A Study of RPL DODAG Version Attacks

Anthéa Mayzaud[1], Anuj Sehgal[2], Rémi Badonnel[1],
Isabelle Chrisment[1], and Jürgen Schönwälder[2]

[1] TELECOM Nancy, Université de Lorraine, LORIA UMR 7503,
Inria Nancy-Grand Est, Villers-lès-Nancy, 54600 France
{anthea.mayzaud,remi.badonnel}@inria.fr, isabelle.chrisment@loria.fr
[2] Computer Science, Jacobs University Bremen,
Campus Ring 1, 28759 Bremen, Germany
{s.anuj,j.schoenwaelder}@jacobs-university.de

Abstract. The IETF designed the Routing Protocol for Low power and Lossy Networks (RPL) as a candidate for use in constrained networks. Keeping in mind the different requirements of such networks, the protocol was designed to support multiple routing topologies, called DODAGs, constructed using different objective functions, so as to optimize routing based on divergent metrics. A DODAG versioning system is incorporated into RPL in order to ensure that the topology does not become stale and that loops are not formed over time. However, an attacker can exploit this versioning system to gain an advantage in the topology and also acquire children that would be forced to route packets via this node. In this paper we present a study of possible attacks that exploit the DODAG version system. The impact on overhead, delivery ratio, end-to-end delay, rank inconsistencies and loops is studied.

1 Introduction

The emergence of low-cost low-power resource constrained nodes capable of wireless networking enables new applications ranging from a smart electricity grid to mobile health solutions. Such constrained network devices could be integrated with the existing Internet infrastructure, so that they may utilize software services already available coupled with the nodes control and data-gathering capabilities.

Lacking any form of multi-hop routing, most networks of such resource constrained devices rely on a router being present on the local link. The Routing Protocol for Low-power Lossy Networks (RPL) was designed by the IETF [1] to fit within resource constraints of embedded devices. RPL not only forms loop-free routing topologies but also optimizes these for application specific goals, e.g. energy conservation, by using metrics available to a device. However, the features that enable RPL to provide this level of flexibility could also be manipulated by malicious nodes to cause harm to the network.

Some of these attacks can be mitigated by using the security mechanisms proposed by RPL, however, important details like key-management are left-out by the current standard [2] and cryptographic algorithms are not only complex

A. Sperotto et al. (Eds.): AIMS 2014, LNCS 8508, pp. 92–104, 2014.

to implement unless there is dedicated cryptographic hardware support, but also consume significant resources on constrained devices [3]. An evaluation of several popular RPL implementations[1] reveals that none of them support secure RPL operation. This makes attacks on RPL quite a possibility.

One attack that can misuse an internal RPL process, normally used for ensuring a loop and error free topology, is the version number attack. A malicious node modifies the version number associated with a topology, thereby forcing a rebuild of the entire routing tree. Since the version number is included in control messages by parents, there is no established mechanism for ensuring the integrity of the reported version number. A forced rebuild can cause increased overhead, depletion of energy reserves, channel availability issues and even loops in the routing topology. An attacker can also become the parent of its neighbors, thereby snooping on or modifying packets routed via itself. This paper studies the effects that a version number attack can have upon an RPL network.

The paper is organized as follows. An overview of related work is presented in Section 2, followed by an overview of the RPL protocol in Section 3. The version number attack is detailed in Section 4 and the experimental setup used for evaluating the impact of this attack is provided in Section 5. Results of the study are discussed in Section 6 before drawing conclusions in Section 7.

2 Related Work

The IETF RoLL working group performed a security threat analysis for RPL [8]. This study identified potential security issues in RPL networks and proposed countermeasures to address them. The identified threats were partly classified into four categories: (1) authentication, (2) confidentiality, (3) integrity and (4) availability. The version number attack considered here could be categorized as an integrity threat, i.e. an unauthorized modification attack. The IETF security threat study does not, however, provide an assessment of the seriousness of such attacks, which is the goal of this study.

In recent years, several studies on attacks in RPL networks have also been performed. For instance, authors of [9] investigated consequences of black-hole attacks in RPL networks and how they can be detected using specific measurable metrics. Others works such as [10] or [11] explored several attacks against RPL, including sink-hole and worm-hole. However, these are attacks that could be addressed by enabling the secure mode of RPL operation as well.

