On Adaptive Timers for Improved RPL Operation in Low-Power and Lossy Sensor Networks

J. Tripathi, J. C. de Oliveira
Electrical & Computer Engineering Department
Drexel University, Philadelphia, 19104, USA.
{jt369@,jau@ece.}drexel.edu

***Abstract*—In this paper, we propose two adaptive algorithms to control Destination Advertisement Option (DAO) packet emissions in RPL non-storing mode: a centralized, or controlled by the DAG root, and a distributed. We propose new mechanisms to derive the most appropriate value for the DelayDAO Timer, a timer that determines how long a node should wait before transmitting its destination advertisement packet to the DAG root or collection point. We also propose similar mechanism for RPL-storing mode by implementing multiple destination aggregation in a single DAO packet. We show that instead of using a fixed universal timer to control DAO emissions, as recommended in the IETF standard, the use of an adaptive timer at each node allows the network to quickly adjust to topological changes. This is aimed at reducing congestion and packet drops, specially near the DAG root, which also impacts memory and buffer requirements, as well as data delivery delay. This is crucial for networks such as Urban LLNs or Smart Grid AMI networks, whose nodes have very limited resources with regards to energy and buffer space. We show how the proposed mechanisms outperform the default method in RPL with regards to these metrics.**

***Index Terms*—RPL, LLN, DelayDAO Timer, Smart Grid AMI Networks.**

I. Introduction and Related Work

Urban Low-Power and Lossy Networks (U-LLN) often span a vast geographical region and consist of thousands of nodes. These networks are characterized by highly time variant nature of the links, with nodes having a mere few KB of flash memory, and large-scale deployments [1]. The RPL (Routing Protocol for Low-Power and Lossy Networks) has been standardized in the IETF ROLL (Routing Over Low-Power and Lossy Networks) Working Group as a suitable routing protocol for networks interconnecting constrained devices in LLNs [2]. RPL provides mechanisms for multipoint-to-point (MP2P), point-to-multipoint (P2MP) and point-to-point (P2P) traffic and is designed to meet the requirements in [1], [3], [4], [5]. It is an IPv6 distance vector routing protocol that builds a Destination Oriented Directed Acyclic Graph (DODAG or simply DAG).

RPL is envisioned as the routing protocol to interconnect smart objects in the Internet of Things (IoT), in smart grid AMI networks, etc. RPL has been designed with mainly two modes of operation: storing and non-storing, with the former implicating nodes' ability to store routing table information. Non-storing mode relaxes that requirement, and therefore is deemed more appropriate for large deployments of nodes with limited memory resources. We have however observed that, for very large networks, RPL non-storing mode operation actually requires large memory and buffer space. Proportional scaling of memory requirements with network size is disadvantageous for any routing protocol intended to be implemented in vast U-LLNs or Smart Grid AMI networks. Investigating the causes for such an non-intuitive behavior from a mode that was actually designed to cope with the large deployments of nodes with limited capacity, we determined the culprit: congestion caused by the Destination Advertisement Option (DAO) packets from every node to the DAG root during a global repair (see Section II), or after the DAG root triggers a network-wide address accumulation using a new DAO Trigger Sequence Number (DTSN). This congestion is lethal due to a number of reasons: Firstly, upon congestion, the buffer requirement will increase, leading to high memory requirement to store packets. Secondly, if high buffer space is not available, DAOs will be lost and the nodes will be forced to send duplicate DAO packets, which may worsen the situation. Finally, important and time-sensitive data or alert packets may be lost as well. Therefore, in order to control the congestion in large deployments of memory constrained nodes, it is essential to design a suitable approach for DAO message emission.

In this paper, we propose two adaptive algorithms to control DAO emissions in RPL non-storing mode (one centralized, or controlled by the DAG root, and one distributed). We show that instead of using a fixed universal timer to control DAO emissions, as recommended in the standard, making use of an adaptive timer at each node allows the network to adjust itself to account for topological changes, and adjust each timer in order to avoid congestion and packet drops, specially near the DAG root, which would also impact data delivery delay. Also, we also propose an aggregation method for DAO packets in storing mode. We propose a simple algorithm to determine the timer to generate, aggregate and forward destination addresses. To the best of our knowledge, this is the first work to reveal the DAO emission issue, and to shed light on how congestion may occur in a large scale data aggregation network, and further, how one may avoid it.

There have been a number of studies on RPL's performance in real networks. In [6], [7], and [8], we studied RPL's performance in small real deployments with respect to several metrics of interest, looked into local and global repair implementations and presented the same metrics for three different sized topologies from real deployments. In [9], Wang et al. presented a comparison between RPL and AODV for smart meter AMI networks using ETX as path metric. In this paper, we go beyond our previous performance analysis from [6], [7], and [8]. We explain the existing DAO emission issue and provide two algorithms capable of motivating it.

This paper is organized as follows: In Section II, a brief overview of RPL is provided. Section III discusses the motivations for the work, stemming from our analysis of the protocols' behavior in simulations of a real topology. Sections IV and V propose distributed and centralized (run at the DAG root) algorithms respectively to determine the time duration between receiving a global repair and issuing a DAO packet while no address aggregation is performed. Simulation details and results that show the benefits of the proposed algorithms are discussed in Section VI. Section VII discusses how aggregation for multiple addresses in a single DAO is possible, and proposes algorithm to take advantage of the same. Corresponding simulation results for RPL storing mode are presented in Section VIII. Finally, in Section IX we overview the conclusions reached and discuss insights on these mechanisms may be further improved and adapted for for use in RPL storing mode.

II. Brief Overview of RPL

As specified in [2], RPL is an IPv6 distance vector routing protocol that builds a DODAG (or simply, DAG) that minimizes the cost of reaching the sink from any node in the network as per the Objective Function (OF). The OF can be defined to minimize a particular metric, such as hop count or ETX (Expected Transmission count). In order to form the DAG, the root node starts broadcasting DAG Information Option (DIO) messages. A DIO message contains information such as the broadcasting node's rank (number of hops from the backbone network to the node), the OF, DAG-ID, etc. Any node that is connected to a non-RPL implementing node or a backbone network, can act as a root or LBR (LLN Border Router) and has a rank equal to 1. Once a node receives a DIO, it calculates its path cost based on the cost in the received DIO, and the cost of reaching the node from itself. A node then chooses its parent, amongst its neighbors, as the node that yields the lowest cost to reach the LBR. RPL defines a number of rules for parent selection based on the local quality of the links, the advertised OF, path cost, rank, etc. Propagation of DIOs from the LBR to the leaf nodes, as shown in Figure 1, creates a DAG rooted at the LBR. Parent - child relationships are illustrated as solid lines in the figure.

