

Routing Loops in DAG-based Low Power and Lossy Networks

W Xie*, M Goyal*, H Hosseini*, J Martocci[†], Y Bashir[†], E Baccelli[‡] and A Durresi[§]

*University of Wisconsin - Milwaukee, Milwaukee, WI 53201 USA, {wxie,mukul,hosseini}@uwm.edu

[†]Johnson Controls, Milwaukee, WI 53202 USA, {jerald.p.martocci,yusuf.bashir}@jci.com

[‡]INRIA, Paris, France, emmanuel.baccelli@inria.fr

[§]IUPUI, Indianapolis, IN 46202 USA, durresi@cs.iupui.edu

Abstract—Directed Acyclic Graphs (DAGs), rooted at popular/default destinations, have emerged as a preferred mechanism to provide IPv6 routing functionality in large scale low power and lossy networks, that include *wireless sensor networks* and those based on *power line communication*. A DAG maintains its acyclic nature by requiring that each DAG node must have a higher ‘rank’ than any of its DAG parents. While a node may decrease its DAG rank safely, increasing its DAG rank to add a new parent may result in a routing loop if the new parent is also a descendant in the DAG. In this paper, we first study via simulations the time required by the network to converge to a stable, loop-free state following a rank increase operation and the number of routing messages generated (the network ‘churn’) during this time. Then, we describe the precautionary measures that can be used to avoid routing loops and evaluate via simulations how these measures affect the time and churn involved in reaching a stable state following a rank increase operation.

I. INTRODUCTION

Designing a routing protocol for large *low-power and lossy networks* (LLN), consisting of thousands of memory, power and CPU constrained nodes and unreliable links, presents unique challenges. The *Routing Over Low-power and Lossy networks* (ROLL) working group in *Internet Engineering Task Force* (IETF) is currently engaged in designing a suitable routing protocol for LLN-based applications in industrial, urban, commercial building and home domains [1]–[4]. These requirements have identified *multipoint-to-point* data collection as a dominant traffic pattern in large scale LLN. Moreover, the need for highly scalable operation is a common theme among all requirement sets. Levis et.al. [5] identified the following scalability requirements that a suitable routing protocol must satisfy:

- The routing state that a node needs to maintain must not increase linearly with the number of nodes in the network or in the neighborhood;
- Local events, such as loss of connectivity between two nodes, must not lead to network-wide broadcast of routing messages;
- The *control cost*, i.e. the rate at which the routing messages are sent or received, must be bounded by the rate of data packets.

In this context, the ROLL working group is leaning towards a routing protocol, called *RPL*¹, that allows the nodes to organize themselves as one or more *directed acyclic graphs* (DAGs), rooted at nodes that serve as popular destinations or provide default routes to rest of the Internet [6]. A DAG differs from a tree in the sense that a node is allowed to select multiple neighbor nodes as its parents in the DAG. The packets going towards the root can only be forwarded along DAG links - a node always forwards a received/self-generated packet to one of its DAG parents. While the DAGs naturally support multipoint-to-point routing towards their roots, simple extensions allow them to be used for point-to-multipoint and point-to-point routing as well [6].

A DAG maintains its *acyclic* (i.e. loop-free) nature by requiring that a node must always have a higher *rank* than any of its DAG parents. A node’s DAG rank is closely related to (but is not necessarily same as) its routing cost to reach the DAG root. A DAG node periodically advertises its DAG rank as well as its routing cost to reach the DAG root by doing a local broadcast of its *DAG Information Object* (DIO) message. Before joining a DAG, a node monitors the DIO messages of its neighbor nodes and selects a subset of them as parents based on the routing costs they advertise. Then the node determines its DAG rank by choosing a *most preferred* parent and adding to the most preferred parent’s rank a *step*, whose value is based on the routing cost of its link to the most preferred parent. While selecting its rank, the node must also ensure that the selected rank is higher than that of any of its parents. Whenever a node changes its rank, it must eliminate from its parent set all the nodes that no longer have a smaller rank than itself. Thus, the DAG ranks are used to enforce an *acyclic* structure on the DAG and are expected to be much less dynamic than the routing costs on which they are based. Once a node has chosen a set of parents in a DAG, it can choose any of these parents, possibly based on the actual routing costs of these parents, to forward a packet towards the root.

The generation of DIO messages by a node is governed by a *trickle* timer [7]. The node maintains a *DIO Interval* (I), which may vary between a minimum (I_{\min} ; possibly as small as a few ms) and a maximum (I_{\max} ; several minutes

¹Routing Protocol for low power and Lossy networks

or higher) value, and a *redundancy counter* (C), which is reset to zero every time the trickle timer is started. The timer is scheduled to fire after a random time in $[I/2, I]$ range. While the timer is running, the node increments counter C whenever it receives a *consistent* (i.e. advertising the same rank and routing costs as before) DIO from a parent. At the firing of the timer, the node generates a new DIO if the redundancy counter C is less than a threshold value, doubles the value of interval I (up to its maximum value I_{\max}) and restarts the trickle timer. The receipt of an inconsistent DIO from a parent or a change in node's own rank and routing cost to DAG root forces the node to immediately reset interval I to its minimum value I_{\min} and restart the trickle timer. Thus, the resetting of interval I to I_{\min} allows fast generation of DIO messages when there is a need to propagate new information down the DAG; otherwise exponential increase in I towards I_{\max} causes the DIOs to be generated infrequently. The redundancy counter provides an additional level of control over DIO generation.

