Load Balancing under Heavy Traffic in RPL Routing Protocol for Low Power and Lossy Networks

Hyung-Sin Kim, Hongchan Kim, Jeongyeup Paek, and Saewoong Bahk

Abstract—RPL is an IPv6 routing protocol for low-power and lossy networks (LLNs) designed to meet the requirements of a wide range of LLN applications including smart grid AMIs, industrial and environmental monitoring, and wireless sensor networks. RPL allows bi-directional end-to-end IPv6 communication on resource constrained LLN devices, leading to the concept of the Internet of Things (IoT) with thousands and millions of devices interconnected through multihop mesh networks. In this article, we investigate the load balancing and congestion problem of RPL. Specifically, we show that most of packet losses under heavy traffic are due to congestion, and a serious load balancing problem appears in RPL in terms of routing parent selection. To overcome this problem, this article proposes a simple yet effective queue utilization based RPL (*QU-RPL*) that achieves load balancing and significantly improves the end-to-end packet delivery performance compared to the standard RPL. *QU-RPL* is designed for each node to select its parent node considering the queue utilization of its neighbor nodes as well as their hop distances to an LLN border router (LBR). Owing to its load balancing capability, *QU-RPL* is very effective in lowering queue losses and increasing the packet delivery ratio. We implement *QU-RPL* on a low-power embedded platform, and verify all our findings through experimental measurements on a real testbed of a multihop LLN over IEEE 802.15.4. We present the impact of each design element of *QU-RPL* on performance in detail, and also show that *QU-RPL* reduces the queue loss by up to 84% and improves the packet delivery ratio by up to 147% compared to the standard RPL.

Index Terms—Low-power Lossy Network (LLN), RPL, IPv6, 6LoWPAN, IEEE 802.15.4, Load Balancing, Congestion Control, Routing, Wireless Sensor Network

1 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) [2], [3], industrial monitoring [4], [5], and wireless sensor networks [6]–[8]. Recently, most LLN deployments employ the open and standardized IP/IPv6-based architecture to connect with the larger Internet. This approach makes LLNs more interoperable, flexible, and versatile, leading to the emerging concept of the *Internet of Things (IoT)*. With the support of various standardization efforts from the IEEE, IETF, and Zigbee, IoT systems are now equipped with protocols and application profiles that are ready for large-scale deployments [9]–[13]. Among these, this article focuses on the load-balancing problem of the recently standardized IPv6 routing protocol for LLN, termed RPL [9].

RPL is designed for resource constrained embedded devices to support upcoming smart grids and many other LLN applications [9]. It is a distance vector type routing protocol that builds directed acyclic graphs (DAGs) based on routing metrics and constraints. In most deployment scenarios, RPL constructs tree-like routing topology called destination-oriented directed acyclic graph (DODAG) rooted at an LLN border router (LBR), and supports bi-directional IPv6 communication between network devices. Each node in RPL advertises routing metrics and constraints through DODAG information object (DIO) messages, and builds a DAG according to its objective function (OF, rules governing how to build a DAG) and the information in DIO messages. Upon receiving DIO messages from its neighbors, a node chooses a routing parent according to its OF and local policy, and then constructs a routing topology (i.e., DODAG).

Cisco's field area network (FAN) solution for smart grids [2] (CG-Mesh) is a good commercial example that uses RPL in LLNs. It is based on the IPv6 architecture, and uses IEEE 802.15.4g/e at the PHY and MAC layer to form LLNs. On top of that, it uses 6LoWPAN [11], RPL, and IPv6 to provide end-to-end two-way communication to each smart metering endpoint. It supports up to 5,000 nodes per LBR (DODAG root), and envisions millions of nodes within a FAN. Cisco's CG-Mesh system provides an initial evidence that use of IPv6 and RPL over IEEE 802.15.4 is feasible towards large scale LLNs. It is also part of growing industry efforts to invest in LLN solutions to facilitate IoT. Furthermore, it shows that although RPL has been mainly designed and used for low rate traffic scenarios, it needs to be capable of handling high rate traffic. This is because, even though each node generates low rate data, nodes near a sink (i.e., LBR) have to relay very high rate traffic. For this reason, congestion and load-balancing issues need to be investigated for RPL under high traffic scenarios [14].

In this article, we tackle the load balancing and congestion problem of RPL. To do so, we first provide an experimental measurement study of RPL in high traffic scenario on a real

- *Saewoong Bahk is the corresponding author.*
- *Hyung-Sin Kim, Hongchan Kim and Saewoong Bahk is with the Department of Electrical and Computer Engineering, Seoul National University, Republic of Korea (email: hskim@netlab.snu.ac.kr, hckim@netlab.snu.ac.kr, sbahk@snu.ac.kr). Jeongyeup Paek is with the School of Computer Science and Engineering, Chung-Ang University, Republic of Korea (email: jpaek@cau.ac.kr).*
- *An earlier version of this article appeared in the proceedings of the 12th IEEE International Conference on Sensing, Communication, and Networking (SECON'15), June 2015 [1].*
- *This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2015R1A2A2A01008240), in part by the ICT R&D program of MSIP/IITP, Republic of Korea. [B0717-16-0026, Research on Disaster Communication for Reliable Network Configuration in a Shadow Area of a Disaster Zone], and in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2014R1A1A2056626).*

multihop LLN with low-power embedded devices. As a result, we identify that most of packet losses under heavy traffic are due to congestion, and that there exists a serious load balancing problem in RPL in terms of routing parent selection. To solve this problem, we propose a simple yet effective enhancement to RPL, termed "*Queue Utilization based RPL*" (*QU-RPL*), that significantly improves the end-to-end packet delivery performance by balancing the traffic load within a routing tree. We show performance enhancements of *QU-RPL* through experimental measurements on a real multihop LLN testbed running RPL over IEEE 802.15.4.

We evaluate our proposal against a prototype implementation of RPL in TinyOS, called TinyRPL, which uses the default objective function *OF0* [15] and hop count based routing metric. We call this *default RPL* or simply *RPL* to distinguish it from our proposed *QU-RPL*. Our implementation of *QU-RPL* is a modification on this implementation.

It is worth noting that the RPL standard [9] decouples the definition of OFs and routing metrics from the main standard to provide a high degree of flexibility. It allows implementations to freely choose OFs, local policies, routing metrics, and constraints to be used for parent selection. Thus, our proposal can be regarded as *optional* from the viewpoint of an RPL implementation, and it is still standard compliant. Similarly, the TinyRPL implementation which combines the hop count for rank calculation (OF) and the ETX for parent selection is also standard compliant.

Although there have been a lot of performance evaluation studies on RPL in LLNs [3], [16]–[22], there is no experimental study of load balancing in RPL over a real multihop multinode LLN testbed. Our results and findings will provide an understanding of the load balancing problem in RPL and suggest its enhancements to build up large scale LLNs.

The remainder of this article is structured as follows. Section 2 presents the background and related work to clarify the motivation of our work, and Section 3 describes the considered scenario and experimental environments. Then, Section 4 discusses the load balancing problem of RPL and its implementation based on TinyRPL. Next, Section 5 proposes *QU-RPL* as a simple but efficient way to alleviate the load balancing problem, and Section 6 compares the performance of *QU-RPL* and RPL using testbed experiments. Section 7 experimentally shows the effect of each design element in *QU-RPL* on its performance. Finally Section 8 concludes the article.

2 Background and Related Work

Path selection and topology construction in RPL are governed by OFs and routing metrics used by RPL. The OF defines how to use link metrics and constraints for the rank computation, and how to select and optimize routes in a DODAG. However, the RPL standard [9] does not mandate any particular OF nor routing metric to be used, and leaves this open to implementations. Thus, it offers a great flexibility in meeting different optimization criteria required by a wide range of deployments, applications, and network design. IETF has defined some recommendations on how to implement OFs [15], [23], [24], but without specifying routing metrics to be used, and still leaves the exact selection of a parent set as an implementation choice. RFC 6551 [25] proposed some routing metrics and constraints to be used for path calculation in RPL, but also leaves the specific selection to implementations.

For this reason, most prototype implementations of RPL use OF0 [15] as their default OF[1] while combining ideas from the 'minimum rank with hysteresis objective function' (MRHOF) [23]. They also use simple routing metrics such as hop count or ETX or their combination for path calculation and parent selection. For example, TinyRPL implementation in TinyOS uses OF0 with hop count for path calculation, and also uses ETX with hysteresis at the link level for parent selection. Furthermore, Cisco's CG-Mesh system [2] is known to use the ETX objective function (ETXOF) [24] which can be regarded as MRHOF combined with ETX metric, and the RPL implementation in Contiki uses MRHOF. All of these implementations are standard compliant thanks to its great flexibility.

Thus design and selection of OFs and routing metrics that meet requirements of applications and network topology are still an open research issue. Recently, Goddour *et al*. proposed QoS aware fuzzy logic OF (OF-FL) that combines a set of metrics to provide a configurable routing decision based on fuzzy parameters with an aim of supporting various application requirements [26]. In [27], the authors proposed a combination of two routing metrics among hop count, ETX, remaining energy, and RSSI. They also proposed two ways of combining these metrics, simple combination and lexical combination, and compared their performance and tradeoffs.

The work in [28] analyzed the impact of OFs on network topology using OF0 and link quality OF (LLQ OF). The authors in [29] proposed a delay efficient RPL routing metric. Macro *et al*. proposed two MAC-based routing metrics that consider not only ETX but also packet losses due to MAC contention [30]. They also considered traffic load in the viewpoint of power consumption and reliability required at application level. However, none of these works investigate the load-balancing problem. The RFC6552 for OF0, which is most widely used, explicitly states that there is no attempt to perform any load balancing. Our work is to fill this gap and to provide a mechanism that achieves load balancing while conforming to the RPL standard.

Aside from investigating OFs and routing metrics, several prior pieces of work have investigated the performance of RPL under various scenarios and configurations. Ko *et al*. experimentally evaluated the performance of RPL using TinyRPL implementation in TinyOS [17], and have shown that its performance is similar to that of collection tree protocol, the *de facto* data collection protocol in TinyOS, while having the benefit of an IPv6-based architecture. In [31], the authors also evaluated and compared the performances of ContikiRPL and TinyRPL using the two most widely used operating systems in wireless sensor networks, i.e., Contiki and TinyOS. They showed that, although their performance can be made comparable, parameter selection and implementation details have significant effects on the performance and interoperability of the network consisting of both implementations.