DIOs are emitted periodically from each node, triggered by a timer (trickle timer) whose duration increases exponentially (doubled after each time it is fired). The smallest possible interval between two DIOs is denoted by $Imin$, and the number of times $Imin$ can be doubled before maintaining a constant rate is denoted by $DIOIntervaldoubling$. On any event that causes a change in the DAG structure (parent node unreachable, new parent selection, new DODAG Sequence Number etc.), this timer is reset to the $Imin$ value given in the DIOs. When a node joins the network, it may wait to receive a DIO or it may alternatively multicast a solicitation message, called the DIS, to proactively solicit DAG information (DIO).

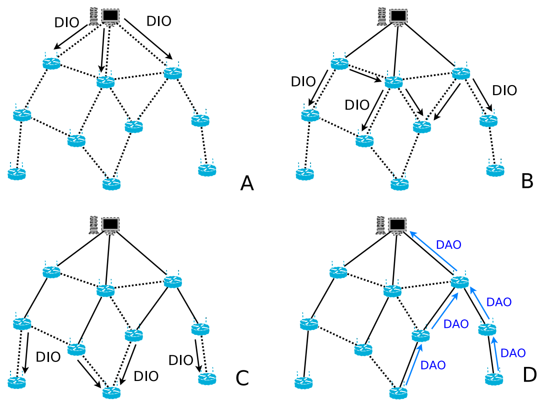


Fig. 1: Creation of DODAG and DAO propagation.

Upon link failure, a node may lose both its parents (preferred and backup) and siblings (neighbors with same ranks). In this case, any packet to be routed up the DAG has to be dropped. RPL proposes two mechanisms for DAG repair, known as $i)$ *global repair* and $ii)$ *local repair*. Global repair is adopted to refresh the DAG from major topological changes, where the LBR will periodically emit a DIO with a new sequence number (DODAG Sequence Number). When a node encounters a DIO with newer sequence number, it again starts the parent selection process as per updated link costs. Also, a local repair mechanism is adopted to fix unavailability of route to the DAG root from a node, by sending out a 'Poison' message to notify its children of the need for alternate parents. Then the node sends out a solicitation message, so that it may join another node as its parent upon receipt of a DIO from that node.

Nodes joining the DAG multicast their addresses and reachable prefixes to parents via Destination Advertisement Option (DAO) messages, which in turn are unicast further up the DAG to support 'down' traffic. Eventually all DAOs reach the LBR. The process of DAO propagation is illustrated in Figure 1-D.

RPL has two modes of operation, 'storing' and 'non-storing' mode. In storing mode, a node in the LLN is capable of storing routing tables and next hop information for all the nodes in its subtree. Whenever the node forwards a DAO message for a destination available via itself or any of its children nodes, it creates a corresponding routing entry for the particular destination. RPL non-storing mode is suitable for large networks with constrained nodes, such as in smart grid or city-wide LLN deployments, where storing routing tables for thousands of destinations is not feasible. In this mode, the DAG root stores routes to all destinations in the DAG. Contrary to storing mode, DAOs in non-storing mode are unicasted to the DAG root, and no intermediate node processes DAO packets. P2P packet routing only takes place via the parent-child links in the DAG. In non-storing mode, every packet needs to go to the DAG root, and the DAG root inserts source routing information on the packet's header to route the packet by inserting a source routing header. This is illustrated in Figure 2. In storing mode, the packet goes upto a common ancestor of the source and destination. Since routing table is stored at each node, the nodes are able to route packets down the DAG, as illustrated in Figure 3.

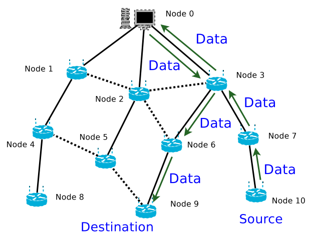


Fig. 2: Data routing in RPL non-storing mode.

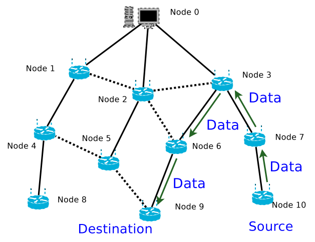


Fig. 3: Data routing in RPL storing mode.

III. DAO specific operation in RPL: Motivation for Optimization

A node generates a DAO packet in the following cases:

- The parent set or most preferred parent of the node has changed;
- The node has received a new DODAG sequence number, indicating a global repair of the network;
- The node has received an increased value of DAO Trigger Sequence Number (DTSN) to refresh downward routes.

As specified in [2], when a new DODAG sequence number or a DAO message from its own sub-DAG is received, a node schedules a new DAO to be propagated upwards. In both cases, the node delays the emission of the new DAO with a timer named DelayDAO Timer. The value of this timer is constant, and its default value is defined by the parameter DEFAULT_DAO_DELAY, which is set as 1 second. Once the DAG root receives information about all the destinations, only then, it will be able to direct P2P or P2MP packets to the proper route. Once it does not have the route to the particular direction, it must drop the packet for that particular destination.

A. *Trade-off on designing the value of DelayDAO Timer*

On one hand, if the value of the DelayDAO Timer is fixed and low, the nodes will quickly emit DAOs after entering global repair (LBR emitted a DIO with a new sequence number) and therefore the whole DAG information should take less time to reach the DAG root. However, within the same DODAG iteration, a node may receive DIOs with better cost to the DAG root through other nodes. Hence, it may switch parents, forcing yet another DAO transmission. On the other hand, a large value of the timer will save on the control plane overhead, but waiting longer at each level of the DAG to generate and/or to forward a DAO to the DAG root will ultimately lead to much delay for the DAG root to construct a full routing table of the DAG. Clearly, there is a trade-off between the number of DAOs transmitted, and the time required by the DAG root to have a full view of the topology.

We have further observed that the effect of the value of DelayDAO Timer, for large scale networks, becomes more severe than a few extra DAOs being issued or the DAOs reaching the DAg root late: intermediate nodes need to store a DAO from their sub-DAG before relaying it to their parents. In an urban LLN implementation consisting of thousands of nodes, nodes closer to the DAG root will need large buffer space to hold DAO packets for later transmission, while these LLN nodes may have only a few KBs of flash memory. Also, as we will observe later in this section, a constant value of DelayDAO Timer for all nodes of different ranks, leads to huge amount of DAOs to be transmitted within a small time duration, therefore creating congestion and further increasing buffer requirement for nodes closer to the DAG root.

