IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 10, 2020, accepted September 28, 2020, date of publication October 6, 2020, date of current version October 16, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3028771

NG-RPL for Efficient P2P Routing in Low-Power Multihop Wireless Networks

YONGJUN KIM AND JEONGYEUP PAEK, (Senior Member, IEEE)
Department of Computer Science and Engineering, Chung-Ang University, Seoul 06947, South Korea

Corresponding author: Jeongyeup Paek (jpaek@cau.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (MSIT) under Grant 2020R1F1A1051282, and in part by the Chung-Ang University Graduate Research Scholarship in 2019.

ABSTRACT RPL, the standard IPv6 routing protocol for low-power and lossy networks in the emerging Internet of things (IoT), is designed mainly for efficient many-to-one data collection scenarios where the majority of the traffic flows from embedded devices to a gateway. Although RPL does support root-to-node downwards routing and peer-to-peer (P2P) communication, its P2P performance is inefficient and unsatisfactory due to excessively high churn and bottlenecks in the P2P path. However, P2P routing is important for machine-to-machine communication where nodes in IoT applications communicate with one another and control devices beyond simply collecting data. In this work, we propose a neighbor-graph-based RPL (*NG-RPL*) that significantly improves P2P routing performance. By including additional routing information when a packet passes through the root node for the first time in a P2P communication, *NG-RPL* finds efficient P2P routes opportunistically, when available, without significant overhead. We implement *NG-RPL* in Contiki-NG and evaluate its performance through extensive Cooja simulations. Results show that *NG-RPL* reduces routing churn, which improves packet reception ratio, round-trip time, and energy usage of P2P communication compared to standard RPL.

INDEX TERMS Low-power and lossy network (LLN), peer-to-peer (P2P), routing protocol, RPL, IPv6, machine-to-machine (M2M) communication, Internet of Things (IoT).

I. INTRODUCTION

With the development of embedded devices and low-power wireless networking technology, the Internet of things (IoT) is coming close to reality. In IoT, numerous sensors and devices are connected to the Internet to provide useful information and convenient services. The applications of IoT are endless. For example, a smart home may connect surveillance cameras, TV, HVAC, and lighting system to work autonomously depending on the environment dynamics and user context, and also to allow remote control access to the users. In a smart factory, status of the facilities may be monitored periodically to predict failures in advance, or to detect a leakage of toxic gas and trigger an alarm.

RPL is the IETF standard IPv6 routing protocol for low-power and lossy networks (LLNs) in IoT [1], [2], and is designed considering the resource constraints of embedded devices [3]–[6]. It enables standard IPv6 networking in low-power wireless multihop networks by forming a multihop routing tree rooted at a single LLN border router (LBR, a.k.a. root/sink/gateway), and each node finds the best upward path to reach the LBR according to a given criteria. Therefore, it is optimized for many-to-one data collection scenarios. However, in numerous IoT application use cases, it is also necessary to support downwards routing and P2P communication among nodes in the network. For example, a smart-home user may wish to turn on their air conditioners from outside, or a gas detection sensor may need to send its signal immediately to an alarming device.

RPL does support downwards routing to each node from an LBR, and also peer-to-peer (P2P) communication between two embedded devices. However, downward routes are constructed as a reverse of upward, and RPL does not have a separate mechanism for P2P routing. Instead, it is achieved naturally and transparently through the upward and downward routing as a result of basic IPv6 address-based forwarding. Due to this reason, its P2P performance is inefficient and unsatisfactory with excessively high churns and bottlenecks in the P2P paths.

To address this problem, this article proposes *neighbor graph based RPL* (*NG-RPL*) which supports more efficient P2P routing within RPL opportunistically. In *NG-RPL*, each

The associate editor coordinating the review of this manuscript and approving it for publication was Thanh Ngoc Dinh.

node piggybacks its neighbor information in addition to parent information in a regular RPL DAO message sent to the root, and the root constructs and maintains a network graph based on this. Then, whenever P2P traffic passes through the root node, the root calculates an optimal P2P path based on the neighbor graph and informs the corresponding source and destination nodes if a better path is available. This notification can be piggybacked into the data traffic if frame size permits, or can be sent separately in a RPL DAO-ACK message. Subsequently, the source and destination nodes use this optimal P2P path to send and receive data via source routing. The intermediate nodes located within this P2P path does not need to be aware of this path; a regular IPv6 forwarding of source-routed datagram will suffice.

NG-RPL uses shorter paths for P2P traffic, which not only reduces end-to-end latency and number of transmissions (and thus energy), but also improves packet transmission reliability by reducing congestion in the network. *NG-RPL* is backward-compatible in the sense that initial P2P communication will always work in the same way as the standard RPL, and a shorter P2P path will be used if and only if the root can calculate one based on the neighbor graph information it has gathered. If no such path exists (e.g. due to insufficient neighbor graph information, or too many non-*NG-RPL* nodes), *NG-RPL* will operate just like the standard RPL. Forwarding of source-routed messages is no different from standard IPv6. The overhead of *NG-RPL* is in sending additional neighbor information, but this is not significant since it utilizes the extra space within regular DAO messages.

Furthermore, since *NG-RPL* root knows the whole neighbor graph information in addition to parent-child relationships in the DODAG topology, it has a pleasant side-effect of having sufficient information to load balance the routing tree. Load imbalance is an important and challenging problem in RPL, and has been studied in several prior work [7]–[11]. Based on centralized neighbor graph information, *NG-RPL* balances the routing tree by recommending each node to select an alternative parent, if necessary, through DAO-ACK messages.

We implement *NG-RPL* on Telosb [12] platform using Contiki-NG [13], and evaluate it through extensive simulations in various scenarios and topology using the Cooja simulator [14]. We show that *NG-RPL* outperforms the standard RPL in terms of P2P path length (route churn), packet delivery reliability (PRR), latency, and the number of transmissions (energy). The proposed method aims to improve performance not only in P2P scenarios, but also for data collection and downwards routing (or mixed) scenarios.

