b)

Fig. 7. Hop distance vs. achieve throughput for each node at heavy load: (a) Downstream and (b) upstream.

transport protocol and traffic load. This shows that RPL allows each node to have a parent node with good link quality in our testbed environment independent from whether TCP or UDP is used. Furthermore, UDP downstream has the worst link ETX among the four cases because RPL selects its parent based on uplink ETX but there are insufficient amount of upstream traffic to accurately measure ETX. Whereas, TCP downstream has sufficient upstream ACK traffic for that purpose.

Fig. 4(b) plots the number of packets dropped at the node queues (except LBR) in the IPv6 layer with varying arrival rate. It shows that queue losses are negligible at all cases despite high traffic load. Moreover, TCP upstream experiences higher queue loss rate than TCP downstream due to the different RTO strategy employed by the embedded TCP and the Linux TCP. Embedded TCP uses a fixed RTO which incurs relatively more aggressive retransmissions under heavy congestion. On the other hand, Linux TCP adaptively increases RTO for nodes that are farther away from the LBR or experiencing more packet losses, resulting in slowdown of the downstream traffic.

Fig. 5(a) plots the end-to-end packet loss ratio with varying packet arrival rates when using UDP over RPL[4]. First, we observe that all nodes experience negligible loss ratio at light load, thanks to the reliable parent selection of RPL. As the traffic load is increased, uplink loss ratio remains low while downstream loss ratio gradually increases. Furthermore, all nodes experience similar level of packet loss at heavy downstream load. This behavior, which does not align with the low link layer ETX and low queue loss rate results as shown in Figs. 4(a) and 4(b) respectively, stems from LBR queue overflow. It is because wired link (PC from/to LBR) delivers packets at a higher rate than what the wireless link (LBR from/to LLN node) can accommodate given the contention in the medium.

B. TCP (Re)transmission Overhead

In this subsection we investigate how embedded TCP behaves differently from Linux TCP and how that impacts the upstream and downstream performance of TCP in LLN. Fig. 6(a) plots the *ETX at the TCP layer* with varying traffic load. We define TCP ETX as the total number of end-to-end transmissions (including retransmissions) that the TCP sender host has made for each data packet. TCP layer ETX of downstream case is higher than upstream case at light load. This is because the RTO value of Linux TCP sender is very small due to small average RTT at light load. Thus, Linux TCP can trigger end-to-end retransmission even for a packet successfully delivered when RTT is relatively large for that particular packet. On the other hand, embedded TCP retransmits a packet only when it is lost since the RTT of a successful packet delivery is sufficiently smaller than the fixed RTO (i.e., 3 seconds) in our network. However, at heavy load, embedded TCP triggers end-to-end retransmissions more aggressively than Linux TCP since it does not adapt the RTO even when RTT increases significantly. This is the reason for higher queue losses in Fig. 4(b).

As aforementioned in Section IV, Linux TCP suffers from another overhead when interoperating with embedded TCP due to extremely small $rwnd\ (=1)$. Since receive buffer of embedded TCP host is always full, Linux TCP host fragments a packet into the smallest size (i.e., 8 bytes) when it detects large RTT. Thus, Linux TCP has larger number of packets to be transmitted than arrived ones, increasing the transmission overhead as shown in Fig. 6(b) which plots the total TCP transmission overhead (including retransmission and fragmentation) compared to arrived packets.

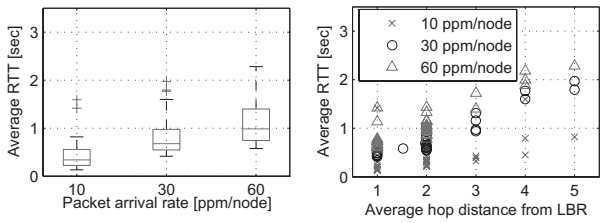
C. Achieved Throughput

Next, we compare the throughput achieved by TCP and UDP over RPL in multihop LLN with tree topology. Fig. 5(b) presents achieved throughput of TCP and UDP for each of 30 nodes with varying packet arrival rate. We first observe that UDP throughput is a direct function of end-to-end packet loss ratio given in Fig. 5(a). Furthermore, UDP provides fair throughput for all nodes even at heavy load. This is due to the fact that RPL allows each node to have a parent with good link quality and the LBR is the traffic bottleneck at heavy load.