B. *Bottleneck due to a constant value of DelayDAO Timer*

Recall that RPL creates a Directed Acyclic Graph (DAG), where links between preferred parents and children create a spanning tree of the network through which data aggregation (MP2P) or dissemination (P2MP) takes place. However, the tree thus created, rooted at the DAG root, is not guaranteed to be a balanced tree, since the number of children under each parent may not be the same. In this section, in the interest of deriving theoretical lower bounds, we assume that the tree created by constructing a DAG is a balanced tree, where every node in the network has B children, and height of the tree is H.

We assume the value of DelayDAO Timer is T_{DD}. The number of DAOs that are generated from the ith level below a node is B^i. These DAOs arrive after $i * T_{DD}$ time, with some random jitter that depends on value of $Imin$. Clearly, in each successive interval of T_{DD}, the node needs to receive B^i DAO packets, and transmit B^{i-1} DAOs. At the same time, during the next interval of T_{DD}, it needs to buffer B^i DAO packets from its own sub-DAG. Hence the minimum buffer requirement would increase exponentially for a node over successive intervals. In Figure 4, we show how the time a node's radio is busy varies with rank and time after it receives a DIO with DTSN increased or DODAG sequence number increased. In Figure 5, the minimum DAO buffer requirement without considering the effect of congestion for nodes of different ranks at different times is plotted. The average number of children is assumed as $B = 3$, and height $H = 8$. Note that this analysis only provides a theoretical lower bound on the required number of DAO packets to be buffered. Clearly, in a real deployment, the radio will find other nodes transmitting at the same time, thus increasing required buffer space.

We also consider a grid like topology of 1000 nodes with disc model for radio propagation, and assume the DAG constructed by RPL is the Breadth First Search (BFS) tree for the topology. In general, the number of DAO packets received by a node at the ith T_{DD} interval will be the same as the number of nodes in the ith level of it's sub-DAG. In Figure 6, we plot the number of times a node's radio is busy transmitting or receiving DAO messages against rank and time interval. Note also that this analysis does not consider the DAO-ACK or acknowledgement

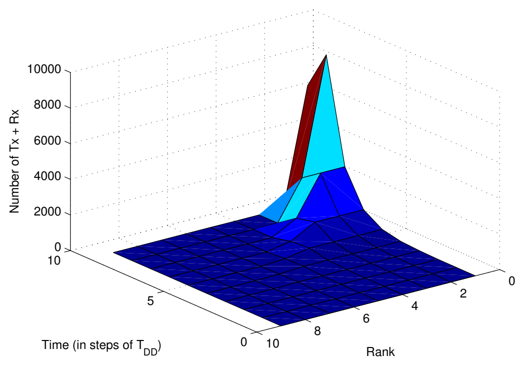


Fig. 4: Total DAO transmissions (Tx) and receptions (Rx) versus time and rank.

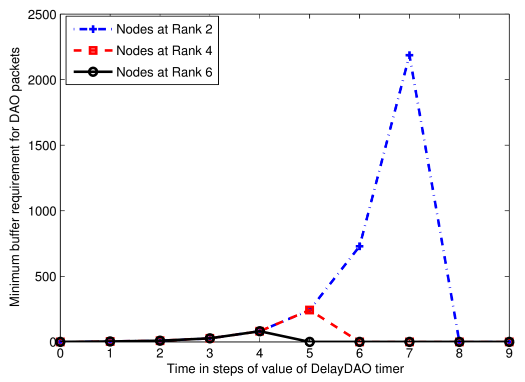


Fig. 5: Minimum DAO buffer requirement against time for different ranks.

packets that traverse down the DAG, while some DAOs are being forwarded up the DAG. Clearly, the results presented in this section are optimistic and provide a lower bound.

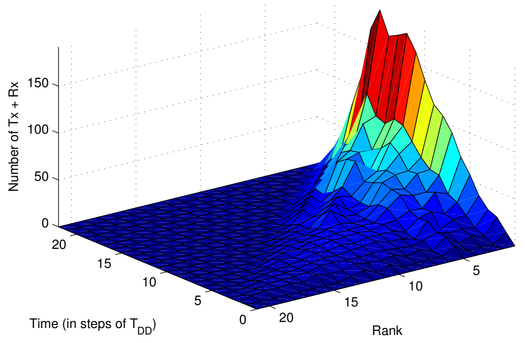


Fig. 6: Total DAO Tx + Rx versus time and rank for the 1000-node grid topology.

From the above results, it is clear that in large networks, during simultaneous route refresh or global repair, the described mechanism to delay DAO packets may fail due to high congestion and excessive buffer requirements. Intuitively, the high buffer requirement stems from the fact that each node needs to store the DAO packets from its sub-DAG for possible aggregation of DAO routes. However, while implementing non-storing mode, route aggregation is not performed. Hence, a node should forward a DAO from its sub-DAG immediately to its parent, as opposite to what is pointed out in [2]. Secondly, in a particular level in the DAG, the generation time for DAO packets should be distributed within a time duration to prevent propagation of a high number of DAO packets at the same time. The time duration should increase exponentially with rank as we have observed that the number of nodes tends to increase exponentially with rank. This is illustrated in Algorithm 1, which determines the DelayDAO Timer duration. It assumes that each node generates a random number amongst an interval that increases exponentially with the rank of the node. In the next sections, we will present mechanisms that will be used to estimate values of the two parameters in this algorithm, 'K' and '$Base$'.

Algorithm 1 Generation of DelayDAO Timer Value

```
seed = <EUI_64>/MAC address or node ID
$T_{DD} \leftarrow$ random($K * Base^{Rank-1}$, $K * Base^{Rank}$);
DAOStartTime := CURRENT_TIME;
Arm DelayDAO Timer with value = $T_{DD}$
Issue a DAO when DelayDAO Timer fires.
```

IV. Determining DelayDAO - A Distributed Algorithm

In each node, the algorithm starts with an initial value of the two parameters, 'K' and '$Base$', and uses adaptive filtering to correctly estimate these parameters. The goal of this algorithm is for the nodes to obtain a value of DelayDAO Timer that minimizes buffer occupancy in nodes near the DAG root. No node, however, is aware of buffer size in any other node but itself. Hence, this problem is different than a classic control theory problem, as the output and feedback are not directly related. Nodes far away from the DAG root can only speculate about a possible congestion near the DAG root by examining the round-trip time of a packet it transmits. In LLNs, not all the data packets need an acknowledgement back to the sender. Hence, this algorithm uses the round-trip time of the DAO packets to estimate probable congestion or large buffer size incident. If this round-trip time is too high, the constant K value is increased by a factor δ_K, and if it is lower than a certain time limit, the value of K is decreased by the same factor. If DAO-ACK-TIME, or the threshold to wait for a DAO-ACK packet, is exceeded, a severe congestion is surmised, and the value of $Base$ is increased by a factor δ_B. Clearly, the base of exponent $Base$ is used for coarse tuning of the value of DelayDAO Timer, where the constant K is used for fine tuning. The routine upon reception of each DAO-ACK is described in Algorithm 2.