The remainder of this article is structured as follows. Section II provides a background and motivation for *NG-RPL*. The proposed protocol, *NG-RPL*, is presented in Section III, and Section IV evaluates the performance of *NG-RPL* and compares it against the standard RPL. Section V provides a brief related work on RPL and P2P routing in LLN. Finally, Section VI concludes this article.

II. BACKGROUND AND MOTIVATION

This section first provides a brief background on RPL, and how downwards routing and P2P communication is performed in RPL. We also briefly introduce how source-routing is done in standard IPv6, and how this is used in RPL for downwards and P2P communication. We then motivate our work by discussing the problem in RPL's P2P routing that this article is tackling.

A. RPL BASICS

RPL is a distance-vector routing protocol that constructs a multihop routing tree rooted at a single LBR (a.k.a. root) by forming a *destination-oriented directed acyclic graph* (DODAGs) between nodes [1], [2]. When a new node joins a RPL network, it selects a parent node (default route) based on the DODAG information that it receives from its neighbors through *DAG information object* (DIO) messages [15]. An *objective function* (OF) [16]–[18] defines the metric and criteria that a node uses to select a parent within an RPL instance. Once a parent node is selected, the node sends *destination advertisement object* (DAO) messages to the root of DODAG to notify the next hop information. The node will also start transmitting DIO messages to advertise itself and its position in the DODAG so that other nodes can join or update their information.

Both the DIO and DAO messages will be sent periodically[1] to adapt to network dynamics, and DAO messages allow the root of the DODAG to have a complete up-to-date view of the parent-child route information of the network. Whether an intermediate node in the path from a node to the root updates its routing table based on a DAO message depends on the downwards routing *mode* of the RPL instance.

B. DOWNWARDS ROUTING IN RPL

RPL provides two modes of support for downwards routing: *storing* & *non-storing* [1], [20]. *Storing mode* is analogous to how traditional IP routers work; each node stores the next hop information towards the nodes in its subtree. It has the advantage of smaller packet overhead and intuitive IP forwarding behavior at the cost of requiring significant memory for the routing table in resource constrained embedded devices. A node closer to the root requires *O(N)* routing table entries in the worst-case corresponding to the total size of the network.

In contrast, nodes in *non-storing mode* do not store route information other than its parent's, and only the root maintains the whole topology information. When a packet is routed downwards, *source-routing* method is used starting from the root and root node only. Despite the increased and variable packet overhead for IPv6 header options due to source-routing (which adds the whole path information in the header), *non-storing mode* is more popular in LLNs [21] because of its advantage of requiring constant *O(1)*

[1]The actual timing of periodic transmission is implementation dependent, where DIO transmission is usually governed by the Trickle timer algorithm [1], [19], and DAO may be transmitted based on either a Trickle or a constant timer interval.

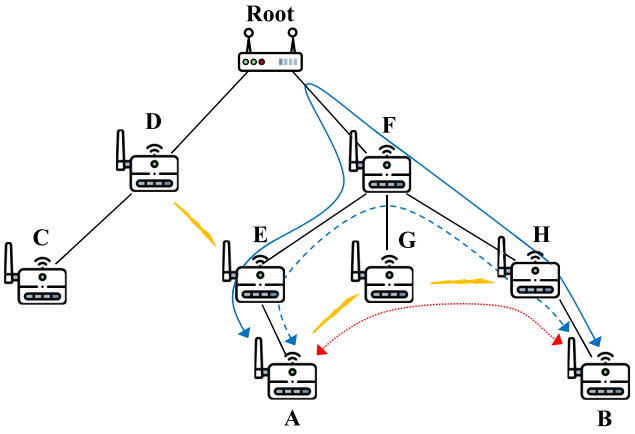


FIGURE 1. Nodes *A*, *G*, and *H* can communicate with one another. The blue lines show the RPL's P2P communication path. The solid and dashed lines correspond to operating in non-storing and storing modes, respectively. The red dotted line shows the expected P2P communication path of *NG-RPL*.

memory in embedded nodes. For this reason, we focus on the *non-storing mode* RPL in our design.

C. PEER-TO-PEER ROUTING IN RPL

RPL does not have a separate mechanism for peer-to-peer routing. Instead, it is achieved naturally and transparently through the upward and downward routing as a result of basic IPv6 address-based forwarding. A packet sent to another peer will naturally move upward first towards the root, and then move downward again when it encounters a node that has an entry corresponding to the destination address present in its routing table. Thus, the P2P communication performance depends on the operational mode described earlier. In non-storing mode, that node must be the root node, whereas in storing mode, the node can be any common ancestor node of the two peers within the routing tree. For example, Figure1 illustrates an example scenario in which nodes *A* and *B* communicate in a P2P fashion. A packet sent from node *A* will first traverse up the tree, and then down to node *B*, where the turning point is node *F* in storing mode (blue dashed arrow, path *A*–*E*–*F*–*H*–*B*) and the root node in non-storing mode (solid blue arrow, path *A*–*E*–*F*–*Root*–*F*–*H*–*B*).

D. THE PROBLEM

The problem is apparent from the preceding example (fig1); RPL's P2P communication is inefficient due to path length longer than necessary. From *A* to *B*, storing mode requires 4 hops (blue dashed arrow, path *A*–*E*–*F*–*H*–*B*) and non-storing mode requires 6 hops (solid blue arrow, path *A*–*E*–*F*–*Root*–*F*–*H*–*B*), despite a better path with 3 hops (dotted red arrow, path *A*–*G*–*H*–*B*) exists. This is because RPL does not provide a separate mechanism for efficient P2P route construction; it simply recycles the upward and downward routing paths. Therefore, resulting P2P path is longer than necessary, and this is called the routing *churn* [22]. High routing churn not only increases end-to-end latency, but also the number of link transmissions and thus energy usage. Furthermore, increased number of transmission will intensify contention in the wireless channel, and may cause network

Next Header		Hdr Ext Len	Routing Type	Segments Left
CmprI	CmprE	Pad	Reserved	
Addresses[1..n]				

FIGURE 2. Source-routing header format.

congestion especially at the turning point nodes of a P2P path which could become a bottleneck. This adds a heavy burden to nodes closer to the root, and excessive workload to the root especially for RPL in non-storing mode. Therefore, inefficient P2P communication in non-storing mode RPL must be resolved.