As mentioned before, a node must always maintain a higher *rank* than any of its parents in the DAG. However, a node need not stay at the same rank at which it joined the DAG initially. Changes in the rank/cost of existing parents or the need to add new parents may require a node to change its rank. A node may decrease its rank, i.e. move closer to the root, at any time. Such a move only requires eliminating from the parent set all those nodes that no longer have a smaller rank than the node itself. A node may safely (i.e. without creating a routing loop) increase its DAG rank if it is not adding a new parent or if the new parent has a smaller rank than the node's current rank. However, if a node (say A) increases its DAG rank in order to add a new parent (B) that has a higher rank than node A 's current rank, such a move may create a routing loop because node B could be in node A 's *sub-DAG*, i.e. the node A could be an *ancestor* of node B . Such a routing loop may be quickly resolved if the increase in node A 's rank forces its children/descendants to remove it as a parent/ancestor without affecting node B 's rank. If the increase in node A 's rank forces an increase in node B 's rank, node A may need to further increase its rank in response and the loop resolution may require more time and *churn* (i.e. the number of DIO messages). In the worst case, when the nodes in the routing loop have no alternate parents, these nodes will repeatedly increase their rank until it reaches the maximum value, i.e. a *count to infinity* situation occurs, generating frequent DIOs in the process.

In this paper, we study the routing loops resulting from rank increase operations in a DAG via simulations on a sample network topology. In the next section, we present simulation results regarding the duration of these routing loops and the number of routing messages generated (the network *churn*) during their resolution. Then, in section III, we describe the precautionary measures that can be used to

avoid routing loops and evaluate, via simulations, how these measures affect the time and churn involved in reaching a stable loop-free state following a rank increase operation. Section IV concludes the paper.

II. SIMULATIONS TO STUDY ROUTING LOOPS CAUSED BY INCREASE IN DAG RANK

In this section, we present the simulation results regarding the duration of routing loops that result when a node increases its DAG rank to add new parents. We also present the number of routing messages generated during the resolution of these loops. These simulations were performed using an implementation of RPL protocol [6] in the NS2 simulator [8]. In these simulations, the simulated nodes use *beaconless* IEEE 802.15.4 operating with default configuration in 2.4 GHz range as the MAC/PHY layer protocol and organize themselves in a DAG. The IEEE 802.15.4 module used in these simulations is an extensively improved version [9] of the native IEEE 802.15.4 module in NS2 simulator.

The simulations were performed on a network of 1001 nodes distributed in a $632m \times 632m$ region. This number represents the expected upper limit on the number of nodes per DAG in the real deployments. The node locations were determined one-by-one in the following manner. The x and y coordinates of a new location were determined in a uniform random fashion in range $\{0m, 632m\}$ under the constraint that the minimum distance between a new location and an existing location should not be less than 10m or larger than 30m. This was done to ensure that a new location is always in the radio range of at least one existing location. The radio range for each node in these simulations was a circle with radius 31.45m. Thus, we ensured that there were no partitions in the simulated topology. Figure 1(a) gives a visual representation of the simulated network topology and figure 1(c) shows the connectivity in the topology in terms of the number of nodes having a given number of neighbors in their radio range.

As mentioned before, a node calculates its rank in the DAG by adding a *step* value to the rank of its *most preferred parent*. A node may choose any neighbor as its most preferred parent based on the local policy in this matter. The rank step has a value between 1 and 16 and is calculated based on the routing cost of the link to the most preferred parent. The rank step values 1, 4 and 16 signify a perfect, normal and an almost unusable link respectively. The rank step value between two neighbor nodes in the simulated network topology was determined as 2^x rounded to the closest integer, where x is a real number uniformly distributed over range $[0, 4]$. The two ends of the link use the same rank step value for the link. The rank step values did not change during the course of a simulation. The same set of rank step values were used in all the simulations reported in this paper. Figure 1(d) shows the distribution of rank step values of the links in the topology. Figure 1(b) shows the