The constants T_U, T_L and T_B depend on the transmission time T_{Tx} of the DAO packets. If the DAO packets have a length of L_{DAO}, and the data rate of the radio is given by B_R, we chose these constants as :

$$T_{Tx} = \frac{L_{DAO}}{B_R}\ ,\ \ T_U = 8{*}T_{Tx}\ ,\ \ T_L = 4{*}T_{Tx},\ \ T_B = 2{*}T_{Tx}.$$

Algorithm 2 Distributed Parameter Estimator

```
$RoundTripTime \leftarrow Current\_Time - DAOStartTime$;
**if** $RoundTripTime > T_U \times Rank$ **then**
    $K \leftarrow K \times (1 + \delta_K)$;
**end if**
**if** $RoundTripTime < T_L \times Rank$ **then**
    $K \leftarrow K \times (1 - \delta_K)$;
**end if**
**if** $RoundTripTime >$ DAO-ACK-TIME **then**
    $Base \leftarrow Base \times (1 + \delta_B)$;
**end if**
**if** $RoundTripTime < T_B \times Rank$ **then**
    $Base \leftarrow Base \times (1 - \delta_B)$;
**end if**
```

V. Determining DelayDAO - Centralized Approach

The DAG roots or border routers in LLNs posses much more computational power than any other node in the network, and also more memory space to buffer packets or store routing tables. Since all nodes send their DAO message to the DAG root, the root has an overall view of the whole network. Hence it would be advantageous to outsource the computations related to the network to the DAG root. In this section, we will present an algorithm that computes the $Base$ and K parameters upon receiving DAOs from the network. After computation, these values are distributed in the network during the next global repair or increased DTSN via a new DIO packet. The values of $Base$ and K can be added as objects in the DIO packet.

We define the node rank set as L, which contains for each rank $R, (1 \leq R \leq H+1)$ the nodes that are at rank R. Also, the DAG root maintains a parent list P to contain the parent for each node n in N. We define the function Find_Rank to intake a node ID, and returns the rank of the node as illustrated in Algorithm 3.

Algorithm 3 Rank Finder Algorithm

```
int Find_Rank(node n)
{
    int Rank = 0;
    P := Parent(node n);
**if** $P == DAG\_Root$ **then**
        return(1);
**end if**
    Rank := Rank + Find_Rank(node P);
    return(Rank);
};
```

Upon receiving each DAO, the data structures are updated as in Algorithm 4.

Before each global repair or DTSN increase, the DAG root estimates the parameters to determine the value of DelayDAO Timer for all nodes in the network as shown in Algorithm 5.

Of course, the centralized and distributed algorithms can be combined to achieve better convergence. The DAG root via the centralized algorithm can provide the initial parameter values for $Base$ and K, which can be further tuned by the nodes via Algorithm 2. Due to space constraints, the effect of the combined approach is not analyzed in this study.

Algorithm 4 Construction of Node Rank Set

```
$n \leftarrow$ Source of DAO message;
$R_n \leftarrow$ Find_Rank(n);
$L[R_n] \leftarrow L[R_n] \setminus \{n\}$;
Update Route_Table with new parent for n;
$\hat{R_n} \leftarrow$ Find_Rank(n);
$L[\hat{R_n}] \leftarrow L[\hat{R_n}] \cup \{n\}$;
```

Algorithm 5 Centralized Parameter Calculator

```
$W \leftarrow \max(\{L[i]\})\ ,\ 1 \leq i \leq H$;
$R_W \leftarrow R$ s.t. $\{L[R] = \max(\{L[i]\})\ ,\ 1 \leq i \leq H+1;\}$
$Base \leftarrow \exp(\frac{\ln W}{R_W})$;
$K \leftarrow \max(2 * T_{Tx} \times \frac{i * \ln i}{Base^i})\ ,\ 1 \leq i \leq H+1$;
```

VI. Evaluation of the algorithms

A. Simulation setup and metrics

To study the behavior of RPL in different networks in [6], [7]) and ([8], the authors designed a detailed RPL simulator. The simulator was developed using OMNET++ [10] discrete-event simulator engine. Since urban LLNs posses highly time dependent attributes, a fixed probabilistic packet delivery or link condition does not represent typical challenges in an urban network. Hence, a database to model how link quality varies in practically deployed LLNs was created by gathering real link layer data from outdoor and smart meter deployments. A topology of 2442 nodes deployed in a city was considered in this study. The deployment is replicated in the simulation by replicating the network topology and simulating identical temporal variation of the links. Each link in the topology 'picks up' the corresponding link quality model between the same neighbor pairs in the original deployment. Therefore, the link's PDR in simulation varies according to the gathered traces of the same link with respect to time, creating a "pseudo-replica" of the real network.

To analyze topology dependency of results, we have also considered random topologies of 200, 500 and 1000 nodes, where the nodes are distributed in a grid fashion. In all these networks, each link quality is time-dependent and modeled after temporal variation of a link in the created database, and the link's PDR varies with time in the same manner the link quality varies with time in a real deployed network. All these networks employ a 802.15.4 MAC/PHY model [11], and a CC2420 radio model for TelosB motes. To compute maximum required buffer space or memory, no buffer drop is simulated. Each simulation is run for a day (simulation time). The traffic pattern, as provided by smart grid AMI meter vendors, is described below.

- On-demand operation traffic: The collection point requests reading of index from each meter in a round robin fashion, which is reported back. Time period between two readings from same node = $F1$ which is normally 2 hours, unless otherwise noted.

- Polling traffic: The collection point retrieves information (statistic report, status, load curve) from each meter once a day in same fashion as above.
- Traffic due to multicast parameter modification (e.g. new tariff): 50 bytes sent by the LBR to a set of nodes, multicast, with frequency once a day.
- Traffic due to alarms: From a meter (node) to the LBR and unsolicited, unidirectional and random. 20 bytes of data is sent by each meter to the LBR, once a day.

We mainly concentrate on three metrics which are closely related to performance of RPL when congestion or memory requirement is concerned.