Furthermore, in [19], Herberg *et al*. compared RPL with LOAD using NS-2 simulations and found that LOAD incurs less overhead if the traffic pattern is bi-directional. The work in [21] presented simulation results on the network convergence process of RPL over an IEEE 802.15.4 multihop network, and investigated improvements and trade-offs. In [20], Accettura *et al*. analyzed performance of RPL using COOJA simulations, and Clausen *et*

1. OF0 is designed as a default OF of RPL that will allow interoperation between implementations in a wide spectrum of use cases. However, it still does not define which link properties to be used as routing metrics.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2016.2585107, IEEE Transactions on Mobile Computing

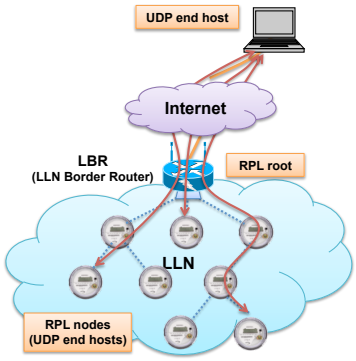


Fig. 1. Illustration of an IoT multihop LLN scenario.

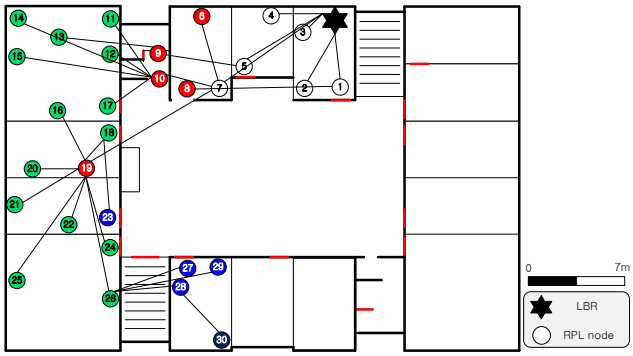


Fig. 2. Testbed topology map with a snapshot of routing path given by RPL.

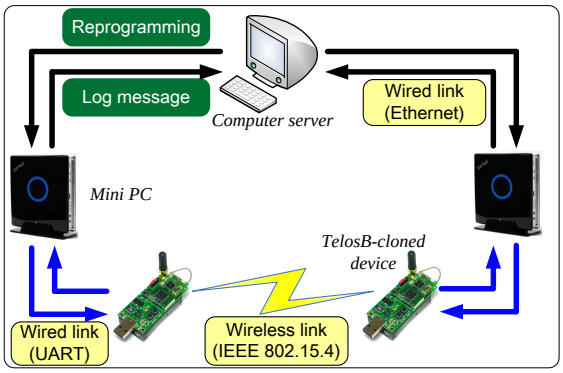


Fig. 3. Testbed architecture to monitor and reprogram each LLN node.

al. provided a critical evaluation of RPL regarding limitations, trade-offs, and suggestions for improvements [18]. The work in [8] addressed the reliability problem of downward routing in RPL and designed an asymmetric transmission power-based network where the root directly transmits downlink packets to destination nodes using much higher transmission power than low power nodes. Ko *et al*. investigated the interoperability problem of two modes of operations (MOPs) defined in the RPL standard, and show that there exists a serious connectivity problem when two MOPs are mixed within a single network [22]. To address this issue, the authors proposed DualMOP-RPL that supports nodes with different MOPs to communicate gracefully in a single network while preserving the high bi-directional data delivery performance. However, none of these works have investigated nor tackled the load balancing problem of RPL over a real multihop LLN testbed.

Ha *et al*. investigated the load balancing problem when using multiple gateways [32]. They proposed MLEq and compared its performance to that of RPL. However, they reduce traffic congestion only by using additional gateways and does not address the load balancing problem in an LLN with a single gateway. Liu *et al*. tackled the load balancing problem in a single gateway network and proposed LB-RPL which improves load balancing performance of RPL [33]. It is similar to our work in that LB-RPL allows a node to prioritize its parent candidates considering their queue utilization. However, a node detects the queue utilization information of its neighbors from how long a neighbor delays its DIO transmission; if a node is congested, it *delays* the dissemination of routing information. This is problematic because DIO packet losses and use of *TrickleTimer* do not allow DIO reception time to exactly reflect the queue utilization. Furthermore, it causes slow recovery due to long DIO transmission interval and *herding effect* by always removing the congested parent from the parent candidate set. More importantly, both of these work evaluated their proposed schemes using NS-2 simulations, and neither conduct experiments in a real LLN nor implement their schemes on real embedded devices.

3 System Model

Let's consider an IoT LLN system as depicted in Fig. 1. There are thousands of LLN endpoints that form a low-power lossy mesh network rooted at an LBR. The LBR connects the LLN to a wide area network (WAN) which can be either the public Internet or a private IP-based network [34]. Multiple servers for various purposes such as applications (e.g. CoAP), network management, DHCP, security, etc. reside behind the WAN. In this scenario, LLN endpoints utilize IEEE 802.15.4 links to communicate with each other, and use RPL to construct routes towards the LBR. On top of that, endpoints use IPv6 to communicate with servers. In this work, we focus on uplink traffic from LLN endpoints to servers.

To study the data delivery performance of RPL in multihop LLNs, we have configured a testbed environment as depicted in Fig. 2. There are 30 LLN endpoints and one LBR (marked with the star) in an office environment. The LBR is composed of a Linux desktop PC and an LLN interface which uses *ppprouter* stack in TinyOS and forwards IPv6 packets to the PC through UART at a baud rate of 115,200. Each LLN node is a TelosB clone device [35] with an MSP430 microcontroller and a CC2420 radio, and uses a transmission power of -13dBm with an antenna gain of 5dB which forms a 5∼6-hop network in our testbed.

TinyOS was used as an embedded software in our experiments. The IPv6 stack and the RPL implementation in TinyOS are called *BLIP* and *TinyRPL*, respectively. We have not used a duty cycling mechanism such as low power listening [36], [37] to focus on high rate traffic, and each node employs the default CSMA of TinyOS and a FIFO transmit queue size of 10 packets.

Using the above hardware and software setup for the experiments, we consider relatively high data rates where each node sends a data packet every 2 seconds, i.e., 30 packets per minute (ppm), or faster. Even though each node generates low rate data in typical LLN applications, traffic environments given by large scale applications such as smart grids can lead to a congested scenario. For example, in a network consisting of 5,000 nodes as in Cisco's CG-Mesh deployment, one packet generation every ~ 5.5 minutes at each node corresponds to the same traffic load at the bottlenecked LBR in our test scenario.

Finally, as depicted in Fig. 3, each LLN node is connected

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2016.2585107, IEEE Transactions on Mobile Computing

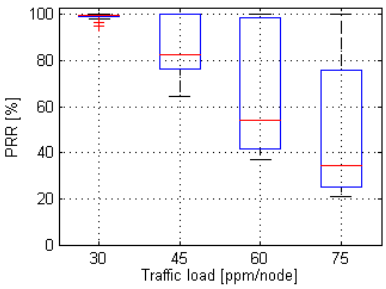


Fig. 4. End-to-end packet reception ratio (PRR) vs. uplink traffic load.

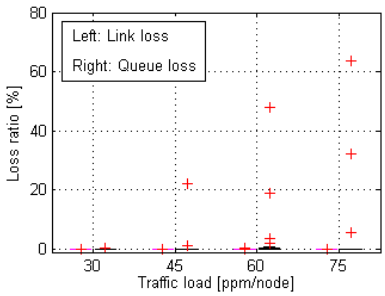


Fig. 5. IP layer packet loss ratio of each node vs. uplink traffic load.

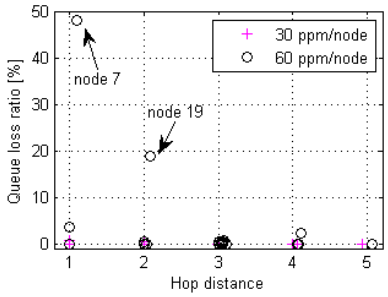


Fig. 6. Queue loss ratio of each node vs. each node's hop distance from the LBR.

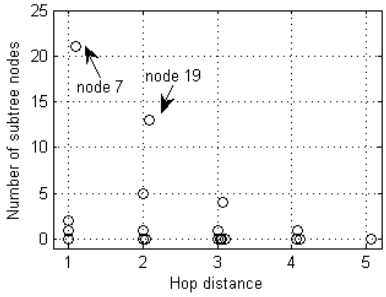


Fig. 7. RPL subtree size of each node vs. hop distance at traffic load of 60 ppm/node.

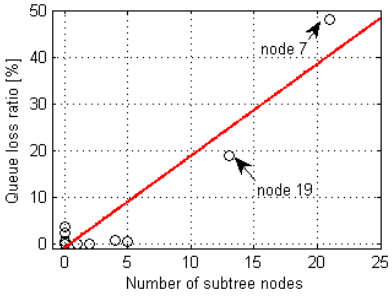


Fig. 8. Queue loss ratio of each node vs. RPL subtree size at traffic load of 60 ppm/node.

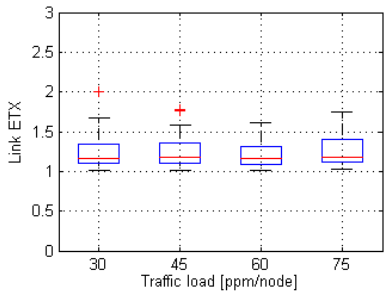


Fig. 9. Link layer ETX vs. uplink traffic load from all 30 nodes.

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. Furthermore, we remotely reprogram each node through the UART and ethernet back-channels. The two back-channels are only for debugging and statistics gathering, and are not used for data communication between nodes.

4 Problem : RPL with OF0

In this section, we first provide an experimental measurement study of RPL with OF0 in high traffic scenarios on a real multihop LLN testbed. We show that most of packet losses in high traffic scenarios are due to congestion, and that there exists a serious load balancing problem in RPL in terms of routing parent selection. We then describe the TinyRPL implementation as a basis for describing our *QU-RPL* in Section 5.

4.1 Load balancing problem of RPL

Fig. 4 plots the end-to-end packet reception ratio (PRR) of RPL (default TinyRPL) with varying traffic load for uplink UDP from all 30 nodes. The packet interval of each node varies from 0.8 to 2 seconds (i.e., 30 to 75 ppm). Considering a network size of 5,000 nodes, this corresponds to the packet interval of 2.2 to 5.5 minutes per node (i.e., 0.18 to 0.45 ppm per node). From the figure, first of all, we observe that our testbed provides near perfect PRR for all nodes when the traffic load is light, meaning that RPL establishes a reliable routing topology in our wireless environments. However, the PRR degrades rapidly with the traffic load, and there are some nodes with PRR as low as $\sim$20% at heavy load. Although some level of degradation is expected due to collisions and interference caused by higher traffic, this result is much more severe than expected.