On the other hand, TCP provides lower throughput than UDP as traffic load increases. This is because TCP has extra overhead caused by ACK transmissions. Moreover, packet loss (including ACK loss) or large RTT induces TCP to spend idle time waiting for ACKs. Also, TCP upstream case achieves higher throughput than TCP downstream case. This again comes from the RTO control strategy and limited receive buffer size of embedded TCP. Aggressive retransmission of embedded TCP due to fixed RTO incurs better throughput performance, whereas Linux TCP can make a node suffer significantly from starvation due to binary exponential backoff when transmissions fail consecutively. Moreover, limited receiver buffer size of embedded TCP incurs inefficient fragmentations in Linux TCP.

Furthermore, unlike UDP, throughput varies significantly among nodes when using TCP. To analyze the details, we use Fig. 7 which plots achieved throughput of each node with respect to the average hop distance when traffic load is 60

[4]TCP provides perfect PRR for all nodes due to end to end retransmission.



(a) (b)

Fig. 8. Average RTT for each node at TCP downstream case: (a) Vs. packet arrival rate and (b) vs. hop distance.

ppm/node. When using TCP, throughput reduces as hop distance increases for both upstream and downstream case. Figs. 8(a) and 8(b) plot average RTT for each node at TCP downstream case versus packet arrival rate and hop distance, respectively[5]. They show that RTT also varies among nodes according to hop distance.

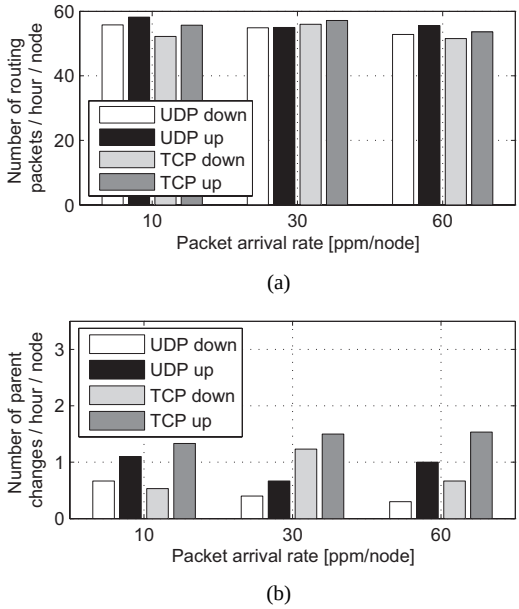
Now we analyze the reason of the throughput unfairness and the relationship between latency and throughput unfairness. In fact, the latency unfairness in a multihop network is unquestionable since physical and topological distances differ between nodes. However, why does the latency unfairness cause the throughput unfairness? It is well known that TCP can provide high throughput even when it experiences large RTT thanks to its congestion control mechanism, and thus throughput and RTT are not strongly correlated to each other. However, we reveal that using embedded TCP for LLN endpoints is what creates the correlation between RTT and throughput in multihop LLN. In upstream cases, limited congestion control of embedded TCP translates RTT unfairness to throughput unfairness even though Linux TCP has enough $rwnd$. In downstream cases, extremely limited $rwnd$ (i.e., no receiver buffer) of embedded TCP invalidates the congestion control mechanism of common TCP and does not allow a common TCP host to transmit a new packet before receiving an ACK for the previous packet. Thus, conventional embedded TCP cannot provide fairness in multihop LLNs although RPL provides a reasonable bi-directional route for each node.