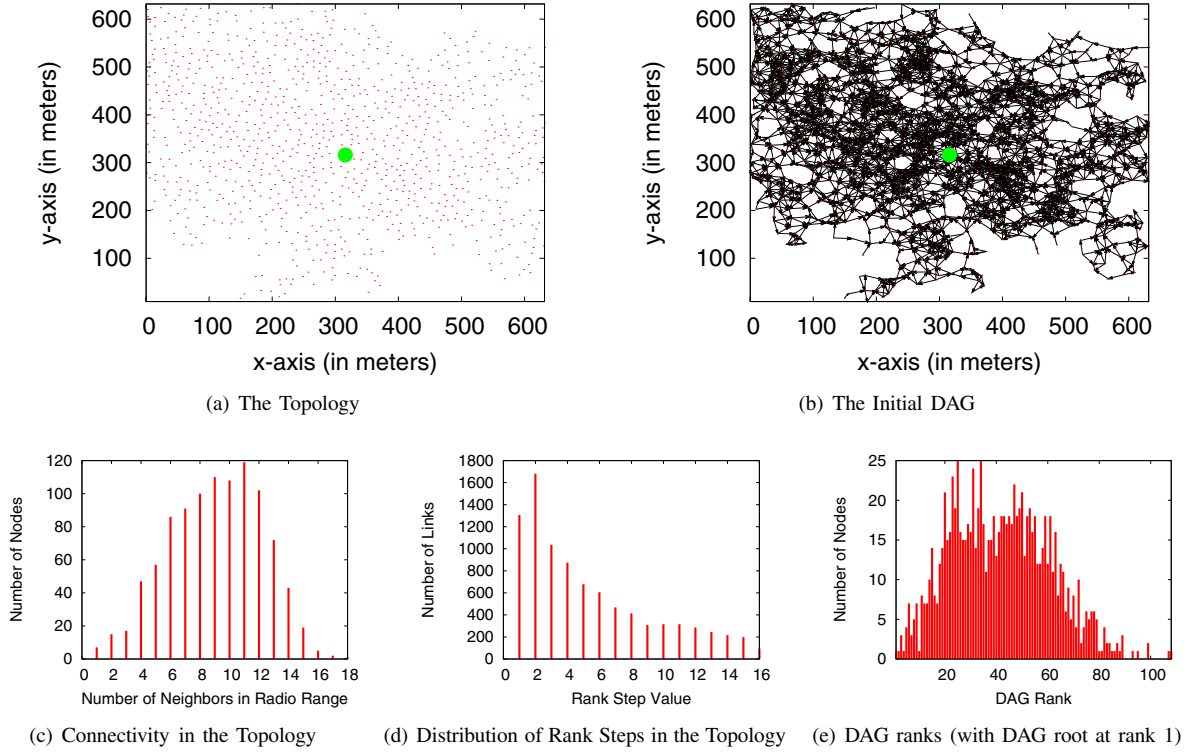


Figure 1. Characteristics of the 1000 Node Topology Used in Simulations

DAG structure that results when a node joins the DAG as soon as it hears the first neighbor DIO advertising the DAG and does not change its rank subsequently (adding as parents all the neighbors with lower rank than itself). Figure 1(e) shows the distribution of DAG ranks acquired by the nodes.

In order to study the behavior of routing loops, we did simulations where a particular node increases its DAG rank so that it can include additional neighbors as parents. This operation introduces one or more routing loops in the network if some of these new parents currently consider the node as an ancestor. In total, we did 1000 simulations: one for each non-root node in the topology. In each simulation, once the nodes have arranged themselves in the DAG structure shown in figure 1(b) and the DIO interval in each node has reached its maximum value I_{max} , we force one particular node (say node i) to increase its DAG rank so that it can include *all* its neighbors as parents. Thus, if $N(i)$ denote the set of neighbors for a node i , the new rank of the node is determined as follows:

$$R(i) = \min(R(j)+S(i,j) \mid R(j)+S(i,j) > R(k) \forall k \in N(i)), \forall j \in N(i)$$

where $R(i)$ denotes the rank of node i and $S(i, j)$ denotes the rank step value associated with link $i : j$. Note that the node does not increase its rank if it already considers all its neighbors as parents.

Node i increases its rank in the manner described above and includes all the neighbors that now have smaller rank

than itself as parents. The inclusion of additional parents introduces one or more routing loops if some of these new parents currently consider node i as an ancestor. These routing loops get resolved in the manner described below. In each simulation, we observe the time required for the routing loops to be resolved and the topology to stabilize (i.e. no more change in the rank or parent set of a node) following the rank increase operation. We also note the number of DIO messages generated by the nodes during this time.

When node i increases its rank and adds new parents, it also resets its DIO interval to I_{min} and restarts its trickle timer. Node i generates a new DIO advertising its new rank when the trickle timer expires. On receiving this DIO, a neighbor node j , that considers node i a parent, eliminates node i from its parent set if node i 's new rank is higher than node j 's rank and node j will have at least one parent left after eliminating node i as a parent. Such a move resolves the routing loop. If node j eliminates node i as a parent and node i happened to be its most preferred parent, node j selects another parent as the most preferred parent such that its new rank is just large enough to allow it to retain its remaining parent set. In other words, node j calculates its new rank in the following manner:

$$R(j) = \min(R(k)+S(j,k) \mid R(k)+S(j,k) > R(l) \forall l \in P(j)), \forall k \in P(j)$$

where R and S are defined as before and $P(j)$ is the set of

current parents for node j . Note that the new rank of node j may turn out to be smaller than its previous rank.