Then our first question is, *"what are the reasons for packet loss?"* To investigate this, we have collected data on how many packets are lost at the link layer and also at the packet queue within a node. We found that the link loss is negligible ($< 0.2\%$) even in a heavy traffic scenario, and most of packet losses occur at the packet queue within a node (Fig. 5). In other words, queue loss is the main reason for PRR degradation in high traffic scenarios. Furthermore, queue loss occurs very severely only at a few nodes, which are suffering extremely high relay burden.

Then our next question is, *"is unbalanced queue loss natural and unavoidable in a multihop network?"* It may be natural that nodes closer to the sink experience more relay burden and inevitably experience high queue losses in high traffic scenarios. However, our investigation claims otherwise as shown in Fig. 6, which depicts the queue loss ratio of each node according to its hop distance from the LBR, for both light and heavy loads. At heavy load, queue loss is significantly unbalanced even among nodes with the same hop distance from the LBR. Specifically, only one node (node 7) experiences very high queue loss ratio among 7 nodes which have hop distance of one. Likewise, only one node (node 19) suffers from severe queue loss among 5 nodes which have hop distance of two. These results lead us to infer that the unbalanced queue loss comes from the unbalanced, and thus inefficient, parent selection mechanism of RPL.

We verify our intuition using evidences in Figs. 7 and 8. First, Fig. 7 depicts subtree size of each node according to its hop distance from the LBR at traffic load of 60 ppm/node, which shows that subtree size is also significantly unbalanced. More importantly, Fig. 8 depicts queue loss ratio of each node according to its RPL subtree size at the same traffic load, which shows that queue loss ratio is roughly proportional to subtree size. Since each node generates data traffic at a same rate, having more nodes in its routing subtree means that more traffic will flow through that node. Lastly, a closer look into the data reveal that the nodes 7 and

19 which have large subtree sizes are exactly the nodes with large queue losses. Thus, we can confirm that unbalanced queue loss comes from unbalanced routing tree. In other words, queue loss is not inevitable but can be reduced significantly by designing an enhanced RPL that balances the subtree size under heavy traffic.

Our last question is, *"does the link ETX used for parent selection reflect traffic congestion?"* Our observation again indicates otherwise as shown in Fig. 9, which depicts the link ETX of each node with varying uplink traffic load. Interestingly, the ETX does not increase notably even in heavy traffic because the link capacity is higher than the queue capacity. It is due to the fact that RPL is implemented on low cost and resource constrained devices which provide queue sizes much smaller than the devices used for high rate communication such as LTE or WiFi (i.e., 1,000 packets at IP layer by default and more than 100 packets at link layer). Small queues start to overflow before traffic congestion becomes heavy enough to be recognized using ETX, and as a result, a node does not change its parent node even when the parent continuously suffers queue losses. For this reason, it is desirable to use a new routing metric and a parent selection strategy to alleviate the load balancing problem of RPL.

4.2 TinyRPL - RPL implementation with OF0

In this subsection we describe TinyRPL, i.e., the default RPL implementation in TinyOS 2.1.2 (latest), which implements the RPL standard [9] with OF0 along with the hop count metric for rank calculation and the ETX for parent selection.

RPL broadcasts the routing information using DIO messages which are transmitted based on the *TrickleTimer* [38] to achieve a balance between control overhead and fast recovery. To this end, the *TrickleTimer* doubles the broadcast period after every DIO transmission and re-initializes it to a minimum value when route inconsistency is detected. Furthermore, *RANK* is defined and used by the OF to represent the routing distance from a node to the LBR, and link and node metrics are used for *RANK* calculation and parent selection.

TinyRPL with OF0 uses *hop count* for *RANK* calculation, and together with *ETX* for parent selection. Specifically, *RANK* of node k is defined as

$$RANK(k) = h(k) + 1 \tag{1}$$

where $h(k)$ is the hop count between node k and the LBR. That is, $RANK(LBR) = 1$, and $RANK(k) = \infty$ before node k joins the network. Node k broadcasts DIO messages containing $RANK(k)$. $ETX(k, p_k)$ measured by node k is a link quality indicator between node k and its parent candidate p_k, and is defined as

$$ETX(k, p_k) = \frac{\text{\# of total transmissions from } k \text{ to } p_k}{\text{\# of successful transmissions from } k \text{ to } p_k}. \tag{2}$$

RPL smoothes the ETX using an exponentially weighted moving average (EWMA) filter, making it robust to sudden changes in link condition.

Each node recognizes its neighbor nodes by DIO messages received from them. Node k generates its parent candidate set $\mathbf{P}_k$ from its neighbor set $\mathbf{N}_k$ as

$$\mathbf{P}_k = \{ n_k \in \mathbf{N}_k \,|\, h(n_k) < h(k),\ ETX(k, n_k) < \delta \} \tag{3}$$

where δ is a threshold to remove neighbors which are connected through unreliable links.

Each node performs parent selection process when its information on parent candidates has been changed. Node k selects its best alternative parent $\hat{P}_k$ as

$$\hat{P}_k = \arg \min_{p_k \in \mathbf{P}_k} \{ R(p_k) \} \tag{4}$$

where $R(p_k)$ is a routing metric given as

$$R(p_k) = RANK(p_k) + ETX(k, p_k). \tag{5}$$

Then, it changes its parent node from the current parent P_k to the best alternative $\hat{P}_k$ if

$$R(\hat{P}_k) < R(P_k) - \sigma \tag{6}$$

where σ is a stability bound to mitigate unnecessary and inefficient parent changes, which is set to 0.5 by default. This is a hysteresis component (similar to MRHOF) of TinyRPL, and we refer to it as the stability condition. Thus, RPL allows each node to select a parent node which has a reliable link and minimum hop distance to the LBR, regardless of traffic load.

5 *QU-RPL*: Queue Utilization based RPL

In this section, we propose *QU-RPL* that is simple, but has the capability of load balancing which is lacking in RPL.

5.1 Routing Metric including Congestion Information

As we revealed in the previous section, ETX is not suited for early detection of traffic congestion in LLNs. It is due to the fact that link losses occur much later than queue losses when congestion starts to grow and becomes heavy enough to be detected by ETX. The number of nodes in the downward routing table (i.e. subtree size) may be an option to use for load balancing. However, because this information is updated by reception of Destination Advertisement Object (DAO) messages (addition) and timeouts of routing table entries (deletion), it does not reflect the actual congestion promptly enough.

For these reasons, we choose to use *queue utilization (QU) factor* at each node k, $Q(k)$, for detecting and distributing information about congestion, which is defined as

$$Q(k) = \frac{\text{Number of packets in queue of node } k}{\text{Total queue size of node } k}. \tag{7}$$

QU-RPL applies the same EWMA filter for statistical calculation of $Q(k)$ as for ETX calculation. Furthermore, node k adjusts $Q(k)$ after each parent selection using $Q(k)$ and $Q(P_k)$, according to

$$Q(k) = \max \{ Q(P_k) - \lambda,\ Q(k) \}. \tag{8}$$

The intuition behind this adjustment comes from our observation that when congestion occurs at the parent of node k, $Q(P_k)$ is usually significantly greater that $Q(k)$. That is, $Q(k)$ may be small even when P_k is severely congested. However, although a node k has low QU, it is better not to be selected as a parent by other neighbors if its parent node suffers from severe congestion. This is because node k forwards all the packets received from its children to P_k after all. Thus, *QU-RPL* performs the above QU adjustment (Eq. 8) to lower the probability of a node being selected when its parent node is congested, where λ is a small positive QU reduction factor which makes $Q(k) < Q(P_k)$. We set λ as 0.25, which means that a congested node can trigger QU adjustment for up to three-hop children nodes.

Finally, for *QU-RPL*, $R(p_k)$ is replaced with a new routing metric $R_{QU}(p_k)$, defined as

$$R_{QU}(p_k) = h(p_k) + 1 + ETX(k, p_k) + \alpha Q(p_k) \quad (9)$$

where α is a coefficient which controls the weight given to the QU with respect to the hop count and ETX metric. α should be greater than one for QU to have a notable effect on the parent selection. Otherwise, $Q(p_k)$ has smaller effect than $h(p_k)$ since $0 \le Q(p_k) \le 1$. We discuss and evaluated the impact of optimal α selection in detail later in Section 7.5.

5.2 Propagation of Congestion Information

In *QU-RPL*, each node distributes its QU information to its neighbor nodes. There are several ways of implementing this within the RPL standard. One way is to use an optional *Metric Container* within the DIO message. For doing so, *QU-RPL* needs to newly define the *QU Metric Container* and modify the OF to use it while processing the DIO. Another way is to modify only the OF and redefine *RANK* to contain the QU value together with the previously defined *RANK* (i.e., hop count). Both approaches are within the scope of RPL standard [9], and thus standard compliant thanks to the flexibility and openness of RPL.

To this end, in our *QU-RPL* implementation, we take the latter approach and newly define $RANK_{QU}(k)$ for the DIO message to contain both $h(k)$ and $Q(k)$ as follows:

$$RANK_{QU}(k) = \beta(h(k)+1) + (\beta - 1)Q(k) \quad (10)$$

where β is a cipher parameter used to embed and decode two values from a single numeric field [2]. Each node broadcasts the DIO message containing $RANK_{QU}$, and extracts the hop count and the QU of a neighbor node n from its received $RANK_{QU}(n)$ as

$$h(n) = \left\lfloor \frac{RANK_{QU}(n)}{\beta} \right\rfloor - 1, \quad (11)$$

$$Q(n) = \frac{\mathrm{mod}(RANK_{QU}(n), \beta)}{\beta - 1}, \quad (12)$$

respectively, where $\lfloor x \rfloor$ is the largest integer which is smaller than or equal to x and $mod()$ is modulo operation. Our *QU-RPL* implementation provides each node with the QU information of neighbor nodes without changing the message format nor adding any optional field to the DIO message.

However, the current DIO transmission strategy in RPL does not provide QU information in a timely manner due to its long *TrickleTimer* period since it is reset to a minimum only when changes in routing topology occur. To alleviate this problem, we also reset the *TrickleTimer* period when a node experiences a certain number of consecutive queue losses[3]. While doing this, however, we need to be careful to minimize the frequency of *TrickleTimer* re-initialization since it increases routing overhead (i.e., DIO messages). Based on these observations, Fig. 10 describes our strategy for *TrickleTimer* reset which balances the speed and overhead of QU information dissemination. Basically, it counts the number of queue loss events and resets the *TrickleTimer* only when it experiences a certain number (φ) of consecutive queue losses and QU indicates congestion (i.e., QU satisfies the congestion condition (14), explained below in subsection 5.4). After the reset, it additively increases the φ threshold to slow down the reset interval and balance the overhead caused by the reset. Otherwise, if no queue loss event occurs during the *noLoss* timeout period, then the algorithm returns to the initial settings. This reset strategy allows a node to fast propagate its QU information when it is congested, which helps its children nodes to quickly move to another parent nodes, while keeping the overhead to a minimum.