D. RPL Operation under TCP

In this subsection, we investigate the mutual influence between RPL and TCP. To minimize the influence of the underlying routing layer (RPL) as well as the link layer (IEEE 802.15.4) to the TCP vs. UDP comparison, we have used the same testbed, ran experiments at similar times of day, and made the software identical except the TCP and UDP part. However, the different characteristics of TCP and UDP may in reverse influence the link layer ETX especially due to the added ACK traffic. If added traffic of TCP influences RPL to make parent changes, which will result in more RPL DIO and RPL DAO traffic, which will in turn influence the performance of TCP again. This mutual influence between TCP and RPL is one of the goals that our work aimed to investigate.

Figs. 9(a) and 9(b) plot the average RPL control packet overhead and the average RPL parent change frequency of a node with varying packet arrival rate, respectively. Our results show

[5]Downstream RTT is measured at PC server using wireshark.



(a)

(b)

Fig. 9. Performance of RPL under TCP per node: (a) Control packet overhead and (b) number of parent changes.

that there is no significant difference in the number of parent changes between TCP and UDP, resulting in no significant difference in the number of RPL DIO and RPL DAO traffic. Considering the results in Fig. 4(a), link capacity is sufficient to deliver the considered traffic load in our experiments, and thus, RPL does not have to trigger parent changes to deal with added traffic from TCP. Thus we believe that the influence from the RPL protocol to the observed inadequacies in previous sections is minimal, and the observed problems are really coming from TCP implementations and flow control operations.

E. Research Challenges

Through the findings from the experiments, we introduce some research challenges which exist in the current state-of-the-art embedded TCP/IPv6/RPL implementations.

RPL research:

Routing topology constructed by RPL has a critical effect on the transport protocol performance. However, we have found that RPL does not change the topology even when traffic load increases since link ETX remains low regardless of the traffic. In other words, RPL does not consider traffic load while constructing routes, and traffic load does not impact ETX enough to let nodes change routes in response to congestion. Moreover, low cost embedded devices in LLNs have much smaller queue size than other communication devices for WiFi or LTE due to limited memory, and thus suffer significantly from queue losses.

To verify the problem, we make another topology where node 3 in Fig. 2 is the LBR and node transmission power is -17 dBm. RPL constructs 7 hop network in the new topology. Fig. 10 plots achieved throughput of each node with respect to its average hop distance from the LBR when packet arrival rate is 60 ppm/node. Compared to the results in Fig. 7, we observe that UDP throughput is significantly degraded and TCP throughput becomes even unfairer among nodes. This is because some nodes experience severe queue losses and causes through-

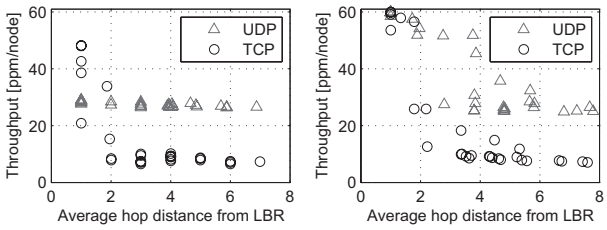


Fig. 10. Hop distance vs. achieve throughput for each node at heavy load (different topology): (a) Downstream and (b) upstream.

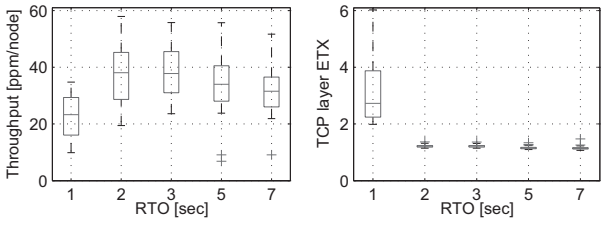


Fig. 11. RTO vs. TCP performance for each node in upstream case: (a) Achieved throughput and (b) TCP layer ETX.

put starvation for their children nodes. However, a starving RPL node cannot escape from the congested parent node since it still has good link quality with that parent. Thus, RPL has load balancing problem at heavy load, and it needs to be improved to take traffic load (i.e., upper layer behavior) into consideration while constructing routes for a large scale (or heavy traffic) applications such as smart grid [36].