On the other hand, if node i is the only parent node j currently has, node j fails to resolve the loop since it can not remove the only parent it has. In this case, node j has no option but to increase its rank to $R(i)+S(j,i)$. This increase in rank may also make it possible for node j to include additional neighbors as parents (and possibly introduce more routing loops). The onus of resolving existing and new loops now lies on the descendants of node j . A change in rank or in the parent set causes node j to reset its DIO interval to I_{\min} , restart the trickle timer and generate new DIO at the expiry of the trickle timer.

Thus, a routing loop, caused by selection of a descendant neighbor as a parent, is resolved when some in-loop node eliminates its in-loop parent from the parent set. In case no node in the loop has an alternate parent, which happens only when the nodes in the loop are isolated from other nodes in the DAG, the loop persists and a *count to infinity* situation occurs, where all the nodes in the loop end up increasing their DAG ranks to 255 (the *infinity* value for 8-bit DAG rank) and thus detach from the DAG. In our simulations, since the in-loop nodes are not isolated from rest of the network, all routing loops get resolved and the *count to infinity* situation never occurs.

Figure 2 shows the simulation results. These simulations were performed with I_{\min} and I_{\max} values 2^7 ms (=128ms) and 2^{21} ms (\approx 35minutes) respectively. Also, in these simulations, a node always generates a new DIO at the firing of its trickle timer irrespective of the value of its redundancy counter.

As mentioned before, a simulation involves forcing a particular node to increase its rank once the initial DAG formation is over. Figure 2(a) shows the ranks of the nodes before (the initial rank) and after the rank-increase operation (ordered according to increasing initial rank). In 98 cases out of 1000 we simulated, the node failed to increase its rank since all its neighbors were already included in its parent set. In 879 cases, the node increasing its rank stablized at the increased rank. In the remaining 23 cases, the node stablized at a smaller rank because it chose a newly added parent as its most preferred parent and later had to remove this parent in order to resolve the routing loops. Selection of the new most preferred parent allowed the node to decrease its rank.

Figure 2(b) shows the *cumulative distribution function* (CDF) for the time required for the network to resolve loops and stablize (i.e., no more change in the rank or parent set of any node) following the rank-increase operation and figure 2(c) shows the CDF for the number of DIOs generated by the nodes during the stablization time. Clearly, in most of the cases (615 out of 902 cases where rank increase took place) the loop resolution is done within $I_{\min}/2$ to I_{\min} interval of the rank increase. This is the time required for the node increasing its rank, say node i , to generate its next

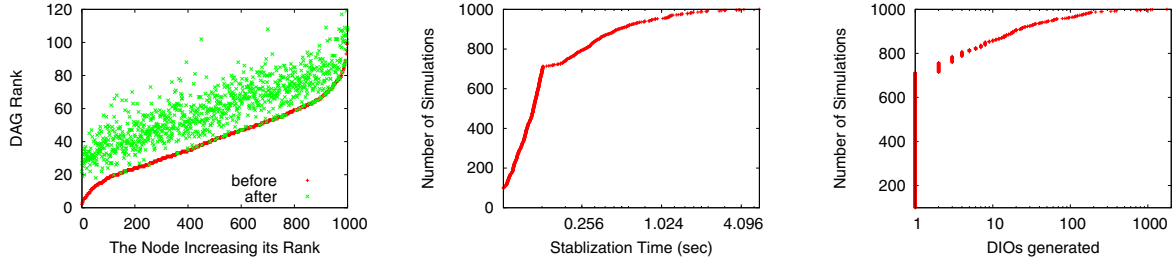
DIO. In these cases, node i chooses one or more children node as parents following the rank increase. These children node have alternate parents and hence are able to remove node i as parent as soon as they receive node i 's new DIO advertising a higher rank than themselves. In these cases, the loop resolution requires just one DIO (the one generated by node i following its rank increase) although the nodes that remove node i as a parent would also reset their DIO intervals and thus generate additional DIOs.

In 287 cases, more than one DIO is required for resolving routing loops. Figure 2(d) shows the ordered set of stablization times for these cases and the corresponding number of DIOs generated during the stablization time. There were 47 cases where the stablization times were greater than 1 second. Figure 2(e) shows the ordered set of these stablization times and corresponding number of DIOs generated. The highest observed value for the stablization time was 5.616s and the 1760 DIOs were generated by the nodes during this time. In this particular simulation, a node (with id 434) increased its rank from 16 to 36 adding 13 neighbors as new parents, 12 of which previously considered the node as a parent. Ten of these children nodes simply removed node 434 as a parent following the increase in its rank. Two children nodes had node 434 as their only parent and hence had to increase their rank to a value larger than node 434's new rank. This increase in rank allowed these nodes to add several new neighbors as parents. In total, the increase in rank of node 434 from 16 to 36 forced 176 nodes to alter their rank or parent sets. Figure 2(f) shows the ordered set of stablization times greater than 1 second and the corresponding number of *affected* nodes, i.e. the nodes that had to change their ranks or parent sets as a result of the rank increase operation.