E. IPv6 SOURCE ROUTING

Source-routing is a method where the source (sender) can specify a complete routing path through which the packet passes through the network. *Source-routing header* (SRH) follows the format of IPv6 *type 0 routing header* (RH0) [23] as shown in fig2. Obviously, source routing can be initiated only by a node with complete path information, such as the RPL root in non-storing mode. RPL inevitably uses source-routing in non-storing mode (since LLN nodes do not have route information other than its parent), where the root is responsible for creating and attaching SRH when downward or P2P routing is needed.

For example, consider the solid blue line in fig1 where node *A* and *B* are communicating with each other. When a packet sent from *A* reaches the root, the root attaches SRH and forwards the packet to node *F*. At this point, the SRH has a *Segment Left* value of 2, and the addresses of nodes *H* and *B* in the *Addresses* field in fig2. Subsequently, it sets the next destination address of the packet to node *H* and puts the existing destination node *F* back into *Addresses*. It sends the packet with *Segment Left* 1 and *Addresses* (*F*, *B*) to Node *H*, and *H* will act similarly. When the packet reaches node *B*, it confirms that *Segment Left* is 0 and destination address is *B*, and then accepts the packet. Reverse direction (from node *B* to *A*) will also work in a similar way.

III. NG-RPL DESIGN

The key idea of *NG-RPL* is that the root node gathers neighbor information of all nodes via regular RPL DAO messages, and constructs a network graph based on this information. Then, in the event of P2P communication between nodes, the root can identify more efficient P2P paths beyond the constraints of using the existing upward and downward paths with only the parent information. Shortest P2P path is calculated opportunistically using the Floyd–Warshall algorithm [24] when P2P traffic passes through the root. This is possible because the LBR where the RPL root resides has more memory and processing power than the embedded LLN nodes. This section describes the design details of *NG-RPL*.

A. DAO MESSAGE WITH NEIGHBOR INFORMATION

The root of a RPL instance creates and manages a network graph that identifies the link connectivity of all nodes in its DODAG. For this purpose, the LLN nodes participating in

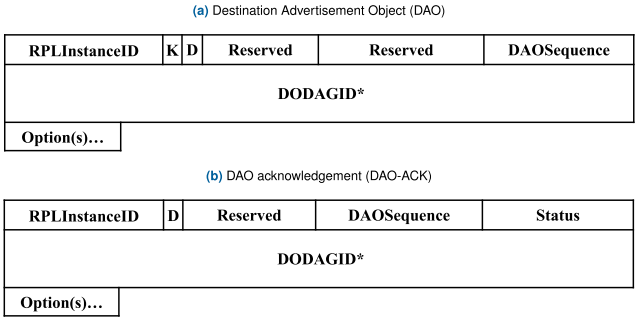


FIGURE 3. RPL DAO and DAO-ACK message format within ICMPv6 packet.

the RPL instance piggybacks neighboring node information into regular RPL DAO messages that it sends to the root periodically. In our implementation, neighbor information is inserted as an option in the *Option(s)* field of a DAO message (fig 3a), which includes the number of neighboring nodes, node addresses, and their respective link states. Node addresses are compressed using the standard 6LoWPAN prefix compression method [25], [26], and link states can be any metric that the RPL instance uses in its OF [16]–[18] to select parents. In our implementation, each node's IPv6 address is compressed down to 2-byte IEEE 802.15.4 short address, and we have used expected transmission count (ETX) [27] as the link metric,[2] thus using 4-bytes per neighbor entry. 6LoWPAN compression of IPv6 addresses allows the additional overhead in DAO messages to be manageable, and 6LoWPAN fragmentation [25] allows large number of neighbor information to be sent without any extra mechanism.

B. SHORTEST P2P PATH CALCULATION

When a P2P packet arrives at the root node; that is, when the root receives a packet from a node in its DODAG and identifies (by referring to its routing table) that it is destined to another node in its DODAG, the root will calculate and search for a shorter direct P2P path between the source and destination of the packet. We use the Floyd–Warshall algorithm [24] to calculate this shortest path. The reason for using this algorithm (instead of another shortest-path algorithm such as Dijkstra's or Bellman–Ford) is that one calculation can identify the shortest path for all node pairs, and thus we can reduce the frequency of path calculation. Instead of re-calculating the path(s) for every P2P packet (or source-destination pair) it receives, it can be done only when there is new or updated information in a DAO message. Path calculation can also be 'rate-limited' with a time interval and a threshold in link state changes so as to reduce the computation burden on the root node.

C. PEER-TO-PEER SOURCE ROUTING HEADER (PSRH)

NG-RPL introduces a new IPv6 header option, the *P2P source-routing header* (PSRH), which is similar in format to the standard IPv6 source-routing header (SRH). Other than the source and destination, any intermediate node receiving a packet with PSRH recognizes PSRH as SRH and performs the same action as done for SRH. The root is responsible for attaching a PSRH, just like the SRH in non-storing mode of RPL, and adds extra P2P path information between the source and destination. In other words, PSRH includes the routing path information that can go downwards from the root node to the destination (same as in the existing SRH), followed by the P2P path information that allows sending messages directly from the destination node to the source node.