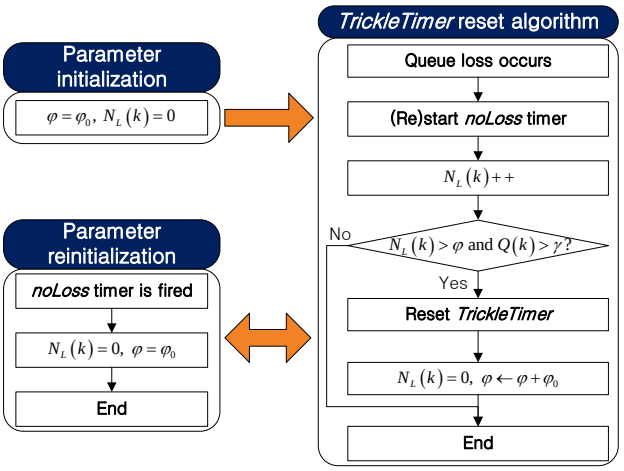


Fig. 10. *TrickleTimer* reset strategy in *QU-RPL*.

5.3 Congestion Indicator μ_k

Before we discuss the parent selection mechanism of *QU-RPL*, we first need to define the congestion indicator μ_k that detects congestion and triggers the parent change for load-balancing. For the congestion indicator μ_k, there are some candidates such as $Q(k)$ and $Q(P_k)$. We empirically observed that $Q(k)$ is much smaller than $Q(P_k)$ even when P_k experiences a lot of queue losses, and thus it cannot reflect the traffic congestion properly. Moreover, when $Q(k)$ is large, the parent selection of node k cannot help reducing the load. Rather, it is more important to let its children nodes migrate to another parent. We also observed that $Q(P_k)$ is a better congestion indicator than $Q(k)$, but still insufficient to meet our needs. This is because, once the traffic load is well balanced after a congestion event as a result of *QU-RPL* in action, each node has low $Q(P_k)$ and the congestion condition (14) is not satisfied. Thus each node changes its parent node in the same way as in RPL, causing the traffic load to be unbalanced again.

Therefore, we consider exploiting $Q_{k,max}$ which is the maximum QU among all the parent candidates that node k had in the recent past, represented as

$$Q_{k,\max} = \max\left\{ \max_{i\in\{1,2,3,4\}} Q_{k,\max}^{[i]},\ \max_{p_k \in \mathbf{P}_k} \{Q(p_k)\} \right\} \quad (13)$$

where $Q_{k,\max}^{[i]}$ is the $\max_{p_k\in\mathbf{P}_k}\{Q(p_k)\}$ in the recent i^{th} hour. In other words, we design each node k to record the $Q_{k,\max}^{[i]}$ every hour, maintain this record for the recent 4 hours, use the sliding window to update this information every hour, and then use

2. β can be any reasonable positive integer (e.g. 10 or 100) that keeps the $RANK_{QU}(k)$ within its 16 bits boundary and allows for deterministic decoding of the two values $h(n)$ and $Q(n)$. For our experiments, we have used $\beta = 100$.

3. It is important whether and when to detect and indicate congestion in response to queue fill up. Since an LLN node has small queue size, the queue can fill up temporarily even when there is no congestion. Thus, if we declare congestion too early, that often results in false positive for congestion, and incurs unnecessary DIO overhead. This is the reason why we reset *TrickleTimer after* experiencing consecutive queue losses.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2016.2585107, IEEE Transactions on Mobile Computing

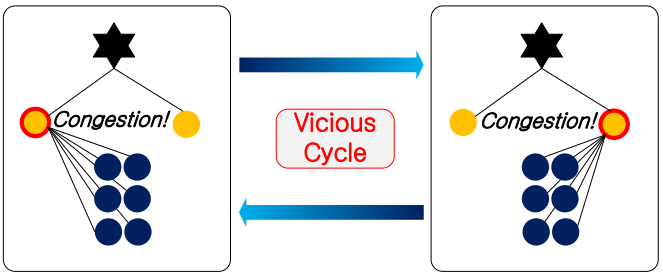


Fig. 11. An illustration of *herding effect* in *QU-RPL* without probabilistic parent change.

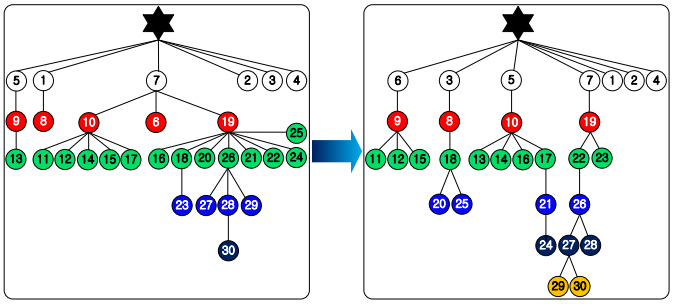


Fig. 12. Routing topology change from the default RPL to *QU-RPL*.

the maximum of these values to calculate $Q_{k,\max}$. With the use of $Q_{k,max}$, each node memorizes the previous congestion event, mitigating hasty parent changes that may occur when the traffic load has been recently balanced by exploiting QU under heavy traffic.

5.4 Parent Selection Mechanism

The two design elements of *QU-RPL* introduced in the previous subsections allow each node to consider congestion when selecting its parent node. However, only using congestion-aware routing metric and fast QU propagation mechanism brings another challenge, termed *herding effect*. Fig. 11 depicts an example of the *herding effect* where all the six 2-hop nodes select the 1-hop node on the left as their parent and incur congestion. The use of new routing metric and QU propagation in *QU-RPL* allows the children nodes to escape from the left parent node by detecting congestion. The problem is that all of them may change the parent simultaneously, which incurs congestion again on the right 1-hop node. In this way, a vicious cycle could be created, where a group of nodes repeat meaningless parent changes indefinitely without achieving load balancing.

To avoid the *herding effect*, we add a *probabilistic parent change* mechanism to our *QU-RPL* design. Each node generates a parent candidate set from its neighbor nodes according to Eq. 3, and opportunistically includes neighbor nodes with the same rank[4] to diversify candidate parents to avoid congestion. Then, it selects the best alternative (new) parent node using Eq. 4 (same as in RPL) while substituting the routing metric with Eq. 9. However, *QU-RPL* operates differently from RPL when determining whether to switch from the current parent to the best alternative or not. *QU-RPL* considers not only the stability condition given by Eq. 6 but also the traffic congestion condition given by

$$\mu_k > \gamma \tag{14}$$

where μ_k is the congestion indicator that indicates the traffic congestion around node k, and γ is a threshold value for deciding when to perform load balancing. If the condition (14) is satisfied, node k needs to change its parent node considering load balancing. Otherwise if only the stability condition is satisfied, node k in *QU-RPL* changes its parent node from P_k to $\hat{P}_k$ in the same way as in the default RPL. In other words, *QU-RPL* has the same parent change mechanism as RPL when a node does not experience congestion.

4. A node includes neighbor nodes with the same *RANK* in the parent candidate set when receiving their DIOs, and removes them when not selected as the parent node.

We empirically chose γ as 0.5 to reflect the idea that our load balancing scheme should come into action when the QU of a congested node is above 50%. If we had selected γ to a lower value, unnecessary load balancing action might take place even when there is no real congestion. This will only cause unnecessary additional overhead (in terms of DIOs and DAOs) when the current topology is capable of supporting the given traffic load. If we had selected γ to a higher value, that would delay the detection of congestion and load balancing effort, which could result in increased packet losses.

Then, most importantly, if both the stability condition and the congestion condition are satisfied, node k changes its parent node from P_k to $\hat{P}_k$ with the probability

$$\max\left\{\kappa\left(Q\left(P_k\right)-Q\left(\hat{P}_k\right)\right),\ 0\right\} \tag{15}$$

where κ is a non-negative coefficient which controls what percentage of children nodes change their parents. That is, *QU-RPL* requires a node to probabilistically change its parent considering its QU differences. This probabilistic parent change is a critical component of *QU-RPL* to avoid the *herding effect*, and we will discuss and evaluate the impact of optimal κ selection on performance in detail in Section 7.5.

5.5 Memory footprint

Overall, our *QU-RPL* implementation requires 4,018 bytes of extra ROM and 22 bytes of extra RAM compared to the current TinyRPL implementation. That is 10% increase for the ROM (39,516 $\rightarrow$ 43,534 bytes) and only 1% increase for the RAM (7,402 $\rightarrow$ 7,424 bytes), and it operates well on low cost embedded devices such as TelosB.

6 Performance Evaluation - Comparison with RPL

In this section, we evaluate the performance measurement results for *QU-RPL* on the testbed setup, and compare it against TinyRPL.

6.1 Routing Topology

First of all, we graphically describe the effect of *QU-RPL* on topology construction using Fig. 12, which depicts a snapshot of the routing topology at the end of experiments for RPL and *QU-RPL*. Let us consider the physical topology in Fig. 2 again for analysis. In RPL, when comparing subtrees of nodes 10 and 19, we can see that node 19 has nearly half of all the nodes in the network as its children nodes, which is more than twice larger than what node 10 has. As a result, node 19 experienced significant

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2016.2585107, IEEE Transactions on Mobile Computing

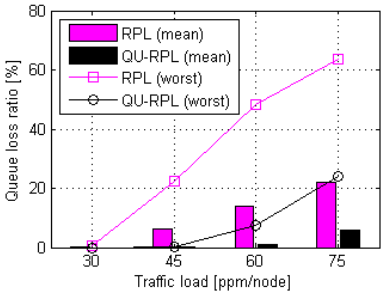


Fig. 13. Loss ratio at packet queue vs. uplink traffic load.

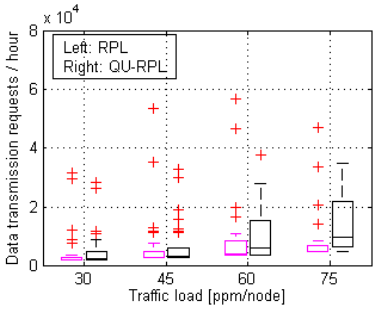


Fig. 14. Data transmission requests of each node vs. uplink traffic load.

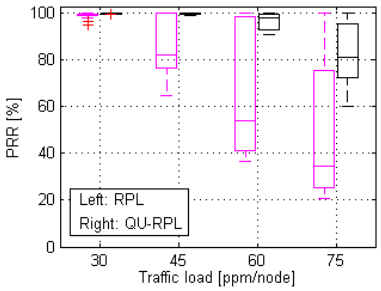


Fig. 15. PRR of each node vs. uplink traffic load.