Version number attacks have also been suggested previously [12,13], but their effects have not been analyzed to understand whether it would be prudent to mitigate such attacks or not. The Version Number and Rank Authentication (VeRA) [13] approach provides integrity of version numbers and ranks advertised in control messages via hash and signature operations. Their approach is not only shown to be faulty by the authors of [14], but another mechanism called TRAIL that uses the root as a trust anchor and monotonically increases node

[1] Contiki 2.6 [4], TinyOS 2.1.2 [5], RIOT 2013.08 [6] and SimpleRPL 1.0 [7] for Linux were evaluated during the course of this study.

ranks is also proposed by them. Considering that both approaches require maintaining state information that is likely to diminish already constrained computing resources, it is interesting that the impact of version number attacks on RPL networks was not analyzed before. As such, this study aims to quantify the consequences a version number attack can have on a network.

3 The RPL Protocol

RPL forms loop-free topologies termed Destination Oriented Directed Acyclic Graphs (DODAGs), which organize nodes into a hierarchical structure of a single root, children and further descendants. Objective functions are used by RPL to optimize the topology based on predefined goals, e.g. energy consumption, hop-count or link quality. Multiple instances of RPL, each being an execution of RPL with a specific objective function, can be run within a network, each with its own DODAGs [1]. While a node may be a member of multiple instances, it can only join a single DODAG in an instance at any point in time.

An RPL DODAG is created and maintained by the use of the following control messages: (1) DODAG Information Solicitation (DIS), (2) DODAG Information Object (DIO) and (3) Destination Advertisement Object (DAO). A node wishing to join a network begins by broadcasting DIS messages. These solicit DIO messages, which contain information, like node ID and objective code point, about the DODAG. Since DIO messages are also periodically broadcast, a node may choose to just wait till it receives one from its neighborhood. The rate at which these DIO messages are broadcast is determined by the trickle algorithm [15]. Essentially, the longer a DODAG has been stable, the fewer the number of DIO transmissions.

A node determines its rank in the DODAG by calculating it from the objective code point specified in a received DIO message. If multiple DIO messages from several neighbors are received, then the neighbor that provides the best rank is chosen as the parent. This approach forms upwards routes, i.e. towards the root. The DAO message, which contains all routable prefixes, is sent up the tree to form downwards routes [1]. Each node receiving the DAO message aggregates the prefixes and propagates it further upwards, thereby making downwards routes available to parents.

Loops are avoided by ignoring messages, which arrived from a descendant, traveling downwards. Furthermore, nodes are normally only permitted to change their parents in case of improvements in rank. Any movement in the topology that results in worse ranks is prohibited, unless it occurs during a loop avoidance measure or when the root creates a new version. Even with built-in methods to avoid loops, it is possible for a loop or rank inconsistencies to occur in some situations. There are a few methods provided by RPL to rectify such situations.

The *data path validation* mechanism is used to detect rank related inconsistencies. Special flags, to indicate expected direction of the packet (*'O' flag*) and rank errors detected while forwarding (*'R' flag*), are carried in the RPL IPv6 header options [16] of multi-hop data packets. A rank inconsistency

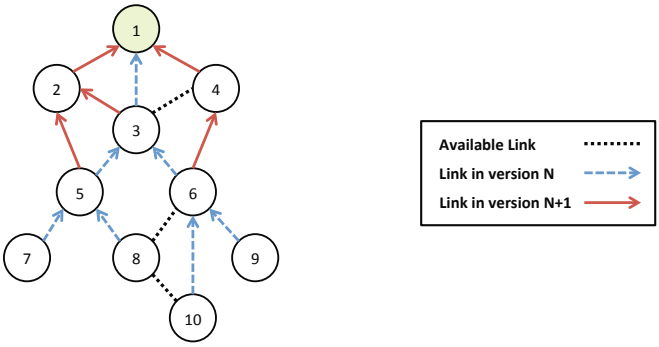

Fig. 1. New DODAG iteration. In red solid arrow, the new DODAG is being built, in blue dashed arrow the old topology is shown. Since the blue graph is an older version of the DODAG, nodes in the new version cannot send packets over blue links. But, packets can be sent from old to new iteration, i.e. from blue to red.

occurs when there is a mismatch in the direction indicated by the 'O' flag [1]. The 'R' flag is set on such a packet and it is forwarded. A loop is detected when a packet with an inappropriate 'O' flag and the 'R' flag enabled is received. This packet is dropped and the trickle timer used for controlling DIO transmissions is reset [15].