As an example, consider fig1. When A sends a P2P packet to B for the first time, the packet will first go up the tree to the root via E and F just like the standard RPL in non-storing mode. Then, the root node generates and attaches a PSRH if it can calculate a shorter path between B and A. In this example, there is a shorter 3-hop path between node A and B through G and H (dotted red line). Thus, the PSRH will have a path from the root to B, and also a path from B to A, which would be;

$$\{Root \rightarrow F \rightarrow H \rightarrow B \rightarrow H \rightarrow G \rightarrow A\}.$$

This packet will naturally traverse down the tree from the root to B based on standard IPv6 source-routing, just like the standard RPL, but will be received by B with three remaining addresses unused. Upon receiving this message, node B now knows a shorter path to A, and will record this in its routing cache entry for A. Subsequently, when node B sends a packet to node A (e.g. an end-to-end ACK or a response message back to A), this shorter path can be used using a SRH or PSRH. Furthermore, when node A receives a packet from B with SRH or PSRH, it can extract the P2P path information in the reverse direction and use it for shorter and more efficient P2P communication. Note that if there is no PSRH, nothing changes from the standard RPL and IPv6.

In our implementation, PSRH has the same format as SRH (fig2), but adds three extra routing information in the existing *Reserved* and *Addresses* fields of SRH. In addition to the original *Segments Left* value and *Addresses* in the source route from the root to the destination, (1) *Addresses* field is extended with the path from the destination to the source. PSRH also adds, (2) the number of nodes in the P2P path from destination to source, and (3) the number of nodes in the whole *Addresses* field which includes the path from the root to the destination and then to the source. These information allow nodes to distinguish PSRH from SRH, and enable shorter and more efficient P2P route between the two end nodes. Finally, although we mentioned 'root/destination/source' in this description for ease of understanding, PSRH would work for any three node trios.

D. PARENT ASSIGNMENT FOR LOAD BALANCING

Since *NG-RPL* root has a complete neighbor graph of the network, it has sufficient information to recommend parents to the nodes. This centralized view allows *NG-RPL* to construct

[2]ETX considers asymmetric links via bi-directional link PRR, and can be measured through IPv6 neighbor discovery protocol.

a better routing tree than the distributed selections made independently by each RPL node. Of course, the definition of 'better' may depend on the application and scenario, and various metrics can be used for selecting a recommended parent such as the number of nodes in the subtree or amount of forwarding traffic. In our implementation, we chose 'load balancing' as our criteria where an alternate parent is recommended if it meets the following three conditions: (1) rank is less than or equal to the current parent's rank, (2) link ETX is less than or equal to the current parent's ETX, and (3) subtree size is smaller than that of the current parent by more than my subtree size plus 2. First two are same as what RPL with OF0 would do in a distributed way, and we need them to avoid contradiction; We're adding (3) for load balancing.

To notify each node of this recommendation, *NG-RPL* uses DAO-ACK messages. A RPL root receives DAO messages periodically from all nodes participating in its DODAG, and updates its routing table based on this information. Then, it sends a DAO-ACK message back to the originator indicating that it has received the DAO successfully. In our implementation, we have added the recommended parent information as a new option in the *Option(s)* field of the existing DAO-ACK format (fig3b) to maintain backward compatibility. If a standard RPL (non-*NG-RPL*) node receives this option, they will simply ignore it. If an *NG-RPL* node receives a recommended parent, it is reflected in the parent selection process.

E. DISCUSSION-RPL STORING-MODE

Our description of *NG-RPL* design so far was implicitly based on the non-storing mode RPL. We have described the use of source-routing and SRH, how root receives neighbor information via DAO messages and calculates shortest P2P paths, and how P2P packets are forwarded within the network more efficiently. However, this was just to simplify the explanation and deliver our key ideas clearly. Although *NG-RPL* would benefit more in non-storing mode RPL since all P2P traffic passes through the root, there is nothing that stops *NG-RPL* from working with the storing mode. The only change is that all non-leaf nodes will perform the same actions as the root. If RPL is used in storing mode, this implies that the LLN nodes have sufficient memory to store $O(N)$ routing table. Intermediate nodes in RPL storing mode will receive DAO messages from nodes in its subtree, and will be able to store neighbor information and calculate P2P paths within its subtree, if they exist (if both the source and destination are in its subtree). Furthermore, source-routing and SRH are IPv6 features; it is required in non-storing RPL, but it works regardless of whether RPL is used in storing or non-storing mode. Thus, source-routing and PSRH will also work in storing-mode of RPL.

IV. EVALUATION

In this section, we evaluate the P2P communication performance of *NG-RPL*, and compare it against the standard RPL through extensive simulation study with varying number of nodes and two different locations of the root.

A. IMPLEMENTATION & SIMULATION SETUP

We implement *NG-RPL* on Contiki-NG (release v4.5) [13], an open-source cross-platform embedded operating system for IoT devices that includes IPv6, 6LoWPAN, RPL, and many other standard protocols for low-power wireless networking. Contiki-NG was released in 2017 as a fork of Contiki-OS [28]. RPL-lite, an implementation of RPL standard in Contiki-NG, is an evolution of ContikiRPL [29] in Contiki-OS where it has simplified and carefully redeveloped only the core and stable functions. By removing complex logic such as support for multiple RPL instances, multiple DODAGs, and storing mode, RPL-lite has a smaller code footprint and better performance than ContikiRPL. In our evaluation, RPL-lite is used as the standard RPL, and *NG-RPL* is implemented as an enhancement to RPL-lite and IPv6 in Contiki-NG. We have made our implementation publicly available as open source.[3]

For the simulations, we use the Cooja simulator [14] with COOJA mote and TmoteSky as the embedded platforms. Multi-Path Ray Tracer Medium (MRM) is used to model the lossy environment since it better reflects the real wireless channel by modeling reflection, refraction, diffraction, and fading [14]. Six different topologies are used where 25, 49, and 100 nodes are placed 200 meters apart in a square grid layout, and the root node is placed either at the center or the corner of the grid.

We simulated a P2P scenario in which each node (source) is assigned a random destination node, and sends one hundred P2P *request* packets at 5 second intervals. First packet of each source node was given a 0∼4999 ms jitter to avoid synchronizing effect between all pairs. Upon reception of a *request* packet, the destination node replies by transmitting a *response* P2P packet back to the source node. Using this setup, we compare the performance of RPL, *NG-RPL* without the load balancing (LB) feature, and the complete *NG-RPL* with LB.