- DAO reach time: Denotes the time required (in seconds) for the DAO of a node to reach the DAG root after the global repair or new DTSN is issued.
- Packet buffer size: Denotes the number of packets in the node, waiting to be transmitted.
- RAM consumption: Denotes flash memory consumption in bytes to store different state variables, data structures, parent tables, buffered packets, etc. but not RPL's code itself, as it may differ among implementations. However, the results do not consider the memory consumption of the DAG root since it is not a constrained device.

We also consider the round trip time of a DAO packet, which is calculated by the time difference between issuing a DAO packet, and receiving the corresponding acknowledgement.

B. Simulation results

Figures 7 and 8 show how the CDFs of DAO reach time vary with time for the distributed algorithm and the centralize algorithm, respectively, run in the 2442 node network. These two plots demonstrate that the proposed approaches have the ability to decrease the DAO reach time, helping the protocol perform better as the time progresses. Figure 9 shows the CDFs of DAO reach time for the proposed algorithms and the default mechanism described in [2]. The default mechanism assumes a DelayDAO Timer value of 6 seconds. The default value of 1 second for DelayDAO Timer proved to be too low for the large network, and incurred congestion that inhibits normal operation of the network, hence not used for default mechanism simulation. The centralized approach delivers DAOs to the DAG root faster than the default mechanism in 95% of the cases, where the distributed approach delivers 80% of the DAO packets before the default mechanism. Clearly, the proposed mechanisms outperform the default one in most cases, besides also winning in memory consumption, as described next.

To study memory consumption, we consider different network sizes of 200, 500, 1000 and 2442 nodes. In Figure 10, we show how the maximum buffer size in the nodes varies as the network size grows larger. In Figure 11, the maximum amount of memory consumed is plotted for the three mechanisms, showing the savings brought on by the centralized and further, the distributed approach. The savings are more significant as the network scale grows; for the 2442-node network, the default mechanism consumes around 800 KB of memory, whereas only 90 and 30 KB of memory are consumed by the centralized and distributed approach, respectively. In Figure 12, we also observe that the proposed mechanisms decrease the average control overhead by a small fraction. The small amount of control overhead savings is a direct result of the reduced number of DAO packets being re-issued due to congestion.

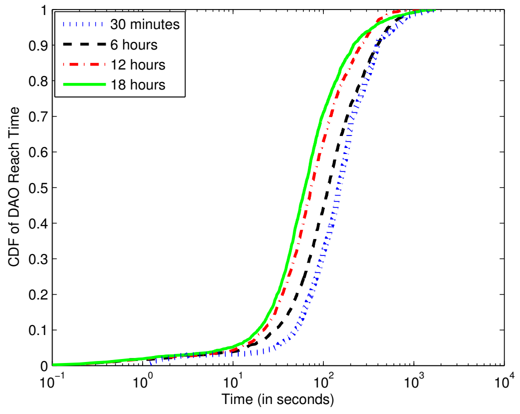


Fig. 7: DAO reach time for distributed algorithm.

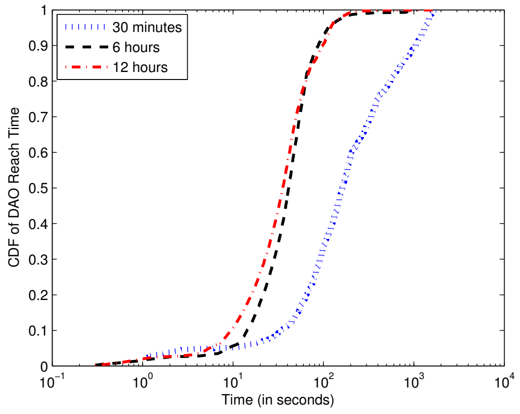


Fig. 8: DAO reach time for centralized algorithm.

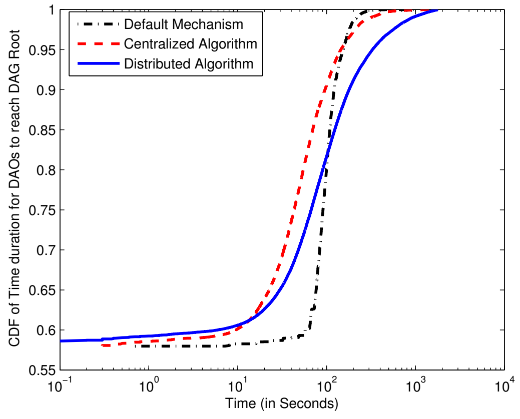


Fig. 9: DAO reach time for all mechanisms.

In Figure 13, we plot the CDF of the round-trip time of DAO

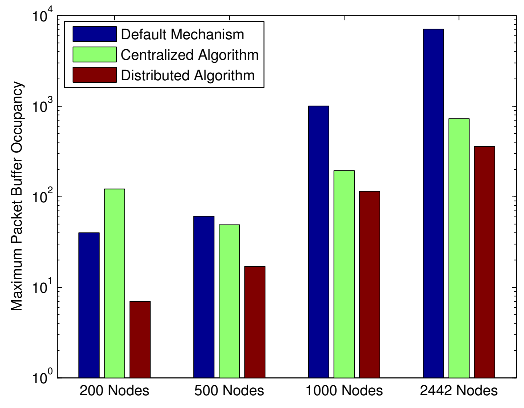


Fig. 10: Maximum packet buffer size.

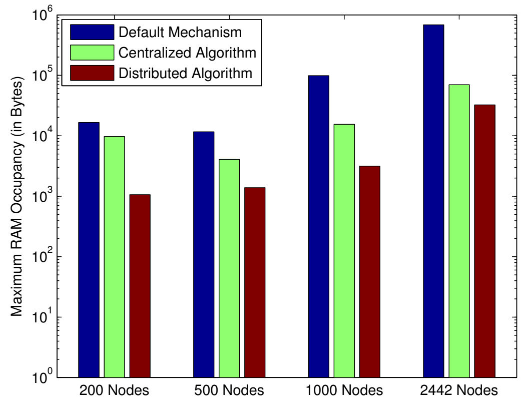


Fig. 11: Maximum RAM consumption.

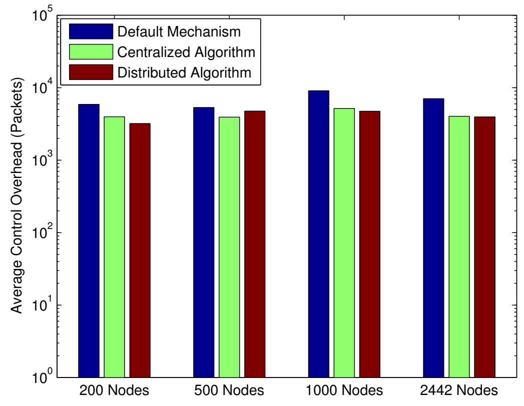


Fig. 12: Average control packet overhead.