Nodes disappearing from a network due to lack of battery power or poor link conditions is more likely to occur in low-power lossy networks than loops in an RPL network [17]. To deal with such situations, RPL provides the global and local repair mechanisms. When the *global repair* mechanism is used, it initiates a rebuild of the entire DODAG by incrementing the version number of the DODAG [1]. The version number is carried in the DIO message; each receiving node compares its existing version number against the one received from its parent and in case the received version is higher, it must ignore its current rank information, reset trickle timers and initiate a new procedure to join the DODAG. Amounting to a reconstruction of a new DODAG, this global repair mechanism guarantees a loop free topology, however, it is also quite costly.

To avoid rebuilding the entire DODAG when a parent node disappears, two *local repair* mechanisms are also provided [1]. The first allows nodes to temporarily route through neighbors of the same rank, while the other approach is to switch parents. These approaches may be used in combination as well to avoid any loss of connectivity.

4 Version Number Attack

The version number is used by the root to control the global repair process of RPL and to ensure that all nodes in the DODAG are up-to-date with the routing state. Every DIO message carries the version number so that in case receiving nodes are part of an outdated DODAG version, they can join the new DODAG by recalculating their rank and then updating their stored version number.

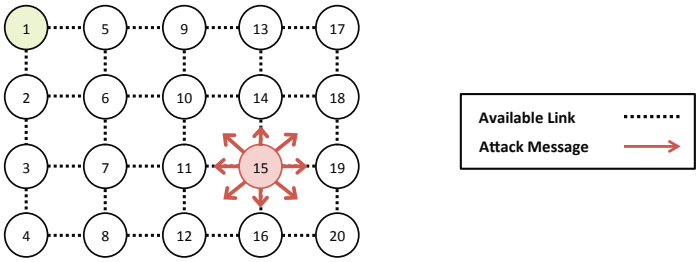

Fig. 2. The grid topology used for performing experimental evaluation of the version number attack. The location of the attacker is varied from nodes 2 to 20.

An older value of the version advertised in DIO messages indicates that the node did not migrate to the new version of the DODAG. Such a node should not be considered a preferred parent by other nodes. While the global rebuild process is ongoing, it is possible for two versions of a DODAG to temporarily coexist. To avoid loops data packets are permitted to transit from the old version to the new one but not the other way, as shown in Figure 1. This is because the old version is no longer a DAG and loop free topologies cannot be guaranteed in this situation.

To avoid possible inconsistencies in the network, the version number should be propagated unchanged through the DODAG. However, there is no mechanism in RPL to check if the integrity of the version number is maintained in received DIO messages. A malicious node may change this field in its own DIO messages to harm the network. Nodes receiving a malicious DIO, with a new version number, will reset their own trickle timer, update the version in their own records and advertise this new version through DIO messages to their neighborhood as well. This can cause the illegitimate version number to propagate through the network.

Such manipulation of the version number in the DIO packets does not only cause an unnecessary rebuild of the whole DODAG but it also generates loops in the topology. This can negatively impact energy reserves of the nodes, routing of data packets and channel availability.

5 Experimental Setup

The Contiki 2.6 operating system [4] was used to perform an evaluation of the version number attack. Contiki is appropriate for this study because not only does it provide an RPL implementation that works on multiple platforms, but also allows the emulation of some common platforms within the Cooja simulator [18].

The default RPL implementation of Contiki does not support the complete data path validation mechanism. While it uses ‘O’ flags to detect possible rank inconsistencies, it does not set the ‘R’ flag or drop packets with this flag. Furthermore, the trickle timers are also not reset the way they should be. Since this mechanism is used to repair inconsistencies and loops, both of which are

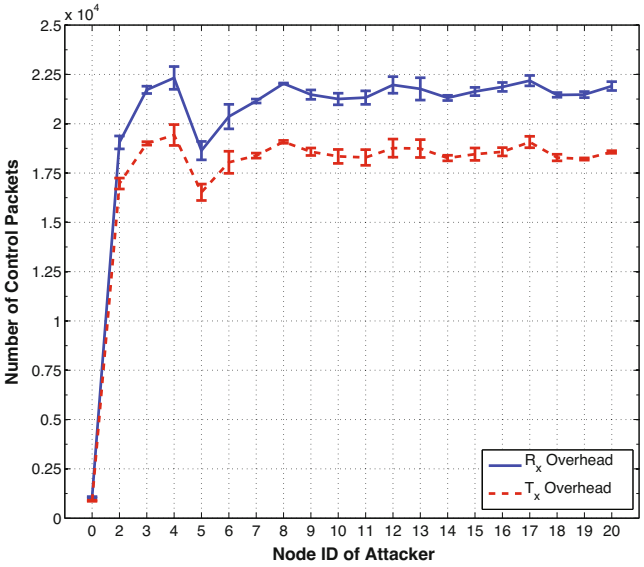

Fig. 3. The incoming and outgoing control packet (DIS, DIO, DAO) overhead for every location of the attacker. (Node ID 0 indicates no attacker; the error bars show the standard deviation between the five simulation runs.)