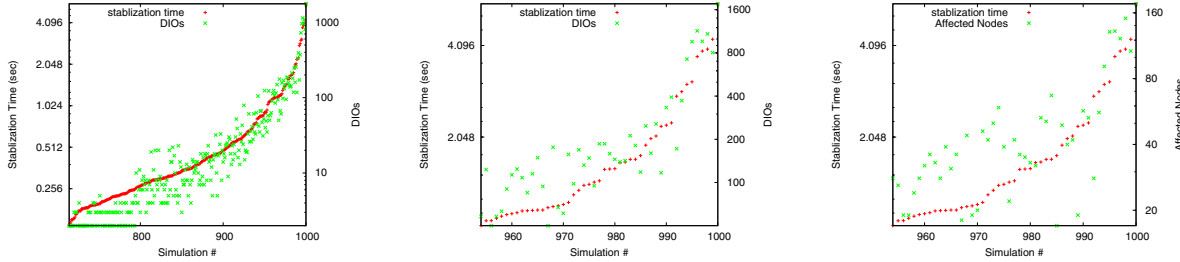
Overall, the simulation results suggest that, for this particular network topology, a vast majority of the routing loops resulting from the rank increase operations get resolved quickly. The subsequent stablization of the network along a new DAG structure is also quick and the overall process causes only a small number of DIOs to be generated. However, in some cases, the rank increase operation by one node triggers a sequence of events that generates multiple routing loops and causes a large number of nodes to be *affected*. In such cases, the network may require a non-negligible time to reconverge to a stable loop-free state and the affected nodes may produce a significant number of DIOs during this stablization time and afterwards.

III. MECHANISMS TO AVOID ROUTING LOOPS AND THEIR EFFECTIVENESS

Routing loops and the mechanisms to deal with them have been a major topic of discussion in ROLL working group at IETF. The decision to arrange nodes in one or more DAGs and forwarding packets only along DAG links is strongly motivated by a desire to prevent routing loops. But, as



(a) The rank of the node: before and after the increase (b) The CDF for the time required for the network to resolve loops and stabilize (c) The CDF for the number of DIOs generated during the stabilization time



(d) Ordered stabilization times and the corresponding number of DIOs (when more than 1 DIO was needed for loop resolution) (e) Ordered stabilization times (greater than 1s) and the corresponding number of DIOs generated (f) Ordered stabilization times (greater than 1s) and the corresponding number of affected nodes

Figure 2. Simulation results regarding the time required for network topology to stabilize when a node increases its rank to include additional parents and the number of DIOs generated during this time

discussed in the previous section, a rank-increase operation by a node in order to add new parents may break the acyclic structure of the DAG which may result in significant churn before the network stabilizes again along a new DAG. Opinions vary regarding how to deal with routing loops:

- Since most routing loops have a fleeting existence, do nothing.
- *Loop Prevention:* Do not allow a node to increase its rank until it receives a new sequence number in a DIO from a neighbor. The DAG root periodically updates the sequence number at which point the formation of a new DAG starts. When a node receives a new sequence number in the DIO of a neighbor, it can safely choose the neighbor as a parent since the neighbor, having received the new sequence number before the node, could not be the node's descendant in the DAG. Once the node has received DIOs bearing the new sequence number from a bunch of neighbors, it discards the DIOs carrying the old sequence number, chooses its new rank/parents and starts listing the new sequence number in its DIOs. We characterize this approach as *loop prevention* since routing loops are not possible under this approach.
- *Loop Detection:* As a packet travels from the source towards the destination, the en-route nodes store their *tags* in the packet's header. A routing loop is detected if a node finds its own tag in a received packet's header.

- *Loop Avoidance:* Whenever a node (say i) needs to increase its rank, it starts a wait timer and generates a new DIO advertising an *infinite* rank, thereby detaching itself from the DAG. As the children node receive this DIO, they either remove node i as a parent or detach from the DAG themselves and generate new DIOs within I_{\min} interval to advertise their new status. In this manner, the entire *sub-DAG* rooted at node i dismantles. Thus, following its detachment from the DAG, node i will receive new DIOs from neighbor nodes advertising their new ranks if these nodes previously considered node i as an ancestor. Thus, if the wait time is large enough, node i would receive the new ranks of the neighbor nodes that were its descendants earlier while the wait timer is still running and, at the expiry of the wait time, choose its new rank correctly without creating any routing loops. We characterize this approach as *loop avoidance* since the routing loops may still occur if the neighbors, that were previously in the node's *sub-DAG*, are not able to detach from the *sub-DAG* by the time the node's wait time is over. This may happen if the DIOs get lost or the wait time is not large enough.