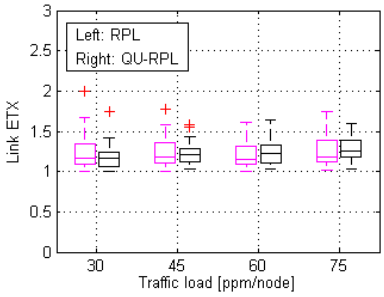


Fig. 16. Link ETX of each node to its parent node vs. uplink traffic load.

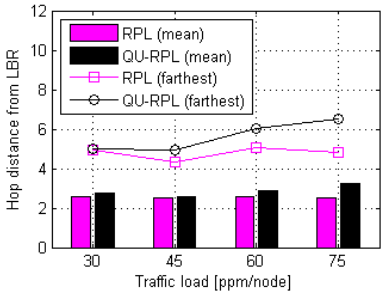


Fig. 17. Hop distance from the LBR vs. uplink traffic load.

relay burden and dropped a lot of packets at its queue. However, the more critical problem is that both nodes 10 and 19 have node 7 as their parent node, which incurs an extremely large number of queue losses at node 7 and causes significant PRR degradation. Nodes 7 and 19 are the two which experienced severe queue loss in Fig. 6. RPL cannot escape from this unbalanced topology because their children nodes are unaware of the severe congestion. From the children nodes' perspective, they are able to successfully transmit almost all the packets to their parents, which is expected by their low ETXs.

In contrast, in *QU-RPL*, although node 19 has the largest number of nodes in its subtree, the number of children nodes of node 10 is only one less than that of node 19, showing balanced load distribution. Furthermore, nodes 8 and 9 have larger subtrees compared to the RPL case. More importantly, nodes 10 and 19 have different parent nodes (i.e., nodes 5 and 7, respectively), which significantly decreases queue loss and improves PRR. Specifically, nodes 16 and 18 do not select node 19 as their parent node even though node 19 is with better link quality, resulting in successful avoidance of traffic congestion. Furthermore, nodes 20, 21, 24, and 25 do not select node 19 even though their current choices have resulted in hop distance increase. The smart use of QU leads those nodes to avoid traffic congestion by rejecting node 19 even though it has smaller hop distance. *QU-RPL* also prevents them from selecting a child node of node 19 as their parent node. This is because, through the QU adjustment (8), *QU-RPL* lowers the probability that a node having a congested parent is selected as a parent node.

Quantitatively, the standard deviation of the number of children nodes per node has decreased from 1.79 to 1.04, and that of the number of nodes in subtree per node has decreased from 4.47 to 2.25, which shows that *QU-RPL* indeed achieves load balancing of children nodes.

6.2 Packet delivery performance

Fig. 13 depicts the queue loss ratio (i.e., dropped packets divided by injected packets into the IP layer packet queue) with varying uplink traffic load. We observed that *QU-RPL* reduces the queue loss ratio significantly, up to 84%, especially at bottlenecked nodes. This reveals that the balanced tree topology of *QU-RPL*, as shown in Fig. 12, has a critical impact on congestion mitigation, and thus enables *QU-RPL* to achieve lower and fairer queue loss compared to RPL.

To show more details, we use Fig. 14 which depicts the data transmission burden of each node with varying uplink load where outliers are marked with plus symbols. From this figure, we directly observe that *QU-RPL* evens out the traffic load much better than RPL. *QU-RPL* significantly reduces the data transmission burden on most congested nodes, while only slightly increasing those of other nodes. This is the main cause of the queue loss reduction shown in Fig. 13.

The reduction in queue losses is directly translated into PRR improvement, as shown in Fig. 15 which depicts the PRR performance of each node for RPL and *QU-RPL* with varying uplink load. It shows that *QU-RPL* enhances the PRR performance significantly for most of the nodes in the network, up to 147%. For RPL, its PRR degradation mainly comes from its inadequate parent selection, which is worsened by the small queues of embedded devices.

Fig. 16 depicts the link ETX of each node to its parent node with varying uplink traffic load. We observe that both RPL and *QU-RPL* allow a node to select its parent with good link quality since the measured ETX is at most 2 in all cases. Moreover, the link ETX results for both protocols do not vary significantly with different traffic loads, revealing that the wireless link capacity is not the main cause of packet loss even under heavy traffic.

Fig. 17 depicts the hop distance of a node from the LBR with varying uplink traffic load. By comparing the average hop distance

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2016.2585107, IEEE Transactions on Mobile Computing

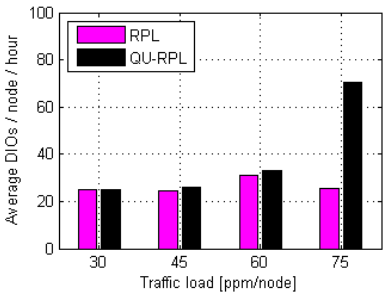


Fig. 18. Average DIO overhead of each node vs. uplink traffic load.

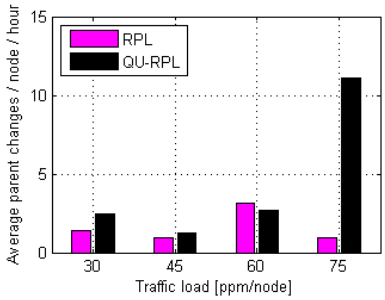


Fig. 19. Number of parent changes of each node vs. uplink traffic load.

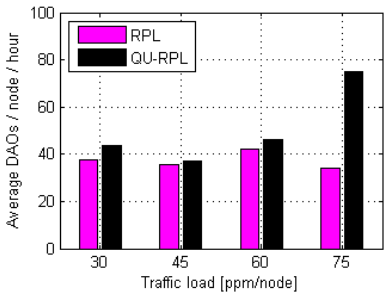


Fig. 20. Average DAO overhead of each node vs. uplink traffic load.

and the hop distance of the farthest node, we can see that *QU-RPL* shows similar hop distances to RPL while achieving load balancing. It is mainly because *QU-RPL* exploits the similar criterion of hop distance as RPL when generating a parent candidate set. Since a node's neighbor nodes with large hop distances are already removed from its parent candidate set, parent selection considering QU causes only slight increase in its hop distance from the LBR. Moreover, the QU weighting factor α of $R_{QU}(k)$ can be tuned considering the trade-off between hop distance and load distribution. For instance, for $\alpha = 2$, when a node experiences severe congestion, its parent node has the distance of at most one hop greater than that given by RPL.

From the results in Fig. 13 through 17, we find that simply providing a multihop route with good link quality (ETX) for each node is insufficient for large scale applications with resource constrained (i.e., small queue sized) devices. *QU-RPL* requires low implementation cost and low operation overhead, but achieves significant improvement in packet delivery performance by using hop distance and QU information together.

6.3 Routing Overhead

Before discussing the routing overhead (DIO and DAO messages) of RPL and *QU-RPL*, we need to first understand the ranges of traffic load. In general, light traffic load means that the network is capable of handling the traffic without congestion at any part of the network. However, heavy traffic load has two ranges; one where congestion may or may not occur depending on the parent selection and load balancing, and the other 'extremely heavy' case where congestion is unavoidable even after load balancing with 'good' parent selection. *QU-RPL* aims to solve the former case.

Fig. 18 plots the average DIO overhead of each node under varying uplink traffic load. Each node in *QU-RPL* generates extra DIO overhead since it resets its *TrickleTimer* more frequently than in RPL to fast distribute the QU information when it suffers from consecutive queue losses. When the traffic load is extremely heavy (i.e., 75 ppm/node) where the channel bandwidth is close to saturation and the PRR performance is low even with use of *QU-RPL*, *QU-RPL* incurs considerably more DIO overhead than RPL. This is because the traffic load is too heavy such that the congestion problem cannot be solved via load balancing, but *QU-RPL* is continuously trying to select less congested parents. However, the increase in DIO overhead of *QU-RPL* is insignificant compared to the total amount of traffic in the network and also the great reduction in relay burden of congested nodes.

Even in the lightest traffic scenario where a node generates 1,800 data packets per hour, the amount of data traffic is 36 times more than that of DIO traffic. In other words, overhead is less than 3% of the data traffic. Furthermore, RPL requires the most bottlenecked node to transmit more than 30,000 data packets per hour in the lightest traffic scenario as shown in Fig. 14, thus its overhead is less than 0.3% of the forwarding traffic. Since *QU-RPL* significantly reduces the data transmission burden of bottlenecked nodes and packet losses at those queues, their overall transmission cost is reduced greatly. We can conclude that increase in DIO overhead of *QU-RPL* is a reasonable cost to pay for much better PRR performance when delivering heavy traffic.

Fig. 19 depicts the average number of parent changes of each node with varying uplink traffic load. First of all, the number of parent changes in RPL is not strongly correlated to traffic load since RPL triggers parent changes using ETX which stays low regardless of traffic load as shown in Fig. 16. It is also observed that the frequency of parent change in *QU-RPL* is similar to that in RPL when the traffic load increases up to 60 ppm/node, but significantly higher under extremely heavy traffic (i.e., 75 ppm/node). This is because, under extremely heavy traffic where even the load balancing cannot resolve the congestion, *QU-RPL* continuously attempts to resolve the problem by new parent selection.

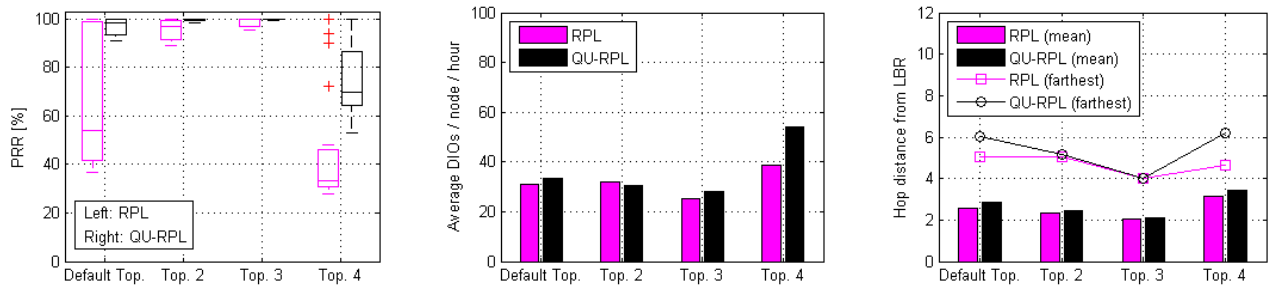
In RPL, there is another routing overhead of transmitting DAO messages which are used for downlink route setup between a node and the LBR. Each node sends DAO messages toward the LBR periodically[5] when its route is consistent, and also when its upstream route has been changed. Fig. 20 plots the average DAO overhead of each node under varying uplink traffic load. It shows that *QU-RPL* requires a similar amount of DAO overhead compared to RPL up to 60 ppm/node, but it has much higher overhead when load balancing cannot resolve the congestion due to the same reason as above.