TCP research:

Design of TCP, which can alleviate the throughput unfairness problem in LLN while maintaining the end-to-end standard compatibility, is an interesting research topic. To this end, in the upstream case, design of a lightweight congestion control mechanism can be considered since Linux TCP has very large $rwnd$. In the downstream case, a smart LBR can be designed to enhance performance without changing the common Linux TCP for interoperability. For example, we may add a new layer above IP layers within LBR, which buffers packets to control RTT of each node and allows the server to experience similar RTT at the TCP layer, both among nodes and across traffic load variations. This can mitigate the effect of highly variable RTT, and also the inefficient packet fragmentation due to small $rwnd$ of embedded TCP. Moreover, if an lightweight reordering scheme can be implemented at the receive buffer of the low cost embedded devices, downlink congestion control can also be considered.

As a short case study, Figs. 11(a) and 11(b) plot the achieved throughput and TCP ETX of each node, respectively, in TCP upstream case with varying constant RTO of embedded TCP. We can see that use of too small RTO increases TCP ETX due to severe queue losses and thus reduces throughput. Large RTO also causes throughput degradation due to waste of time waiting for retransmission timeout. Based on this observation, it may be possible to devise a new mechanism which allows each node to adaptively (and collaboratively) optimize their RTOs according to the operation environment to improve throughput while maintaining fairness among nodes.

VI. SUMMARY AND CONCLUSION

This paper presents a comprehensive measurement study of TCP over RPL in an IPv6 and IEEE 802.15.4-based LLN. Specifically, we measure the performance through testbed experiments where a *common Linux TCP host* exchange data with *multiple embedded TCP hosts* through an LLN of *multihop tree* topology constructed by *RPL*. Our results confirm that although it is feasible to use TCP over RPL in LLN, TCP sacrifices significant throughput to maintain its reliability. Furthermore, current design of embedded TCP incurs throughput unfairness among nodes. More importantly, although TCP does not alter the operations of RPL in terms of routing overhead or parent changes, TCP suffers from unfairness and starvation depending on the tree topology constructed by RPL. The unfairness problem is critical since the most poorly performing node defines the minimum performance achieved by the network and whether it is able to meet application requirements. We have also identified some problems in the implementation of RPL and embedded IPv6 stack in TinyOS which will help future users of the protocols when deploying their network. The findings and contributions of this work provide an understanding of the performance trade-offs of using embedded TCP over RPL in an IPv6-based LLN and also research challenges to achieve feasible interoperation between TCP and RPL in LLNs.