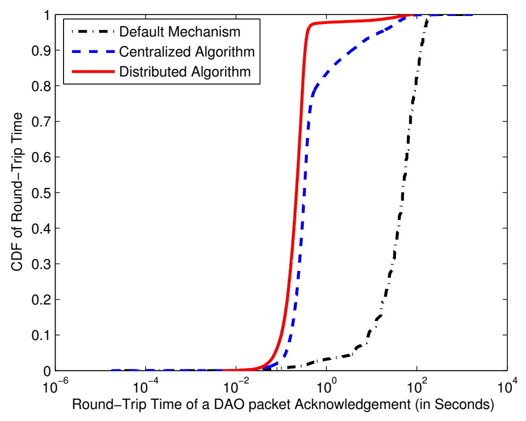


Fig. 13: DAO round-trip time.

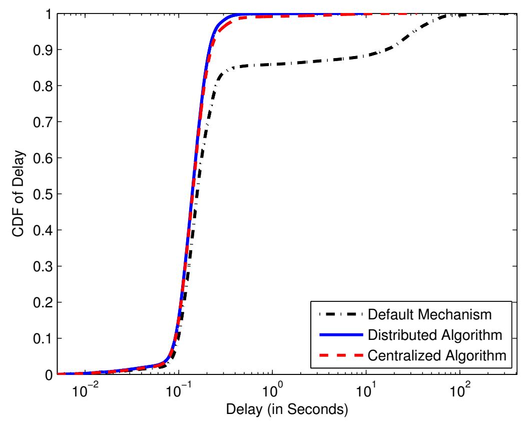


Fig. 14: Data packet delivery latency.

packets, which is calculated as the time difference between issuing a DAO packet and receiving the corresponding ACK. Clearly, the two algorithms decrease the round-trip time by considerable amount. These results help establish that the proposed approaches do function as intended in a large network. We also considered the end-to-end delay experienced by data packets, since congestion highly effects this metric. In Figure 14, it can be observed that the delay experienced is much smaller for the proposed methods than for the default one.

Also, since we consider a distributed adaptive filtering based approach to obtain the parameters to determine DelayDAO Timer value, it is important to show that the approach leads to a stable output with respect to time. In Figure 15, we show how the 'K' parameter value varies with time for nodes of different ranks in the network. We plot the average 'K' parameter values for all nodes at rank 4, 8, 12 and 16, and observe that all these values become stable after a few hours of operation.

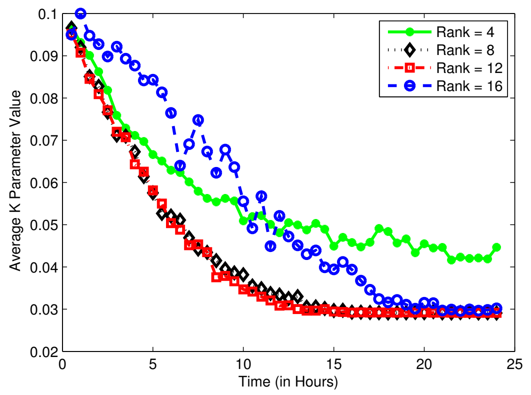


Fig. 15: Stability of 'K' with time.

VII. Emission of DAOs in RPL Storing Mode

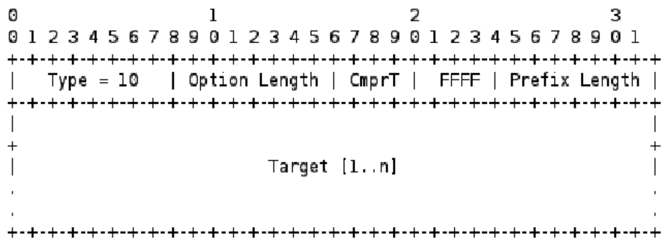
In RPL storing mode, DAO packets are not unicasted to the DAG root, but are rather advertised to the node's parents. Thus, these DAO packets reach the DAG root in a hop-by-hop fashion, being acknowledged at every single hop. Nodes in storing mode may or may not aggregate DAOs from their subtree. If aggregation of DAO packets is not performed, we propose the storing mode to employ similar technique as non-storing mode for DAO generation. In absence of prefix aggregation, a node in storing mode has to forward the same number of DAO packets as in non-storing mode for the same topology. Hence, the same centralized approach as in non-storing mode is expected to provide similar benefits in RPL storing mode. The proposed distributed approach, however, cannot be applied to storing mode, as in this mode the DAOs are acknowledged at every single hop, and no end-to-end acknowledgement is available.

As described in [2], a node should delay sending the DAO message in order to aggregate the DAO information from other nodes for which it is a DAO parent. RPL non-storing mode requires the parent node to send their DAO packets before the children, so the value of DelayDAO Timer should increase with the node's rank in this mode of operation. However, aggregation of destination addresses will surely get hurt by such a mechanism. The default mechanism of a constant universal timer will lead the nodes with higher rank to release their DAO later, thus harming the aggregation process. For aggregation to be successful, nodes of farthest distance (in rank) from the DAG root should release their DAO first. In an ideal situation, a node should wait till it receive DAOs from all nodes in its subgraph, aggregate them as much as possible, and finally send it to its parent. Intuitively, DAO aggregation needs a mechanism where the value of the DelayDAO Timer decreases with the rank. However, the RPL standard ([2]) does not specify rules on aggregated DAO packet generation and processing. Hence, in this Section, we will briefly introduce a new packet format for DAO aggregation that does not violate RPL standard message format and adds to the original DAO Base Object defined in the RFC. At the same time, we will introduce a novel approach for DAO packet generation, storage and forwarding.

A. Aggregated DAO packet format

In [2], Type = 0x05 is reserved for a target option that adds to a DAO base object. This target option carries the destination address in a DAO packet. Type values within the range $0\mathrm{x}01-0\mathrm{x}09$ are reserved. We define a new DAO option, called *RPL Aggregated Target Option*, which is defined by the type value 0x10. The Aggregated Target Option possesses the same format as the target option originally defined. However, 4 bits amongst the 8 unused flag bits are used as a new field, denoted as *CmprT*. Our proposed RPL Aggregated Target Option is illustrated in Figure 16.

In our proposed RPL Aggregated Target Option, the CmprT field defines the level of compression applied to each target address, so that each target has an address length of



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 10   | Option Length | CmprT |  FFFF | Prefix Length |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                         Target [1..n]                         |
.                                                               .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 16: Proposed DAO target option for aggregation.