6.4 Effect of Topology Variation

In this subsection, we experiment with several different physical topologies and investigate whether and how topology variation affects the performance of *QU-RPL*. For this purpose, we created multiple topologies on the same testbed deployment (Fig. 2) by modifying the location of the LBR (root node) and adjusting the transmission power. Our indoor testbed environment has quite uneven node density due to obstacles such as doors, walls, windows, etc., which helps us to generate various topologies. Specifically, the *'default topology'* refers to the topology in Fig. 2 that we have been using so far throughout the paper, *'topology 2'* refers to a

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2016.2585107, IEEE Transactions on Mobile Computing



(a) PRR performance (b) DIO overhead (c) Hop distance

Fig. 21. Performance of RPL and *QU-RPL* with various topologies.

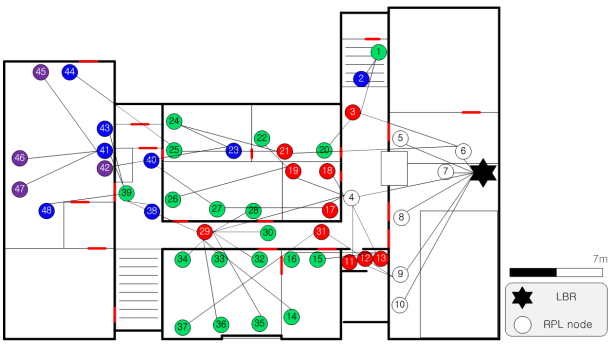


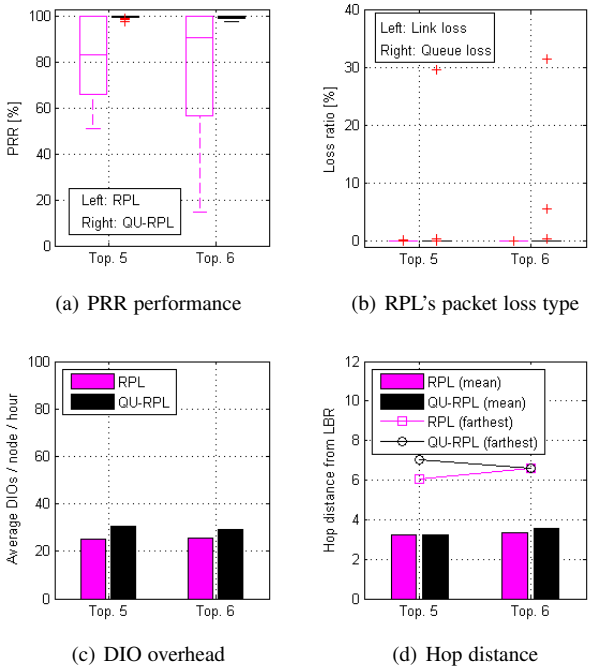
Fig. 22. Testbed topology map for scalability test (49 nodes) with a snapshot of routing path given by RPL.

topology where node 9 acts as the LBR with transmission power of -22dBm, *'topology 3'* refers to a topology where node 14 acts as the LBR with transmission power of -22dBm, and *'topology 4'* refers to a topology where node 30 acts as the LBR with transmission power of -9dBm. For topologies 2 through 4, the node that was the LBR of the 'default topology' acts as a regular data node, and thus the total number of data nodes remain as 30.

Fig. 21 compares the performances of *QU-RPL* and RPL for the considered four topologies when each node generates upward traffic with 60 ppm. We first observe that performances of both RPL and *QU-RPL* significantly rely on topology configuration. However, in all cases, *QU-RPL* achieves much better PRR performance than RPL by balancing traffic load. Results of DIO overhead and hop distance show the same trends as we explained above. Thus, we can conclude that *QU-RPL* achieves more reliable packet delivery than RPL with slightly larger routing overhead and hop distance, regardless of node placement and transmission power setting.

6.5 Scalability - a short glimpse

To get an idea of the scalability of *QU-RPL*, we constructed another larger indoor testbed with 49 nodes (one LBR and 48 LLN enpoints) as shown in Fig. 22 at the basement of our university building. Note that this testbed is physically distinct from our earlier testbed which was at the 3rd floor of our building. On this new testbed, we made each node use transmission power of -22dBm, which forms a 5-hop network using RPL. We created two topologies in this larger testbed. '*topology 5*' is illustrated in Fig. 22 and '*topology 6*' refers to a topology where node 45 acts as the LBR and the LBR of the '*topology 5*' becomes a data node. Lastly, each node generates upward traffic with 36 ppm, which



(a) PRR performance (b) RPL's packet loss type

(c) DIO overhead (d) Hop distance

Fig. 23. Performance of RPL and *QU-RPL* with two topologies in a 49-node testbed.

makes the LBR experience traffic load similar to the 60 ppm/node case in the 30-node testbed.

Figs. 23(a) through 23(d) compare various performance metrics of *QU-RPL* and RPL in the two topologies. Fig. 23(a) shows that RPL experiences significantly more packet losses (low PRR) than *QU-RPL* in both topologies. Furthermore, Fig. 23(b) reveals that this low PRR of RPL does not come from link losses but queue losses at small number of nodes. These results confirm that RPL experiences severe load balancing problem in this larger testbed as well, which implies that the observations and arguments that we have made in the earlier sections are still valid in this larger network as well. That is, the load balancing issue could also be problematic in a large-scale network.

More importantly, we observe that *QU-RPL* achieves more dramatic performance improvement over RPL compared to the 30-node cases. Specifically, *QU-RPL* improves average PRR from 80.53% to 99.65% in '*topology 5*' and from 76.69% to 99.39% in '*topology 6*'. *QU-RPL* improves PRR more drastically for the worst performing node, from 51.16% to 97.41% in '*topology 5*' and from 14.58% to 97.78% in '*topology 6*'. We believe this is because larger (and deeper) network has more opportunity to be-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2016.2585107, IEEE Transactions on Mobile Computing

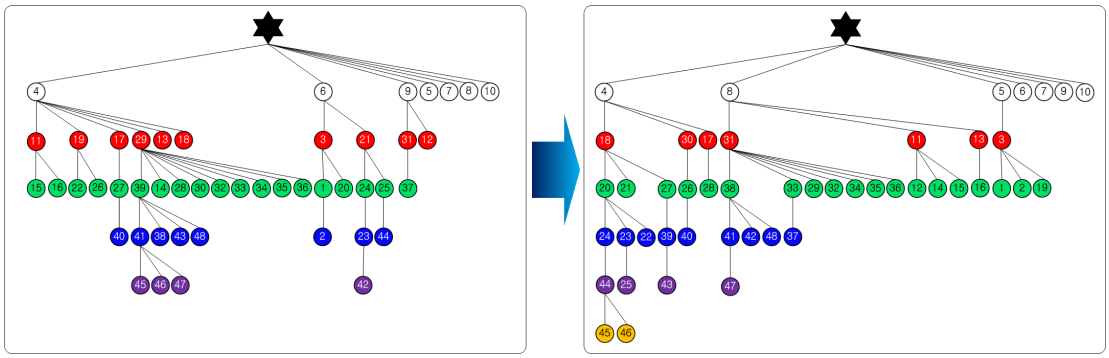
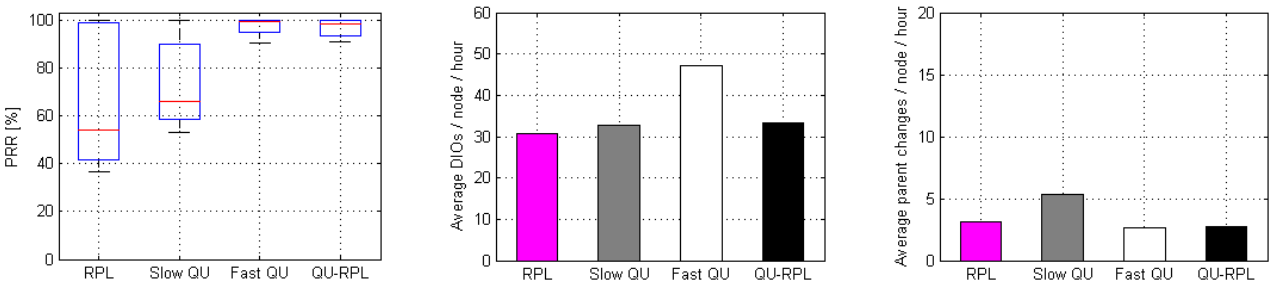


Fig. 24. Routing topology change from the default RPL to *QU-RPL* in '*topology 5*' of a 49-node testbed.



(a) PRR performance (b) DIO overhead (c) Parent changes

Fig. 25. The effect of fast QU propagation mechanism on performance.

come unbalanced in terms of tree construction, which makes load balancing more important and effective. We also believe that the large number of forwarding traffic that travels over the multihop network is what makes larger & deeper multihop networks more challenging than small & shallow networks in terms of congestion, and thus load balancing problem becomes more critical.

Fig. 24 graphically depicts the load balancing effect in '*topology 5*' similar to Fig. 12. We observe that both RPL and *QU-RPL* have 7 nodes that are connected to the LBR in a single hop, only three of which have subtree nodes due to the constraint of given physical topology. However, when taking a deeper look, we can find out that the two protocols provide quite different subtrees. RPL constructs the three subtrees with size of 3, 10 and 28, respectively, which clearly shows unbalanced topology. In contrast, *QU-RPL* forms these subtrees with size of 4, 18 and 19, which achieves load balancing. Specifically, our *QU-RPL* reduces standard deviation of the subtree size from 4.8 to 4.3, and size of the largest subtree from 28 to 19.

Moreover, by combining the results of Figs. 23(b) and 24, we confirm that the node which suffers from extremely large number of queue losses when operating RPL in '*topology 5*' was node 4 (i.e., parent node of the largest subtree with 28 children nodes). This verifies that queue losses of RPL come from unbalanced tree structure. We have also verified that *QU-RPL* constructs balanced tree structure in '*topology 6*' as well (figure omitted for brevity). All the results reveal that performance improvement of *QU-RPL* comes from its balanced tree topology.

Overall, our experiments on a 49-node testbed verify that both RPL and *QU-RPL* exhibit similar behavior as we observed in the 31-node cases. Furthermore, our experimental results show that impact of load balancing on the performance becomes more significant in a larger network. As a final note, even though our experiments cannot cover the case of real large-scale networks comprising thousands of nodes, we believe that this load balancing problem will also occur in these large networks due to two reasons. First, environmental factors in real LLN deployments such as obstacles and human activity complicate link characteristics, which makes some nodes have much more neighbor nodes than others even if nodes are uniformly deployed. Second, given that an LLN node is resource-constrained and has small queue size, higher volume of forwarding traffic in a large-scale multihop network can cause severe queue losses.