REFERENCES

[1] Cisco, "Smart grid-field area network." [Online]. Available: http://www.cisco.com/web/strategy/energy/field_area_network.html
[2] E. Ancillotti, R. Bruno, and M. Conti, "The role of the rpl routing protocol for smart grid communications," *IEEE Commun. Mag.*, vol. 51, pp. 75–83, Jan. 2013.
[3] D. Popa *et al.*, "Applicability statement for the routing protocol for low power and lossy networks (RPL) in AMI networks," *Internet draft*, Jan. 2015.
[4] V. Gungor *et al.*, "A survey on smart grid potential applications and communication requirements," *IEEE Trans. Ind. Inform.*, vol. 9, pp. 28–42, Feb. 2013.
[5] J. Ko *et al.*, "Connecting low-power and lossy networks to the Internet," *IEEE Commun. Mag*, vol. 49, Apr. 2011.
[6] J. Wang and V. Leung, "A survey of technical requirements and consumer application standards for IP-based smart grid AMI network," in *Proc. ICOIN*, Jan. 2011.
[7] "NIST framework and roadmap for smart grid interoperability standards, release 2.0," 2012.
[8] Z. Fan *et al.*, "Smart grid communications: Overview of research challenges, solutions, and standardization activities," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 1, pp. 21–38, 2013.
[9] T. Winter *et al.*, "RPL: IPv6 routing protocol for low-power and lossy networks," *RFC 6550*, Mar. 2012.
[10] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 packets over IEEE 802.15.4 networks," *RFC 4944*, Sept. 2007.
[11] J. Hui and P. Thubert, "Compression format for IPv6 datagrams over IEEE 802.15.4-based networks," *RFC 6282*, Sept. 2011.
[12] V. Gungor, B. Lu, and G. Hancke, "Opportunities and challenges of wireless sensor networks in smart grid," *IEEE Trans. Ind. Electron.*, vol. 57, Oct. 2010.
[13] J. Ko, S. Dawson-Haggerty, O. Gnawali, D. Culler, and A. Terzis, "Evaluating the performance of RPL and 6LoWPAN in TinyOS," in *Proc. IP+SN Workshop*, Apr. 2011.
[14] T. Clausen, U. Herberg, and M. Philipp, "A critical evaluation of the IPv6 routing protocol for low power and lossy networks (RPL)," in *Proc. IEEE WiMob*, Oct. 2011.
[15] U. Herberg and T. Clausen, "A comparative performance study of the routing protocols load and rpl with bi-directional traffic in low-power and lossy networks," in *Proc. ACM PE-WASUN*, 2011.
[16] J. Ko *et al.*, "DualMOP-RPL: Supporting multiple modes of downward routing in a single RPL network," *ACM Trans. Sen. Netw.*, vol. 11, pp. 39:1–39:20, Mar. 2015.

[17] J. W. Hui and D. E. Culler, "IP is dead, long live IP for wireless sensor networks," in *Proc. ACM SenSys*, 2008.
[18] A. Dunkels, "Full TCP/IP for 8-bit architectures," in *Proc. ACM MobiSys*, 2003.
[19] A. Dunkels *et al.*, "Low-power IPv6 for the Internet of Things," in *Proc.. INSS*, June 2012, pp. 1–6.
[20] T. Sauter and M. Lobashov, "End-to-end communication architecture for smart grids," *IEEE Trans. Ind. Electron.*, vol. 58, pp. 1218–1228, Apr. 2011.
[21] J. Ko *et al.*, "Beyond interoperability: Pushing the performance of sensor network IP stacks," in *Proc. ACM SenSys*, 2011.
[22] H. Kermajani and C. Gomez, "On the network convergence process in rpl over IEEE 802.15.4 multihop networks: Improvement and trade-offs," *Sensors*, vol. 14, no. 7, pp. 11993–12022, 2014.
[23] E. Ancillotti, R. Bruno, and M. Conti, "Reliable data delivery with the ietf routing protocol for low-power and lossy networks," *IEEE Trans. Ind. Inform.*, vol. 10, pp. 1864–1877, Aug. 2014.
[24] D. Wang, Z. Tao, J. Zhang, and A. Abouzeid, "RPL based routing for advanced metering infrastructure in smart grid," in *Proc. IEEE ICC Workshops*, May 2010.
[25] N. Bressan *et al.*, "The deployment of a smart monitoring system using wireless sensor and actuator networks," in *Proc. IEEE SmartGridComm*, Oct. 2010.
[26] S. Duquennoy, F. Österlind, and A. Dunkels, "Lossy links, low power, high throughput," in *Proc. ACM SenSys*, 2011.
[27] V. Altmann, J. Skodzik, F. Golatowski, and D. Timmermann, "Investigation of the use of embedded web services in smart metering applications," in *Proc. IEEE IECON*, Oct. 2012.
[28] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny web services: Design and implementation of interoperable and evolvable sensor networks," in *Proc. ACM SenSys*, 2008.
[29] G. Moritz, F. Golatowski, C. Lerche, and D. Timmermann, "Beyond 6lowpan: Web services in wireless sensor networks," *IEEE Trans. Ind. Inform.*, vol. 9, pp. 1795–1805, Nov. 2013.
[30] Y. Zhu *et al.*, "On deploying relays for connectd indoor sensor networks," *J. Commun. Netw.*, vol. 16, pp. 335–343, June 2014.
[31] H.-S. Kim *et al.*, "MarketNet: An asymmetric transmission power-based wireless system for managing e-price tags in markets," in *Proc. ACM SenSys*, Nov. 2015.
[32] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *Proc. IPSN/SPOTS*, Apr. 2005.
[33] J. Ko *et al.*, "ContikiRPL and TinyRPL: Happy together," in *Proc. IP+SN Workshop*, Apr. 2011.
[34] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcpfriendly highspeed tcp variant," *ACM SIGOPS OSR*, vol. 42, pp. 64–74, July 2008.
[35] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound tcp approach for high-speed and long distance networks," in *Proc. IEEE INFOCOM*, 2006.
[36] H.-S. Kim, J. Paek, and S. Bahk, "QU-RPL: Queue utilization based RPL for load balancing in large scale industrial applications," in *Proc. IEEE SECON*, June 2015.