The following metrics are used for our evaluation;

- *Number of traversed hops* on the P2P path between the source and destination nodes.
- *Round-trip time (RTT)* for the source node to receive the response packet after sending the request packet.
- *Packet reception ratio (PRR)* is the ratio of the number of response packets received by the source node to the number of request packets sent by the source node.
- *Number of transmissions* is the total number of packet transmissions for each simulation run, including link retransmissions.

B. EVALUATION RESULTS

NG-RPL has improved the overall P2P routing performance compared to the standard RPL. This can be seen from the following simulation results.

First of all, we look at the *average number of hops traversed by the P2P packets* from each source node to their destination, since this is the first key measure that we aimed

[3]https://github.com/StaySharp0/contiki-ng/tree/ng-rpl

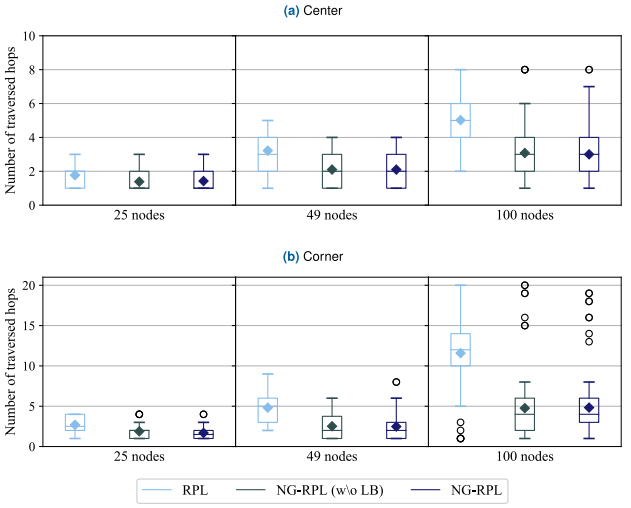


FIGURE 4. Average number of traversed hops with varying number of nodes and two different root positions.

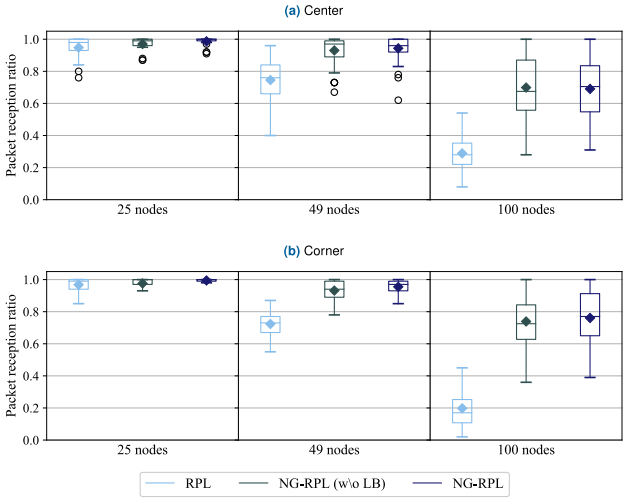


FIGURE 5. PRR with varying number of nodes and two different root positions.

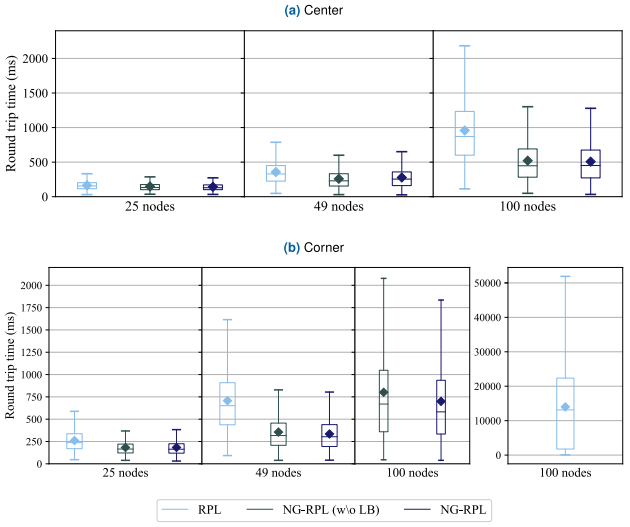


FIGURE 6. Average RTT of all P2P pairs with varying number of nodes and two different root positions. (Note the y-axis for RPL on 100-nodes, corner).

to reduce; the 'routing churn'. In other words, instead of recycling the upward and downward routes of RPL for P2P communication, our goal is to use *shorter and more direct P2P paths* for more efficient operation with reduced number of transmissions, reduced latency, and less contention at the bottlenecks resulting in improved reliability.

fig4 plots this result with varying number of nodes (25, 49, 100) and two different root positions ('center' or 'corner' of the grid topology network). In the figure, the difference may look small for 25-node network with root at the center since the network is small and shallow to begin with; P2P paths are ∼2 hops on average, and there is not much room for improvement. However, if we move to the 100-node network, the improvement is apparent and significant. When the root is at the center, average of ∼5 hops reduces to ∼3 hops, and when the root is at the corner (deeper network), average of 11.58 hops reduces to 4.32 hops, a 58.29% reduction. Significant reduction in P2P path length will not only reduce latency, but also the number of link transmissions which will reduce energy usage on embedded low-power devices. It will also reduce contention on the wireless channel, and there will no longer be serious bottlenecks since P2P paths are direct without turning points, which will improve reliability.

fig5 plots the PRR of all end-to-end P2P transactions, again with varying number of nodes and two different root positions. *NG-RPL* (with or without LB) clearly improves PRR significantly compared to RPL, especially in a larger and deeper topology. Specifically, PRR increases by only 2.64% on the 25-nodes (96.8%→99.44%) because the PRR of RPL was good and there is not much room for improvement. However, on the 100-node corner topology, PRR increase by a huge 56.29%, almost 4x improvement from unusable (19.80%) to reasonable (76.09%). This owes to the shorter P2P paths without bottlenecks which reduces the total number of link transmissions and contention in the wireless medium. The load balancing feature does provide a little extra improvement of 2.2% (73.89%→76.09%). However, the real advantage and benefit from load balancing would be seen for upward data collection scenarios rather than P2P scenarios since we have balanced the tree in terms of parent-rank-subtree sizes, not in terms of P2P path or traffic load.

fig6 plots the RTT results for RPL and *NG-RPL*, and it shows that *NG-RPL* achieves significantly better RTT than RPL. For example, on 100-node centered-root topology, RTT reduces from 958.25 ms to 507.08 ms, a 47.08% decrease. For the 100-node corner topology, the reduction is much more significant due to excessive congestion at the bottleneck nodes. This is an obvious and expected outcome considering the reduction in the number of hops traversed by the P2P packets (fig4) thanks to shorter direct P2P paths.