There may be strong reasons to prefer one approach over the others in the context of a particular application. However, there are costs involved with each approach. The *prevention* approach requires a node to wait for a sequence number

update by the DAG root before it can choose new parents that require it to increase its rank. The *detection* approach requires carrying the node tags in the packet thereby reducing the already small space available for carrying the data.² The *avoidance* approach requires dismantling the sub-DAG rooted at the node performing the rank increase, which may be too steep a price to pay if the routing loop would have been resolved quickly with a minor change in DAG structure otherwise.

In this paper, we study the effectiveness of the *loop avoidance* approach. Specifically, we compare via simulations the stablization time and DIOs generated following a rank increase operation by a node with and without the loop avoidance. The simulation results with no loop avoidance were reported in the previous section. In this section, we compare those results with the corresponding simulation results with loop avoidance in effect. The loop avoidance scheme used in these simulations behaved in the manner described above and was enforced whenever a node needed to increase its rank, e.g. when the only parent the node has advertises a higher rank than before. The wait time used in the loop avoidance scheme was calculated as $(\text{rand}(0,1) + 3) \times I_{\min}$. This wait time value is based on the assumption that, in most cases, two radio-range neighbors would be at most three DAG hops away from each other. Thus, a wait time between 3 and 4 times I_{\min} should generally be sufficient for neighbors to detach from the sub-DAG of the node planning to increase its rank. When the wait time is over, the node chooses its new rank such that it is just large enough to allow the node to select as parents all its neighbors that still advertise a *finite* rank.³ Additionally, it resets its trickle timer and generates a new DIO advertising its new rank. As before, the I_{\min} value used in these simulations was 128ms. The rest of the simulation setup was also same as that described in the previous section. In particular, we ensured that the network converges to the same DAG structure initially (before the rank increase by a node) in both sets of simulations.

Figure 3 shows the simulation results comparing the performance with and without loop avoidance. Figure 3(a) compares the ranks of the nodes doing the rank increase *after* the rank increase operation with and without loop avoidance in effect. The ranks of these nodes *before* the rank increase operation were same. With loop avoidance in effect, the rank increase operation makes the node advertise an infinite rank and start a wait timer. The node chooses a new rank when the wait timer fires. In 52 cases out of 1000 we simulated with loop avoidance in effect, the node acquired the same rank after the firing of the wait timer as before. This happened because the node already had selected

all its neighbors as parents and its initial rank was already just large enough to include all the neighbors as parents. In 34 cases, the new rank of the node was smaller than before. Again, in these cases, the node already had all its neighbors as parents; however its initial rank was more than what was required to include all neighbors as parents.⁴ In general, as figure 3(a) shows, the rank increase with loop avoidance in effect was somewhat less than the rank increase without loop avoidance. This happened because, under the loop avoidance scheme, the detachment of the node doing the rank increase forces its children, that consider it their most preferred parent, to recalculate their ranks to the minimum required to keep all the remaining parents. Thus, in general, the detachment of the node forces its children to reduce their ranks before the node chooses its new rank, which is not the case when loop avoidance is not being used.

Figures 3(d) and 3(g) show the stablization times without loop avoidance (arranged in increasing order) and the corresponding stablization times when the loop avoidance is in effect. The minimum stablization time under loop avoidance is $3 \times I_{\min}$ (i.e. 384ms), which is much larger than the corresponding stablization time when loop avoidance is not being used. In 912 cases out of 1000 we simulated, the stablization times under loop avoidance scheme were between $3 \times I_{\min}$ (384ms) and $4 \times I_{\min}$ (512ms). Figure 3(g) zooms in on figure 3(d) focussing on the large stablization times. This figure indicates that the stablization times under loop avoidance are generally smaller but could be significantly higher in some cases. Thus, there is no clear advantage of using loop avoidance in terms of reducing the stablization times. Figure 3(b) shows the CDF for the stablization times with and without loop avoidance.

Figure 3(e) shows the number of DIOs generated during the stablization time without loop avoidance (arranged in increasing order) and the corresponding numbers when the loop avoidance is being used. Figure 3(c) shows the CDFs for the number of DIOs generated during the stablization times with and without loop avoidance. Note that, in cases where only a small number of DIOs are generated during stablization times without loop avoidance, the corresponding number of DIOs generated with loop avoidance in effect could be much larger. This is simply an artifact of larger (in range $[3 \times I_{\min}, 4 \times I_{\min}]$) stablization time associated with loop avoidance. Smaller stablization times without loop avoidance mean that the DIOs generated by the children nodes, on receiving the DIO of the node doing the rank increase, do not fall within the stablization time. Higher stablization times under loop avoidance allow these DIOs to fall within the stablization time.