Hyung-Sin Kim received the B.S. degree from Seoul National University (SNU), Seoul, Korea in 2009 and the M.S. degree from SNU in 2011, all in Electrical Engineering. He is currently working towards the Ph.D. degree in Electrical and Computer Engineering at SNU. His research interests are in the area of low power wireless embedded systems and Internet of Things including real-world sensornet applications, network and transport protocols, asymmetric transmission power-based network architecture, and also in the area of wireless cellular networks including channel estimation, beamforming, and resource scheduling.

Heesu Im received B.S., M.S., and Ph.D. degrees in the School of Electrical Engineering and Computer Science, SNU in 2008, 2010, and 2015, respectively. He is currently working in Samsung Electronics as a Senior Engineer. His research interests include network protocol design, wireless sensor networks, and P2P networks.

Myung-Sup Lee received the B.S. degree from SNU, Seoul, Korea in 2014 in Electrical and Computer Engineering. He is currently working towards the M.S. degree in Electrical and Computer Engineering at SNU. His research interests include visual sensor network, asymmetric transmission power-based network architecture, and TCP and RPL protocols for low power and lossy network.

Jeongyeup Paek is currently an Assistant Professor at the School of Computer Science and Engineering, Chung-Ang University, Seoul, Korea. He received his B.S. degree from SNU in 2003 and his M.S. degree from University of Southern California in 2005, both in Electrical Engineering. He then received his Ph.D. degree in Computer Science from the University of Southern California in 2010. Jeongyeup Paek conducted research as a Postdoctoral Research Associate at USC, and also as a Research Intern at Deutsche Telekom Inc. R&D Labs USA, both in 2010. He then joined Cisco Systems Inc. in 2011, where he was a Technical Leader in the Internet of Things Group, Connected Energy Networks business unit. Until recently, he was an Assistant Professor at the Department of Computer Information Communication Engineering, Hongik University, Korea.

Jeongyeup Paek's prior research has focused on topics in wireless sensor network systems including network and transport protocols, architecture for tiered embedded networks, real-world sensornet applications, and also in the areas of mobile smartphones systems. His recent research interests are in low-power lossy networks for Internet of Things with focus on protocols, architecture, software, and performance enhancements.

Saewoong Bahk received B.S. and M.S. degrees in Electrical Engineering from Seoul National University in 1984 and 1986, respectively, and the Ph.D. degree from the University of Pennsylvania in 1991. From 1991 through 1994, he was with AT&T Bell Laboratories as a Member of Technical Staff where he worked for AT&T network management. In 1994, he joined the School of Electrical Engineering at SNU and currently serves as a Professor. He has been serving as TPC members for various conferences including ICC, GLOBECOM, INFOCOM, PIMRC, WCNC, etc. He is on the Editorial Boards of IEEE Transactions on Wireless Communications (TWireless), Computer Networks Journal (COMNET), and Journal of Communications and Networks (JCN). His areas of interests include performance analysis of communication networks and network security. He is an IEEE Senior Member and a Member of Whos Who Professional in Science and Engineering.