7 Effect of Design Elements

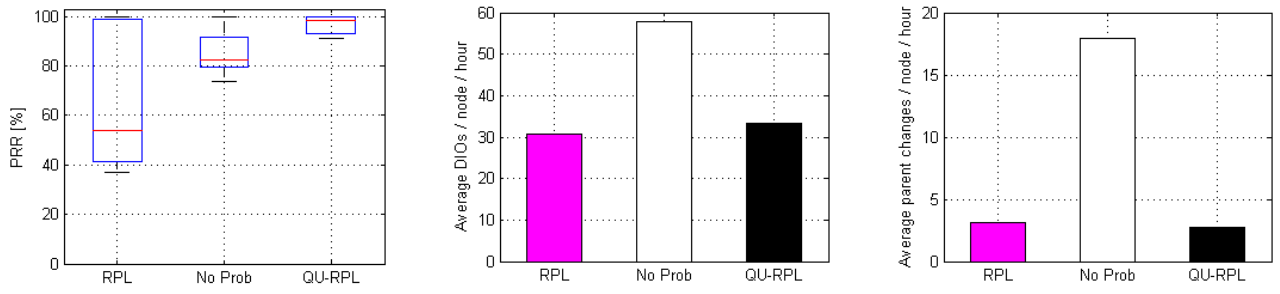
In this section, we investigate the effect of each design element in *QU-RPL* on the performance. Our aim is to verify that each component of *QU-RPL* contributes to achieving performance improvement. To this end, we selectively remove each design element of *QU-RPL* from the full version of *QU-RPL*, and compare the performance of each implementation with the full *QU-RPL* and the default RPL when each node generates upward traffic with 60 ppm. We consider PRR, DIO overhead, and the frequency of parent change as the evaluation metrics.

7.1 *TrickleTimer* Resetting Strategy

To evaluate the effect of *TrickleTimer* resetting strategy in *QU-RPL*, which fast propagates the QU information of a node when it experiences congestion, we present Fig. 25 with two additional schemes; '*Slow QU*' and '*Fast QU*'. '*Slow QU*' uses the same *TrickleTimer* as RPL which does not reset the DIO broadcast period even when a node suffers from congestion. '*Fast QU*' resets *TrickleTimer* whenever a node experiences more queue losses than a predetermined threshold (Section 5.2).

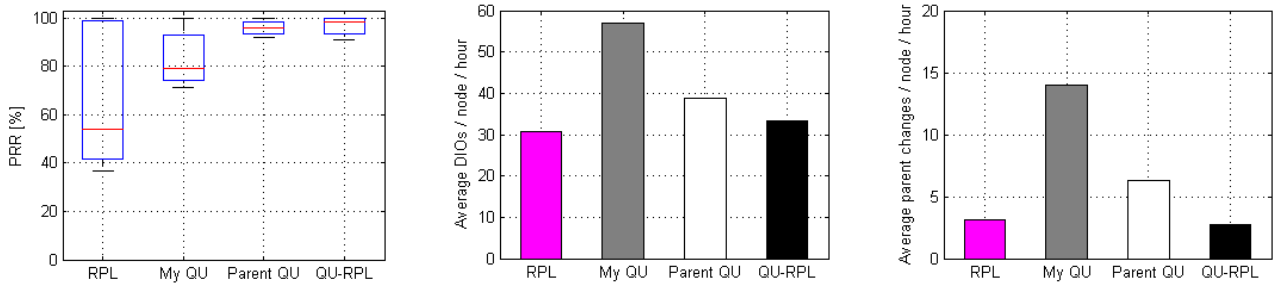
From Fig. 25(a), we observe that *QU-RPL* and its variants provide better PRR than RPL thanks to its design elements other

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2016.2585107, IEEE Transactions on Mobile Computing



(a) PRR performance (b) DIO overhead (c) Parent changes

Fig. 26. The effect of probabilistic parent change on performance.



(a) PRR performance (b) DIO overhead (c) Parent changes

Fig. 27. The effect of congestion indicator on performance.

than *TrickleTimer* resetting strategy. Furthermore, '*Slow QU*' shows the worst performance among the three *QU-RPL*-related ones since it cannot fast disseminate the QU information of a congested parent node to its children nodes. The slow update of QU makes it hard to balance traffic load since each node changes its parent node based on inaccurate and outdated QU information. As a result, '*Slow QU*' repeats meaningless parent changes again and again, which causes the largest number of parent changes as shown in Fig. 25(c).

On the other hand, '*Fast QU*' greatly improves the PRR performance with less parent changes than '*Slow QU*' by propagating real-time QU information when congestion occurs. However, as shown in Fig. 25(b) , it requires the largest DIO overhead due to frequent re-initialization of *TrickleTimer* . Thus, this result indicates that *QU-RPL* achieves the same performance as the better one of either '*Slow QU*' or '*Fast QU*' by re-initializing *TrickleTimer* with an adaptive threshold φ which increases when a node continuously suffers from congestion.

7.2 Probabilistic Parent Change

Fig. 26 shows the effect of probabilistic parent change mechanism in *QU-RPL*, which is designed to mitigate *herding effect*. Here '*No Prob*' selects the best alternative parent using the same routing metric $R_{QU}(p_k)$ as *QU-RPL*, but changes the parent from the current one to a newly selected one as RPL (i.e., no use of Eq. 15).

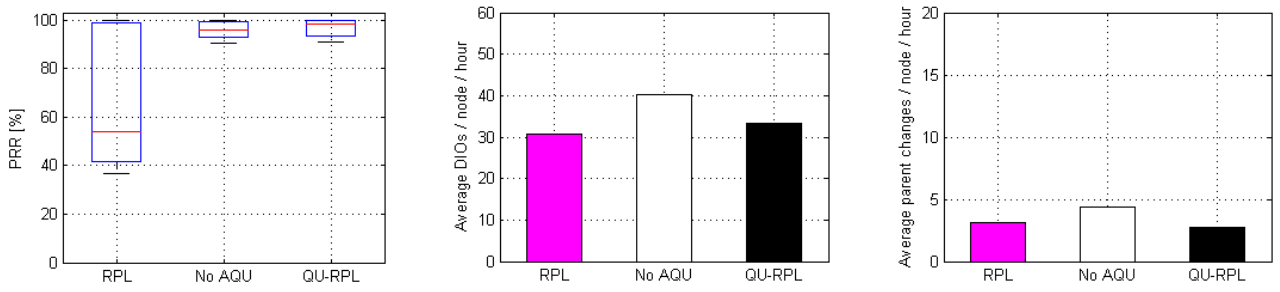
Based on the routing metric including QU, '*No Prob*' detects congestion and tries to resolve the problem by changing parents and transmitting DIOs much more frequently than RPL and *QU-RPL*, as shown in Fig. 26(b) and 26(c). However, Fig. 26(a) shows that, even though '*No Prob*' improves PRR performance compared to RPL, it still provides worse PRR than *QU-RPL*. This is because '*No Prob*' allows each child node of a congested parent to be simultaneously attached to its best alternative parent, resulting in repetitive and meaningless parent changes with limited performance improvement.

7.3 Congestion Indicator μ_k

Fig. 27 shows the effect of congestion indicator μ_k on the performance. Here '*My QU*', '*Parent QU*', and *QU-RPL* use $Q(k)$, $Q(P_k)$, and Eq. 13 as μ_k, respectively. Firstly, Fig. 27(a) reveals that *QU-RPL* and its variants outperform RPL thanks to other design elements in *QU-RPL* except the congestion indicator. Among the three *QU-RPL*-related protocols, '*My QU*' provides the worst PRR performance since $Q(k)$ could be small even when a parent node of node k experiences severe congestion (i.e., large $Q(P_k)$). Node k cannot detect the congestion of its parent node with use of $Q(k)$ and incurs *herding effect* because the condition (14) is not satisfied. As a result, '*My QU*' incurs the largest DIO overhead and the largest number of parent changes, as depicted in Fig. 27(b) and 27(c).

'*Parent QU*' significantly improves PRR performance over '*My QU*' by using $Q(P_k)$, which means that $Q(P_k)$ is a more desirable congestion indicator than $Q(k)$ in the perspective of parent selection. However, '*Parent QU*' still provides PRR performance slightly lower than *QU-RPL* while incurring larger DIO overhead and more parent changes than *QU-RPL*. This is because the use of $Q(P_k)$ well balances tree topology at first, but can make it unbalanced again. Specifically, once the topology is balanced, $Q(P_k)$ becomes small, which allows node k to disable probabilistic parent change. On the other hand, $Q_{k,max}$ helps node k to memorize congestion events and maintain the probabilistic parent change. Since *QU-RPL* can balance tree topology in a more stable manner, it provides the best performance among the three competitive schemes.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2016.2585107, IEEE Transactions on Mobile Computing



(a) PRR performance (b) DIO overhead (c) Parent changes

Fig. 28. The effect of QU adjustment on performance.

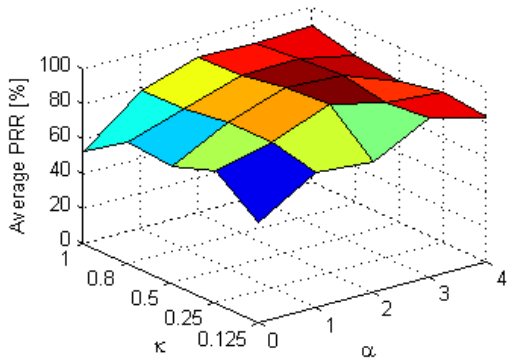


Fig. 29. Average PRR with varying α and κ, which shows the effect of parameters on performance.

7.4 QU Adjustment

Fig. 28 shows the effect of QU adjustment considering the QU of a parent node as Eq. 8. '*No AQU*' is the same as *QU-RPL* but does not use the QU adjustment in Eq. 8. From the three figures, we observe that QU adjustment slightly improves the PRR performance while reducing DIO overhead. This is because the QU adjustment mechanism increases QU of a node when it has a congested parent, which allows each node to avoid selecting a node as a parent, when it has a congested parent (thus avoiding congested grandparent). The results confirm that *QU-RPL* achieves load balancing more effectively by propagating QU information of each node to its subtree.

7.5 Weighting Factors α and κ

Lastly, we analyze the effect of the design parameters α and κ on the performance of *QU-RPL*. Fig. 29 depicts the PRR given by *QU-RPL* with varying α and κ. First of all, we observe that the PRR first increases and then decreases with α. This is due to the trade-off between congestion avoidance and routing direction. For large α, a node mainly considers QU when selecting its best alternative parent, and easily avoids traffic congestion. However, it may take a path that is significantly longer than the shortest path by ignoring hop distance information of parent candidates. We also observe that the PRR first increases and then decreases with κ as well. This is because, for large κ, a node aggressively changes its parent to avoid traffic congestion, which shows the trade-off between fast load balancing and *herding effect*.