²Popular IEEE 802.15.4 MAC/PHY protocol allows a maximum packet size of 133 bytes including the PHY/MAC/IP headers [10].

³A *finite* rank neighbor is not selected as a parent if it requires the node to acquire an *infinite* (i.e. 255) rank.

⁴Note that, during the initial DAG formation, the node selects its rank as soon as it receives the first DIO about the DAG and it does not change its rank unless it is selected to perform the rank increase operation or its most preferred parent advertises an infinite rank.

The most interesting result from these simulations is revealed when we consider how the use of loop avoidance affects the number of DIOs generated during the stabilization times when this number is large. Figure 3(h) zooms in on figure 3(e) focussing on the cases where the number of DIOs generated during stabilization times are large. It is clear that, in most such cases, the use of loop avoidance results in the generation of a larger number of DIOs during the stabilization times. This observation is significant because this increase in the number of DIOs generated is not because of larger stabilization times under loop avoidance. From figure 3(g), we know that in such cases the stabilization times under loop avoidance are generally smaller. This increase in the number of generated DIOs can be attributed to the larger number of *affected*⁵ nodes under loop avoidance. Figures 3(f) and 3(i) show the number of affected nodes without loop avoidance (arranged in increasing order) and the corresponding number of affected nodes with loop avoidance in effect. Clearly, the number of affected nodes under loop avoidance are almost always greater (some times significantly) than or same as the number of affected nodes without loop avoidance. Apparently, the increase in the number of affected nodes due to the dismantling of the sub-DAG, rooted at the node doing the rank increase operation, undoes the decrease in the number of affected nodes due to loop avoidance. The larger number of affected nodes means a larger number of DIOs generated under loop avoidance following the rank increase operation.

Overall, it appears that the dismantling of the sub-DAG, rooted at the node doing the rank increase, causes more turmoil in the network than the routing loops themselves and the use of loop avoidance is not necessarily beneficial in the maintenance of *directed acyclic graphs* in low power and lossy networks.

IV. CONCLUSION

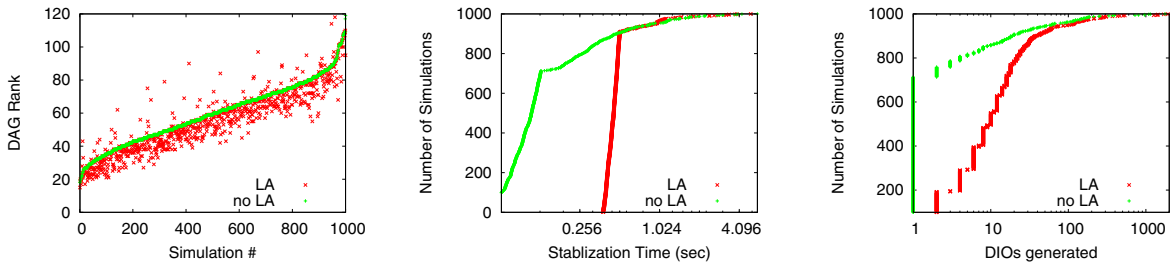
The ROLL working group in IETF is currently engaged in the development of a routing protocol for large scale low-power and lossy networks based on organizing the network topology along one or more *directed acyclic graphs* (DAGs) rooted at popular/default destinations. Routing loops and the mechanisms to deal with them (*prevention, avoidance and detection*) have been a major topic of discussion within the working group. A DAG maintains its loop-free nature by requiring that each DAG node must have a higher 'rank' than any of its DAG parents. While a node can decrease its DAG rank safely without creating a routing loop, increasing its DAG rank to add a new parent may result in a routing loop if the new parent is also a descendant in the DAG. In this paper, we first reported simulation results regarding the time required by a representative network topology to

converge to a stable, loop-free state (and the number of routing messages generated during this time) following a rank increase operation if there are no restrictions on the nodes increasing their ranks. Then, we reported results of simulations where the nodes use a *loop avoidance* strategy that requires a node to dismantle the sub-DAG rooted at itself before increasing its rank and compared the two sets of results. This comparison revealed that the turmoil caused by dismantling of the sub-DAGs in order to increase ranks may be much more than what the routing loops themselves will cause. Consequently, the use of such loop avoidance mechanism in the operation of a DAG based routing protocol can not be universally recommended.