The load balancing feature allows *NG-RPL* to construct a more balanced routing tree structure than the standard RPL. This can be seen visually in fig7 which depicts the routing

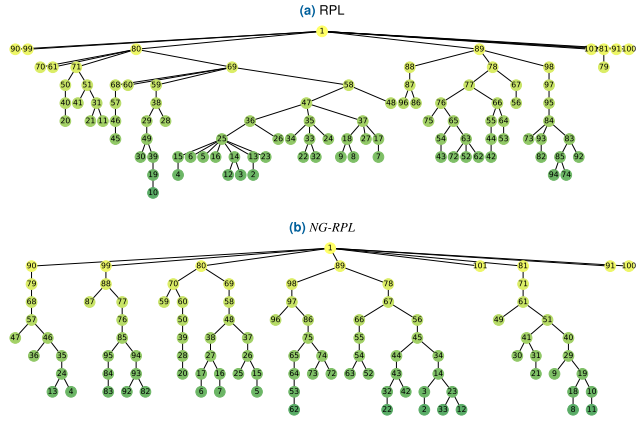


FIGURE 7. Logical routing topology constructed by RPL and *NG-RPL* for 100-node grid topology network where the root is at a corner.

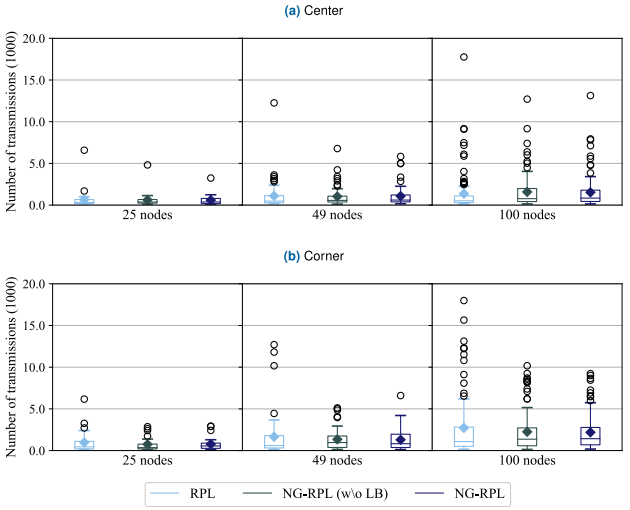


FIGURE 8. Total number of link transmissions during P2P data transfer, with varying number of nodes and two different root positions. Ideally, this should be significantly reduced for *NG-RPL* compared to RPL since it is proportional to the P2P path length in fig4. However, the reduction is small due to early drops (mainly due to queue overflow) in RPL case. It can be understood as proportional to the product of fig4 and fig5.

topology of RPL and *NG-RPL* for the 100-node network when the root is in the corner. RPL has an unbalanced topology with first-level subtrees of sizes 56, 35, and many 1∼2s. On the other hand, *NG-RPL* forms first-level subtrees with sizes 10, 12, 21, 33, 16, and a few 1s, reducing the maximum subtree size from 56 to 33. It also reduces the standard deviation of the all subtree sizes from 8.91 to 5.84, and average subtree sizes from 7.96 to 6.97 for all nodes (excluding the leaf node which has no subtree nor children). In terms of number of immediate children nodes, *NG-RPL* reduces the maximum from 7 to 2 (excluding the root), and average from 1.72 to 1.40 (excluding the leaf). These results confirm that *NG-RPL*'s parent recommendation is in action, and does improve the balance of the DODAG tree structure. This can lead to performance improvement especially for upward/downward traffic scenarios rather than P2P scenarios since *NG-RPL* balances the tree by adjusting the routing parent-child relationships and P2P traffic in *NG-RPL* uses direct path through neighbors.

To verify that the reduced number of average traversed hops in fig4 lead to reduction in total number of link transmissions, we plot fig8. At a first glance, surprisingly, the numbers did not reduce much, only by 19.65% for the 100-node corner case. This was less than expected since the number of hops traversed by P2P traffic reduced by 58.29% (fig4). After careful investigation, the reason turned out to be that the P2P packets in RPL had longer end-to-end path than *NG-RPL*, but the packets were often lost early in the path and did not add to transmission counts for latter part of the path. For example, if RPL and *NG-RPL* each had 11-hop and 5-hop P2P path respectively, but if RPL's P2P packet got lost after 5-hops where as *NG-RPL*'s packet made it all the way through, then the total number of transmissions will look similar. In our 100-node corner-root RPL simulations, large number of packets were lost at nodes 80 and 69 due to queue overflow (symptom of congestion) which had subtree sizes of 55 and 44, respectively. After knowing this fact, fig8 makes more sense; it can be understood as proportional to the product of fig4 and fig5. Thus, we can say that *NG-RPL* does reduce the number of transmissions for a P2P packet that is successfully delivered, and therefore it is more efficient in terms of energy usage per P2P transactions.