We conclude that these parameters do impact the performance of *QU-RPL*, and they can be empirically optimized by observing network performance. The results show that *QU-RPL* provides the best PRR performance when $\alpha = 2$ and $\kappa = 0.25$. Therefore we have exploited these values throughout our testbed experiments and evaluation in Section 6.

8 Conclusion

In this paper, we have discussed the congestion and load balancing problem of the RPL standard. We have identified the cases where routing concentrates on a small set of forwarding parents resulting in packet delivery failures due to queue overflows, and also verified our findings through proof-of-concept implementation and testbed experiments. To address this issue, we have proposed a light-weight but effective solution, called *QU-RPL*, that aims to achieve load balancing by allowing each node to select its parent node according to the queue utilization of its neighbor nodes as well as their hop distances to the border router. We have also evaluated the performance of *QU-RPL* through extensive experiments on a real testbed in comparison with the TinyRPL, and proved that our proposal greatly alleviates the packet loss problem at queues, thereby achieving significant improvement in end-to-end packet delivery performance.

References

[1] H.-S. Kim, J. Paek, and S. Bahk, "QU-RPL: Queue Utilization based RPL for Load Balancing in Large Scale Industrial Applications," in *IEEE International Conference on Sensing, Communication, and Networking (SECON 2015)*, June 2015.

[2] Cisco, "Connected Grid Networks for Smart Grid - Field Area Network," http://www.cisco.com/web/strategy/energy/field_area_network.html.

[3] E. Ancillotti, R. Bruno, and M. Conti, "The role of the rpl routing protocol for smart grid communications," *Communications Magazine, IEEE*, vol. 51, no. 1, pp. 75–83, Jan. 2013.

[4] German Federal Ministry of Education and Research, "Project of the Future: Industry 4.0," http://www.bmbf.de/en/19955.php.

[5] V. Gungor and G. Hancke, "Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 4258–4265, Oct. 2009.

[6] J. Paek, B. Greenstein, O. Gnawali, K.-Y. Jang, A. Joki, M. Vieira, J. Hicks, D. Estrin, R. Govindan, and E. Kohler, "The Tenet Architecture for Tiered Sensor Networks," *ACM Transactions on Sensor Networks*, vol. 6, no. 4, pp. 34:1–34:44, 2010.

[7] J. Paek, J. Hicks, S. Coe, and R. Govindan, "Image-Based Environmental Monitoring Sensor Application Using an Embedded Wireless Sensor Network," *Sensors*, vol. 14, no. 9, pp. 15 981–16 002, 2014.

[8] H.-S. Kim, H. Cho, M.-S. Lee, J. Paek, J. Ko, and S. Bahk, "MarketNet: An Asymmetric Transmission Power-based Wireless System for Managing e-Price Tags in Markets," in *Proceedings of the 13th ACM International Conference on Embedded Networked Sensor Systems (SenSys'15)*, Nov. 2015, pp. 281–294.

[9] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," *RFC 6550*, Mar. 2012.

[10] "IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks. Specific requirements – Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)," Available at http://www.ieee802.org/15/pub/TG4.html, May 2003.

[11] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," *RFC 4944*, 2007.

[12] H.-S. Kim, J.-S. Bang, and Y.-H. Lee, "Distributed Network Configuration in Large-Scale Low Power Wireless Networks," *Computer Networks*, vol. 70, pp. 288–301, Sep. 2014.

[13] H.-S. Kim, H. Im, M.-S. Lee, J. Paek, and S. Bahk, "A Measurement Study of TCP over RPL in Low-power and Lossy Networks," *Journal of Communications and Networks*, vol. 17, no. 6, pp. 647–655, Dec. 2015.

[14] S. Bahk and M. E. Zarki, "Dynamic multi-path routing and how it compares with other dynamic routing algorithms for high speed wide area network," in *ACM SIGCOMM Conference'92*, Aug. 1992.

[15] P. Thubert, "Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL)," *RFC 6552*, Mar. 2012.

[16] D. Wang, Z. Tao, J. Zhang, and A. Abouzeid, "RPL Based Routing for Advanced Metering Infrastructure in Smart Grid," in *IEEE International Conference on Communications Workshops (ICC)*, May 2010.

[17] J. Ko, S. Dawson-Haggerty, O. Gnawali, D. Culler, and A. Terzis, "Evaluating the Performance of RPL and 6LoWPAN in TinyOS," in *Proceedings of the Workshop on Extending the Internet to Low power and Lossy Networks (IP+SN 2011)*, Apr. 2011.

[18] T. Clausen, U. Herberg, and M. Philipp, "A critical evaluation of the ipv6 routing protocol for low power and lossy networks (rpl)," in *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct. 2011.

[19] 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 *ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks*, 2011.

[20] N. Accettura, L. Grieco, G. Boggia, and P. Camarda, "Performance analysis of the rpl routing protocol," in *IEEE International Conference on Mechatronics (ICM)*, Apr. 2011.

[21] 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. 11 993–12 022, 2014.

[22] J. Ko, J. Jeong, J. Park, J. A. Jun, O. Gnawali, and J. Paek, "DualMOP-RPL: Supporting Multiple Modes of Downward Routing in a Single RPL Network," *ACM Transactions on Sensor Networks*, vol. 11, no. 2, pp. 39:1–39:20, Mar. 2015.

[23] O. Gnawali and P. Levis, "The Minimum Rank with Hysteresis Objective Function," *RFC 6719*, Sep. 2012.

[24] ——, "The ETX Objective Function for RPL," *draft-gnawali-roll-etxof-01*, May 2010.

[25] J. Vasseur, M. Kim, K. Pister, N. Dejean, and D. Barthel, "Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks," *RFC 6551*, Mar. 2012.

[26] O. Gaddour, A. Koubaa, N. Baccour, and M. Abid, "Of-fl: Qos-aware fuzzy logic objective function for the rpl routing protocol," in *International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, May 2014, pp. 365–372.

[27] P. Karkazis, H. Leligou, L. Sarakis, T. Zahariadis, P. Trakadas, T. Velivassaki, and C. Capsalis, "Design of primary and composite routing metrics for rpl-compliant wireless sensor networks," in *International Conference on Telecommunications and Multimedia (TEMU)*, Jul. 2012.

[28] A. Brachman, "Rpl objective function impact on llns topology and performance," in *Internet of Things, Smart Spaces, and Next Generation Networking*, ser. Lecture Notes in Computer Science, S. Balandin, S. Andreev, and Y. Koucheryavy, Eds., 2013, vol. 8121, pp. 340–351.

[29] P. Gonizzi, R. Monica, and G. Ferrari, "Design and evaluation of a delay-efficient rpl routing metric," in *International Conference on Wireless Communications and Mobile Computing (IWCMC)*, Jul. 2013.

[30] P. Di Marco, C. Fischione, G. Athanasiou, and P.-V. Mekikis, "Mac-aware routing metrics for low power and lossy networks," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Apr. 2013.

[31] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, J.-P. Vasseur, M. Durvy, A. Terzis, A. Dunkels, and D. Culler, "Beyond Interoperability: Pushing the Performance of Sensor Network IP Stacks," in *ACM SenSys*, 2011.

[32] M. Ha, K. Kwon, D. Kim, and P.-Y. Kong, "Dynamic and distributed load balancing scheme in multi-gateway based 6lowpan," in *IEEE/ACM iThings*, Oct. 2014.

[33] X. Liu, J. Guo, G. Bhatti, P. Orlik, and K. Parsons, "Load balanced routing for low power and lossy networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*, Apr. 2013, pp. 2238–2243.

[34] Y. Z. et al, "On deploying relays for connectd indoor sensor networks," *Journal of Communications and Networks*, vol. 16, no. 3, pp. 335–343, Jun. 2014.

[35] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling Ultra-Low Power Wireless Research," in *Proceedings of IPSN/SPOTS*, 2005.

[36] D. Moss, J. Hui, and K. Klues, "Low power listening," *TinyOS TEP 105*.

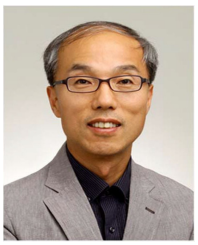
[37] K. T. Cho and S. Bahk, "Duty cycle optimization for a multi hop transmission method in wireless sensor networks," *IEEE Communications Letters*, vol. 14, no. 3, Mar. 2010.

[38] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A Self-regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," in *Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI'04)*, 2004.

Hyung-Sin Kim received the B.S. degree from Seoul National University (SNU), Seoul, Korea in 2009, the M.S. degree from SNU in 2011, and the Ph.D. degree from SNU in 2016, all in electrical engineering. He is currently working as a postdoctoral researcher in Network Laboratory (NETLAB) at SNU. His research interests include wireless IoT systems for urban marketplaces and Smart Grid, network and communication protocols for low power wireless systems.

Hongchan Kim received the B.S degree in Electrical and Computer Engineering from Seoul National University (SNU) in 2015. He is currently studying for a master's degree in Network Laboratory (NETLAB) at SNU. His research interests include designing routing protocols for low-power networks and constructing mobile IoT systems.

Jeongyeup Paek received his B.S. degree from Seoul National University 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. He worked at Deutsche Telekom Inc. R&D Labs USA as a research intern in 2010, then joined Cisco Systems Inc. in 2011 where he was a Technical Leader in the Internet of Things Group, Smart Grid business unit. He is currently an assistant professor at the School of Computer Science and Engineering, Chung-Ang University, Korea.



Saewoong Bahk is a professor at Seoul National University (SNU). He served as Director for the Institute of New Media and Communications during 2009-2011. Prior to joining SNU, he was with AT&T Bell Laboratories as a member of technical staff from 1991 to 1994 where he had worked on network management. He received KICS Haedong Scholar Award in 2012. He has been leading many industrial projects on 3G/4G/5G and IoT connectivity supported by Samsung Electronics, LG Electronics, SK Telecom, etc and published more than 200 technical papers and holds 72 patents. He was TPC Chair for IEEE VTC-Spring 2014 and general chair of JCCI 2015. He is co-EIC of IEEE/KICS Journal of Communications and Networks (JCN), and was on the editorial board for Computer Networks Journal (COMNET) and IEEE Transactions on Wireless Communications (TWireless). He is an IEEE senior member and a member of Whos Who Professional in Science and Engineering. He received B.S. and M.S. degrees in Electrical Engineering from SNU in 1984 and 1986, respectively, and the Ph.D. degree from the University of Pennsylvania in 1991.