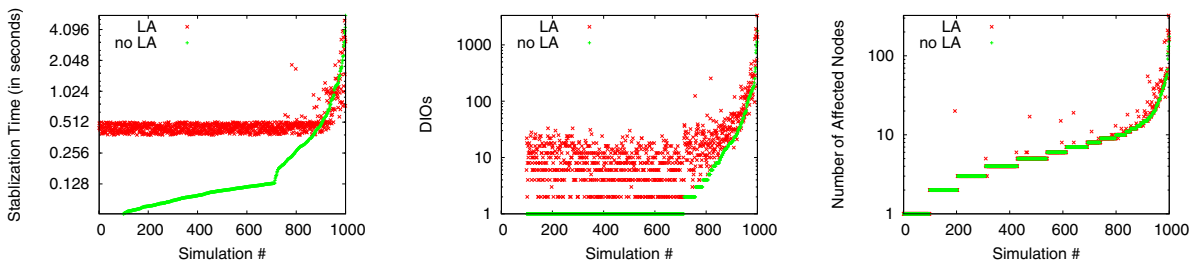
REFERENCES

- [1] K. Pister, P. Thubert, S. Dwars, and T. Phinney, "Industrial Routing Requirements in Low-Power and Lossy Networks," IETF, Request For Comments 5673, Oct. 2009.
- [2] M. Dohler, T. Watteyne, T. Winter, and D. Barthel, "Routing Requirements for Urban Low-Power and Lossy Networks," IETF, Request For Comments 5548, May 2009.
- [3] J. Martocci, P. DeMil, W. Vermeylen, and N. Riou, "Building Automation Routing Requirements in Low Power and Lossy Networks," IETF, Internet-Draft draft-ietf-roll-building-routing-reqs-07, Sep. 2009.
- [4] A. Brandt and G. Porcu, "Home Automation Routing Requirements in Low Power and Lossy Networks," IETF, Internet-Draft draft-ietf-roll-home-routing-reqs-08, Sep. 2009.
- [5] P. Levis, A. Tavakoli, and S. Dawson-Haggerty, "Overview of Existing Routing Protocols for Low Power and Lossy Networks," Internet Engineering Task Force, Internet-Draft draft-ietf-roll-protocols-survey-07, Apr. 2009, work in progress.
- [6] T. Winter and P. Thubert, "RPL: IPv6 Routing Protocol for Low Power and Lossy Networks," IETF, Internet-Draft draft-ietf-roll-rpl-04, Oct. 2009.
- [7] P. Levis, E. Brewer, D. Culler, D. Gay, S. Madden, N. Patel, J. Polastre, S. Shenker, R. Szewczyk, and A. Woo, "The emergence of a networking primitive in wireless sensor networks," *Communications of the ACM*, vol. 51, no. 7, pp. 99–106, Jul. 2008.
- [8] S. McCanne and S. Floyd, "ns network simulator." [Online]. Available: <http://www.isi.edu/nsnam/ns/>
- [9] M. Goyal, "Zigbee/IEEE 802.15.4 module for NS2 simulator," 2008. [Online]. Available: <http://www.cs.uwm.edu/~mukul/wpan.html>
- [10] "Part 15.4: Wireless MAC and PHY layer specifications for low-rate wireless personal area networks," *IEEE Std 802.15.4-2006*, 2006.

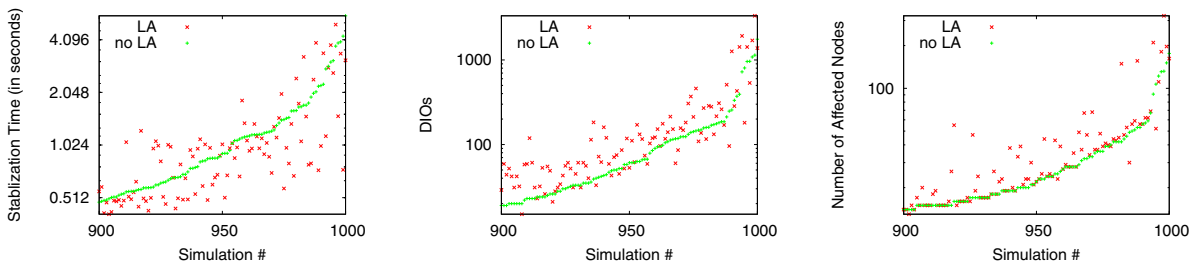
⁵A node is said to be *affected* by a rank increase operation if the operation causes the node to change its rank or the parent set.



(a) The rank of the node after the increase with and without loop avoidance (LA) in increasing order of the rank without loop avoidance (b) The CDF for the time required for the network to stabilize with and without loop avoidance (LA) (c) The CDF for the number of DIOs generated during the stabilization time with and without loop avoidance (LA)



(d) The stabilization times with and without loop avoidance (LA) in increasing order of the stabilization time without loop avoidance (e) The DIOs generated during the stabilization time with and without loop avoidance (LA) in increasing order of the DIOs generated without loop avoidance (f) The number of affected nodes following the rank increase operation with and without loop avoidance (LA) in increasing order of the number of affected nodes without loop avoidance



(g) Zooming in on Figure 3(d) (h) Zooming in on Figure 3(e) (i) Zooming in on Figure 3(f)

Figure 3. Simulation results comparing the performance with and without loop avoidance when a node increases its rank to include additional parents