$(16 - CmprT)$. As the possible values of CmprT range between 0 and 15, the length of each address field in the Aggregated Target Option may vary between 1 and 16 octets. The number of targets present in the aggregated DAO can easily be calculated as:

$$n \leftarrow \frac{(Prefix\ Length-4)}{(16-CmprT)}$$

We define $MaxTarget$ as the maximum number of target destination addresses that may be accommodated in a single aggregated DAO. Clearly, if the Maximum Transmission Unit (MTU) of a network is known, and $DAOOverhead$ defines the overhead associated with a DAO packet without the 'Target' field, $MaxTarget$ can be calculated as:

$$MaxTarget \leftarrow \frac{(MTU-DAOOverhead)}{(16-CmprT)}$$

We will explain this with a couple of examples. For 802.15.4 links, the MAC layer presents a maximum of 102 bytes of payload to the network layer. If we consider DAO packets with the proposed aggregation option, it can be shown that the total overhead incurred is 64 bytes without the 'Target' field. If the targets include a full IPv6 address of 16 bytes, the scope of aggregation is clearly, very limited. However, we can use header compression as in [12], or use a compressed address field of 2 bytes instead of the full 16 bytes. Using a 2 bytes address field and unchanged header format, a maximum of $(102-64)/2$ or 19 destinations can be aggregated in a single DAO packet. Also, 802.15.4g [13] has recently been amended to support Smart Utility Networks (SUN). 802.15.4g can handle a maximum frame-size of 2047 bytes, well over IPv6 defined maximum frame-size of 1280 bytes. If we consider a full IPv6 packet with an MTU of 1280 bytes, even with uncompressed header and uncompressed address field of 16 bytes length, a single DAO may support up to 76 destinations together. However, link PDR may decrease with increased packet size, forcing an upper bound on $MaxTarget$ for certain environments.

B. DAO aggregation algorithm

The following notation and parameters are used in the algorithm:

- SubTree: The number of valid routes in a node's routing table;
- R: Current rank of the node;
- ReleaseQ: A queue of unreleased destinations present at

each node, with a timestamp of when the destination is added;

- $T_L = 4 * T_{Tx}$, as defined in Section IV.

Upon receipt of a new DODAG sequence number or a new DTSN, Algorithm 6 is run to determine the generation of a DAO packet by the node. The DelayDAO timer is inversely proportional to rank, as discussed earlier in this section.

Algorithm 6 Generation of DelayDAO Timer Value

```
    seed = <EUI_64>/MAC address or node ID
    $T_{DD} \leftarrow$ random($\frac{T_L}{R-1} \times SubTree$ , $\frac{T_L}{R} \times SubTree$);
    Arm DelayDAO Timer with value = $T_{DD}$.
    Enque (self, CURRENT_TIME) to ReleaseQ.
    when DelayDAO Timer fires.
```

Clearly, even when a DAO is generated, it is not forwarded immediately. A node always keeps an eye on its ReleaseQ, to check if a destination is waiting too long to be transmitted. The routine is described in Algorithm 7.

Algorithm 7 ReleaseQ Checker

```
  $T \leftarrow$ Timestamp of first entry in ReleaseQ.
  if $T - CURRENT\_TIME \geq \frac{T_L}{R} \times SubTree$ then
      if Size of ReleaseQ < MaxTarget then
          Aggregate all destinations in a DAO.
      else
          Aggregate $MaxTarget$ destinations in a new DAO.
      end if
  end if
```

Upon receipt of a DAO packet from another node, Algorithm 8 defines the rules for DAO aggregation and forwarding.

Algorithm 8 Destination Aggregation and DAO Emission

```
  if $n < MaxTarget - 1$ then
      Extreact each destination, $D$.
      update route table with $D$.
      Enqueue each ($D$, $CURRENT\_TIME$) into ReleaseQ.
      if Size of ReleaseQ $\geq$ MaxTarget then
          Aggregate $MaxTarget$ destinations in a new DAO.
      end if
  else
      Update route_table with all destinations.
      Forward DAO Packet as is, without inserting it into ReleaseQ.
  end if
```

VIII. RPL Storing Mode Evaluation

We used a similar setup and terminology as in Section VI. However, along with 802.15.4 links, we implemented 802.15.4g as well, to observe the effect of the maximum allowable aggregation. Simulations with 802.15.4 links consider a compressed address field of 2 bytes, whereas no compression is employed for 802.15.4g links, as mentioned in Section VII. To be noted, even if 802.15.4g allows a maximum frame size of 2047 octets, a DAO packet in our simulation may have a maximum frame size of 1280 octets, as RPL is designed to run under IPv6.

In Figure 17, we plot the CDF of DAO reach time value for both, default mechanism and proposed modification for storing mode. This result shows how the proposed algorithm, with the help of aggregation, improves the DAO reach time helping them reach the DAG root faster than with the default mechanism. It should be noted that with increase in MTU, nodes wait longer to aggregate and transmit the DAO, and hence, the DAO reach time also increases.

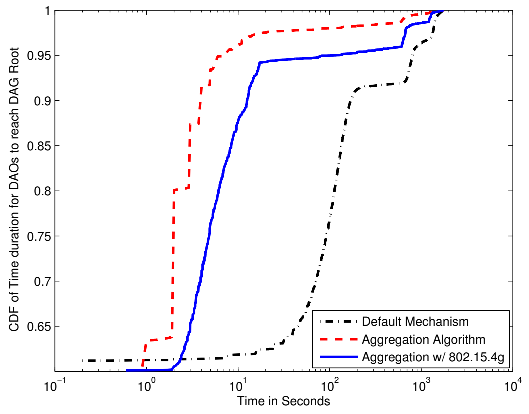


Fig. 17: DAO reach time for both mechanisms.

We next consider 5 networks of size 86, 200, 500, 1000 and 2442 nodes to evaluate topology-dependency of the results. In Figure 18, we show how the proposed mechanism benefits RPL in terms of maximum memory consumption. Note that we do not compare the number of control packets for these two mechanisms. As aggregation includes multiple addresses in a single DAO, the number of control packets is definitely smaller than with the default mechanism, but each packet has larger size. Hence, to be fair, in Figure 19 we plot the average control overhead (in bytes) for the two mechanisms for networks of different size.