Finally, regarding the extra overhead added by *NG-RPL*, the most significant would be the increase in DAO message size due to added neighbor information. It has increased from 46 bytes to 77.32 bytes, an 68% increase on average. However, this still fits within the 127 byte MTU limit of IEEE 802.15.4, and the number of DAO messages is usually much smaller compared to the number of data messages which means that the cost can be amortized by the amount of data traffic. DAO-ACK size has also increased due to parent recommendation for load balancing, but only slightly from 4 bytes to 4.16 bytes, on average. This is because, once the routing tree is (roughly) balanced, parent recommendation is no longer needed. Furthermore, when we calculate the average size of SRH/PSRH, it actually decreased from 48.56 bytes to 31.39 bytes. This is because, except for the first P2P packet that passes through the root, following subsequent P2P packets can use a much shorter path with smaller SRH/PSRH.

V. RELATED WORK

Prior to the standardization of RPL in 2012, several reactive routing protocols based on ad-hoc on-demand distance vector (AODV) have been proposed to provide *any-to-any* routing support in LLNs. Examples of these protocols include LoWPAN-AODV [30], NST-AODV [31], LOAD-ng [32], and TinyAODV [33]. These protocols have simplified the original AODV, and optimized it to account for the resource constraints, lossy wireless links, and network dynamics of LLNs and their embedded devices. However, the flooding nature of route request/response messages still had huge network overhead, and this was an inevitable cost to allow generic on-demand any-to-any routing. If the traffic pattern

of the application can be known a priori (e.g. many-to-one data collection), then a different approach such as RPL can be more efficient. Of course, RPL had to sacrifice the P2P performance to achieve this efficiency.

In RPL, Bacceli *et al.* [34] proposed P2P-RPL to improve the P2P performance of RPL, which uses route messages to create optimized P2P paths. The performance of P2P-RPL was validated in testbed experiments, and had been submitted as an experimental document in RFC6997 [26]. Later, extensive simulation studies were conducted for P2P-RPL using NS-3 simulators [35]. However, P2P-RPL has the problem of low response rate for route requests, and its broadcasting of route messages can easily congest the network. In addition, it is not backward compatible in the sense that P2P communication would not work when the network contains nodes that do not understand P2P-RPL messages or ignore them in their policy.

Zhao *et al.* proposed ER-RPL [36], which reduces such route message overhead of P2P-RPL and improved reliability by adding geographical location to DIO messages. In contrast to P2P-RPL which requires all nodes to search for paths, ER-RPL explores only a subset of nodes based on geographic location information. Thus, ER-RPL can find a near optimal routing path in terms of energy-saving and reliability. However, having geographic location information in resource constrained LLN devices is a costly requirement if not impractical, and it focuses only on stat networks.

Farooq *et al.* proposed ERPL [37] which has a mechanism to forward packet directly without upward routing if the destination node is a direct neighbor to the source node or a direct neighbor to the parent nodes of the source node. ERPL achieves an effect similar to multicast DAO (MDAO) messages through DIO, without the need to transmit additional MDAO messages. However, in order to benefit from this feature of ERPL in P2P communication, the destination node must be a direct neighbor node among the parent nodes of the source node. For example, The packets generated from node A in fig1 cannot benefit from ERPL if nodes B, H, or C are the destinations.

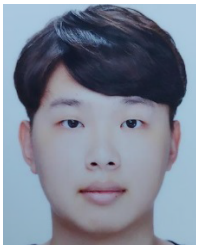
VI. CONCLUSION

IoT applications in which low-power and low-cost devices communicate with one another are becoming essential elements of our lives. RPL, the IPv6 standard routing protocol for LLN, is one of the key enablers for data collection in IoT, but with less than satisfactory P2P performance. To address this shortcoming, this work proposed *NG-RPL* that improves the performance of P2P communication in RPL by constructing a shorter P2P routing path opportunistically, when available, without significant overhead. We have implemented *NG-RPL* in Contiki-NG embedded operating system for IoT, and demonstrated an average performance improvement of 20.55% in PRR, 38.09% in RTT, and 12.50% in energy consumption compared with RPL on the 49-node topology.