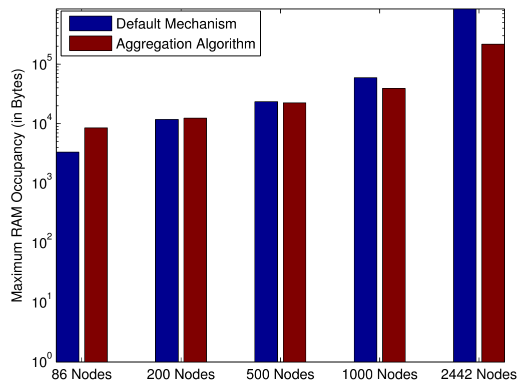


Fig. 18: Maximum RAM consumption.

We also considered the end-to-end delay experienced by data packets in the large 2442-node network, as we did for the

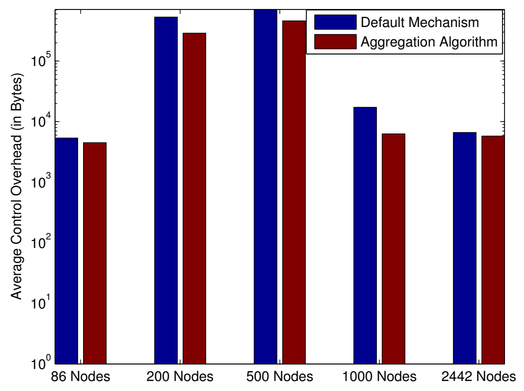


Fig. 19: Average control packet overhead.

non-storing mode algorithms. In Figure 20, it can be observed that the delay experienced is much smaller for the proposed method with aggregation than for the default one. Note that the proposed aggregation method does not aggregate any data packet. Still, by reducing congestion and thus providing more resource to the network, the proposed approach provide less delay for 65% of the packets. As observed, the maximum delay experienced by any data packet for the proposed approach is also much smaller. Clearly, the link MTU does not have a big effect on data delivery once the congestion is mitigated, as we can observe the two MAC/PHY models of 802.15.4 provide very close data delivery delay.

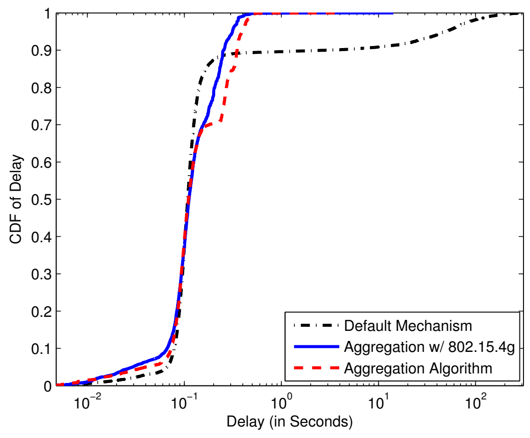


Fig. 20: Data packet delivery latency.

IX. Conclusions and Future Work

In this paper, two mechanisms are proposed to resolve the congestion and high memory and buffer requirements in the standard RPL non-storing mode implementation, which makes use of a fixed, universal DelayDAO Timer. The proposed mechanisms are even more useful in topologies where the data collection point is at the center of the deployment, and where the number of nodes exponentially grows with distance from the root. It can be observed, that the centralized approach provides less time for the DAG root to gather routing information for the whole network than the distributed algorithm. Both proposed mechanisms provide fairly low RAM usage, especially in large topologies. These algorithms also incur less delay for data packet delivery by controlling the control plane congestion. A DAO generation and aggregation mechanism is also proposed for RPL storing mode, which is observed to provide the DAG root with faster congregation of addresses in the network, less control overhead and faster delivery of data packets, by lowering DAO congestion.

Different approaches can be considered which may use the information from the DAO-ACK packets to indirectly detect large buffer occupation and congestion near the DAG root or low rank nodes in RPL non-storing mode. This information may be used to delay transmission of DAO packets to reduce buffer requirement in LLNs. Different congestion control techniques for transport protocols exist in the literature, which might shade new light or hint at new directions to this problem. Nevertheless, it is of most importance to devise a suitable technique to schedule DAO packets in RPL for large networks.

X. Acknowledgements

The authors would like to acknowledge JP Vasseur's invaluable guidance in the design of the RPL simulator.

References

[1] M. Dohler, T. Watteyne, T. Winter, and D. Barthel, "Routing Requirements for Urban Low-power and Lossy Networks," RFC 5548, May 2009. [Online]. Available: http://tools.ietf.org/html/rfc5548

[2] T. Winter et al., "RPL: Routing Protocol for Low Power and Lossy Networks," RFC 6550, March 2012. [Online]. Available: http://tools.ietf.org/html/rfc6550

[3] A. Brandt, J. Buron, and G. Porcu, "Home Automation Routing Requirements in Low power and Lossy Networks," RFC 5826, April 2010. [Online]. Available: http://tools.ietf.org/html/rfc5826

[4] J. Martocci, P. D. Mil, N. Riou, and W. Vermeylen, "Home Automation Routing Requirements in Low power and Lossy Networks," RFC 5867, June 2010. [Online]. Available: http://tools.ietf.org/html/rfc5867

[5] K. Pister, P. Thubert, S. Dwars, and T. Phinney, "Industrial Routing Requirements in Low-Power and Lossy Networks," RFC 5673, October 2009. [Online]. Available: http://tools.ietf.org/html/rfc5673

[6] J. Tripathi, J. de Oliveira, and J. Vasseur, "A performance evaluation study of RPL: Routing Protocol for Low power and Lossy Networks," in *Proceedings of the 44th Conference on Information Sciences and Systems (CISS'10)*, March 2010.

[7] ——, "Applicability Study of RPL with Local Repair in Smart Grid Substation Networks," in *Proceedings of SmartGridComm'10*, October 2010.

[8] ——, "Performance Evaluation of Routing Protocol for Low power and Lossy Networks (RPL)," DRAFT, April 2012. [Online]. Available: http://tools.ietf.org/html/draft-tripathi-roll-rpl-simulation-08

[9] D. Wang, Z. Tao, J. Zhang, and A. Abouzeid, "RPL Based Routing for Advanced Metering Infrastructure in Smart Grid," in *Proceedings of IEEE ICC'10*, May 2010.

[10] A. Varga, "The OMNeT++ Discrete Event Simulation Systems," in *Proceedings of the European Simulation Multiconference*, June 2001.

[11] "IEEE Standard for Local and metropolitan area networks– Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)," September 2006. [Online]. Available: http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf

[12] J. Hui and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," RFC 6282, September 2011. [Online]. Available: http://tools.ietf.org/html/rfc6282

[13] "IEEE 802.15 WPAN Task Group 4g (TG4g)Smart Utility Networks." 2012. [Online]. Available: http://standards.ieee.org/findstds/standard/802.15.4g-2012.html