REFERENCES

[1] T. Winter, *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*, document RFC 6550, Mar. 2012.
[2] H.-S. Kim, J. Ko, D. E. Culler, and J. Paek, "Challenging the IPv6 routing protocol for low-power and lossy networks (RPL): A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2502–2525, 3rd Quart., 2017.
[3] T. W. Brandt, *Routing Requirements for Urban Low-Power and Lossy Networks*, document RFC 5548, May 2009.
[4] E. K. Pister, E. P. Thubert, S. Dwars, and T. Phinney, *Industrial Routing Requirements in Low-Power and Lossy Networks*, document RFC 5673, Oct. 2009.
[5] A. Brandt, J. Buron, and G. Porcu, *Home Automation Routing Requirements in Low-Power and Lossy Networks*, document RFC 5826, Apr. 2010.
[6] E. J. Martocci, P. D. Mil, N. Riou, and W. Vermeylen, *Building Automation Routing Requirements in Low-Power and Lossy Networks*, document RFC 5867, Jun. 2010.
[7] T. B. Oliveira, P. H. Gomes, D. G. Gomes, and B. Krishnamachari, "ALABAMO: A LoAd alancing model for RPL," in *Proc. SBRC*, Jun. 2016, pp. 1–5.
[8] M. Ha, K. Kwon, D. Kim, and P.-Y. Kong, "Dynamic and distributed load balancing scheme in multi-gateway based 6LoWPAN," in *Proc. IEEE Int. Conf. Internet Things(iThings)*, Sep. 2014, pp. 87–94.
[9] X. Liu, J. Guo, G. Bhatti, P. Orlik, and K. Parsons, "Load balanced routing for low power and lossy networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2013, pp. 2238–2243.
[10] H.-S. Kim, H. Kim, J. Paek, and S. Bahk, "Load balancing under heavy traffic in RPL routing protocol for low power and lossy networks," *IEEE Trans. Mobile Comput.*, vol. 16, no. 4, pp. 964–979, Apr. 2017.
[11] H.-S. Kim, J. Paek, D. E. Culler, and S. Bahk, "PC-RPL: Joint control of routing topology and transmission power in real low-power and lossy networks," *ACM Trans. Sensor Netw.*, vol. 16, no. 2, pp. 1–32, Apr. 2020.
[12] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *Proc. 4th Int. Symp. Inf. Process. Sensor Netw.*, 2005, pp. 364–369.
[13] (2017). *Contiki-NG*. [Online]. Available: https://github.com/contiki-ng/contiki-ng
[14] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with COOJA," in *Proc. 31st IEEE Conf. Local Comput. Netw.*, Nov. 2006, pp. 641–648.
[15] J. Vasseur, M. Kim, K. Pister, N. Dejean, and D. Barthel, *Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks*, document RFC 6551, Mar. 2012.
[16] P. Thubert, *Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL)*, document RFC 6552, Mar. 2012.
[17] O. Gnawali and P. Levis, "The ETX objective function for RPL," IETF Internet Draft, Internet Engineering Task Force, Fremont, CA, USA, Internet Draft draft-gnawali-roll-etxof-01, May 2010.
[18] O. Gnawali and P. Levis, *The Minimum Rank with Hysteresis Objective Function*, document RFC 6719, Sep. 2012.
[19] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proc. USENIX/ACM NSDI*, Mar. 2004, pp. 1–5.
[20] 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 Trans. Sensor Netw.*, vol. 11, no. 2, p. 39, 2015.
[21] Cisco Systems. *Connected Grid Networks for Smart Grid—Field Area Network/CG-Mesh*. Accessed: Oct. 6, 2020. [Online]. Available: http://www.cisco.com/web/strategy/energy/field_area_network.html
[22] O. Gnawali, R. Fonseca, K. Jamieson, M. Kazandjieva, D. Moss, and P. Levis, "CTP: An efficient, robust, and reliable collection tree protocol for wireless sensor networks," *ACM Trans. Sensor Netw.*, vol. 10, no. 1, pp. 1–49, Nov. 2013.
[23] S. Deering, *Internet Protocol, Version 6 (Ipv6) Specification*, document RFC 2460, 1998.
[24] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, Jun. 1962.
[25] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, *Transmission of IPv6 packets over IEEE 802.15. 4 networks*, document RFC 4944, 2007.
[26] M. Goyal, E. Baccelli, M. Philipp, A. Brandt, and J. Martocci, *Reactive Discovery of Point-to-Point Routes in Low Power and Lossy Networks*, document RFC 6997, Aug. 2013.

[27] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *Proc. 9th Annu. Int. Conf. Mobile Comput. Netw.*, 2003, pp. 134–146.
[28] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki–a lightweight and flexible operating system for tiny networked sensors," in *Proc. 29th Annu. IEEE Int. Conf. Local Comput. Netw.*, 2004, pp. 455–462.
[29] N. Tsiftes, J. Eriksson, and A. Dunkels, "Low-power wireless IPv6 routing with ContikiRPL," in *Proc. 9th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, Jan. 2010, pp. 406–407.
[30] G. Montenegro and N. Kushalnagar, "AODV for IEEE 802.15. 4 networks," *Proc. Internet Draft*, 2005, pp. 1–6.
[31] C. Gomez, P. Salvatella, O. Alonso, and J. Paradells, "Adapting AODV for IEEE 802.15.4 mesh sensor networks: Theoretical discussion and performance evaluation in a real environment," in *Proc. Int. Symp. World Wireless, Mobile Multimedia Networks (WoWMoM)*, 2006, p. 6.
[32] T. H. Clausen and A. C. D. Verdière, "The LLN On-demand Ad hoc Distance-vector Routing Protocol - Next Generation (LOADng)," INRIA, Paris, France, Res. Rep. RR-7692, Jul. 2011.
[33] (2007). *TinyAODV implementation*. [Online]. Available: https://github.com/tinyos/tinyos-main
[34] E. Baccelli, M. Philipp, and M. Goyal, "The P2P-RPL routing protocol for IPv6 sensor networks: Testbed experiments," in *IEEE Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM)*, 2011, pp. 1–6.
[35] M. Zhao, A. Kumar, P. H. Joo Chong, and R. Lu, "A comprehensive study of RPL and P2P-RPL routing protocols: Implementation, challenges and opportunities," *Peer Peer Netw. Appl.*, vol. 10, no. 5, pp. 1232–1256, Sep. 2017.
[36] M. Zhao, I. W.-H. Ho, and P. H. J. Chong, "An energy-efficient region-based RPL routing protocol for low-power and lossy networks," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1319–1333, Dec. 2016.
[37] M. O. Farooq and D. Pesch, "ERPL: An enhanced Peer-to-Peer routing mechanism for low-power and lossy networks," in *Proc. 11th IFIP Wireless Mobile Netw. Conf. (WMNC)*, Sep. 2018, pp. 1–8.



YONGJUN KIM received the B.S. degree from the School of Computer Science and Engineering, Chung-Ang University, Seoul, South Korea, in 2019, where he is currently pursuing the M.S. degree with the Department of Computer Science and Engineering. He is also a Research Assistant with the Networked Systems Laboratory led by Dr. Jeongyeup Paek, with research interests in wireless and time-sensitive networking.

JEONGYEUP PAEK (Senior Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, in 2003, and the M.S. degree in electrical engineering and the Ph.D. degree in computer science from the University of Southern California, in 2005 and 2010, respectively. He worked with the Research and Development Laboratories, Deutsche Telekom, Inc., USA, as a Research Intern, in 2010. He joined Cisco Systems, Inc., in 2011, where he was the Technical Leader with the Internet of Things Group, Connected Energy Networks Business Unit (formerly the Smart Grid BU). At Cisco Systems, Inc., he was one of the lead engineers for RPL and IEEE 802.15.4e/g Software in CG-mesh smart-grid AMI system. In 2014, he was with the Department of Computer and Information Communications Engineering, Hongik University, as an Assistant Professor. He has been an Associate Professor with the School of Computer Science and Engineering, Chung-Ang University, Seoul, South Korea, since 2015. He is a member of ACM.