expected to be created by the version number attack, the Contiki implementation was modified to include it. Another motivation behind this was to study the impact of the version attack on RPL while any mitigation approaches described by the standard are functioning.

The TelosB mote with the ContikiMAC RDC was used as the development platform since its resources, while limited, are enough for the Contiki RPL implementation to function. Rather than building a topology of actual nodes the TelosB binary was used in Cooja, which uses MSPSim to emulate the MSP430 architecture and the performance of a MSP430F1611 microcontroller used by the TelosB. This allows us to test a relatively large topology size under multiple scenarios. The effect of the lossy IEEE 802.15.4 is also minimized allowing us to study the behavior of RPL, rather than its interaction with the channel.

A grid topology of 20 nodes using the UDGM radio model [18], shown in Figure 2, was setup for all experiments in Cooja. Across all experiments, node 1 is the DODAG root and also the sink to which all other nodes send messages every twenty seconds. The attacker is designed to constantly send incorrect version numbers, which are greater than the root's by one. This scenario is adopted because it allows relocation of the attacker to multiple positions easily, making it possible to study the consequences of the attack from different locations and neighborhood scenarios within a network. A random back-off of up to six seconds is also added to this periodic transmission time on all nodes so that packet collisions are avoided when possible. The nodes are placed at a regular distance

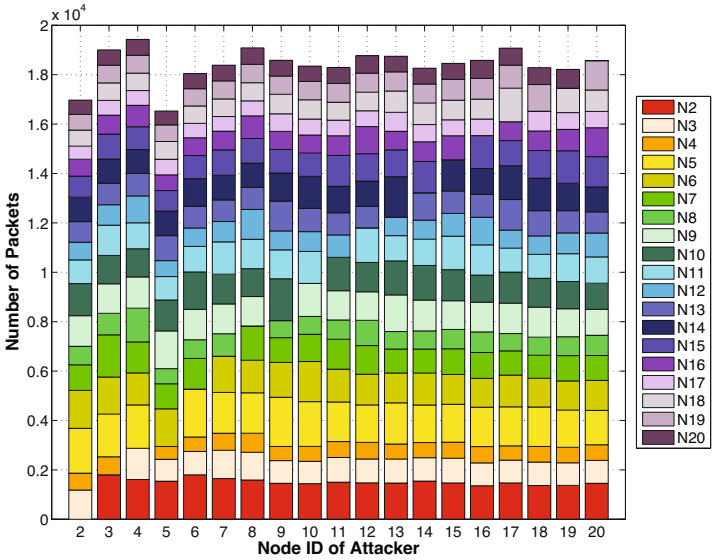

Fig. 4. The per node outgoing packet overhead (DIS, DIO, DAO), for every location of the attacker

of 30m from their vertical and horizontal neighbors. The transmission strength is set such that packets are received successfully by nodes within a 30m radius and the signal causes interference with other nodes for a radius of 60m. This ensures that every node only has vertical and horizontal neighbors reachable during the simulation, thereby adding predictability and ease of analysis to the results.

Each simulation lasts for a lifetime of fifty minutes. One simulation was executed without any attacker in the network so as to obtain a baseline for comparisons against. Further nineteen simulations are also run, with the location of the attacker being fixed to one of nodes 2 to 20, such that at least one simulation with the attacker located at every node between 2 and 20 is executed. Moving the position of the attacker in the network allows us to study the impact that the position of an attacker and the size of neighborhood have upon the behavior of the RPL network. This entire set of twenty simulations is repeated five times to obtain some statistical significance in order to ensure dependability. Attacks start after five minutes of simulation time, so that the network has enough time to settle and a stable RPL topology emerges.

The following metrics are used to perform this study: (1) *Packet overhead*, which is the total number of RPL control packets, i.e. DIS, DIO and DAO message, transmitted (outgoing overhead) and received (incoming overhead) in the network. As such, in the no attacker scenario these are the messages necessary to form and maintain an RPL DODAG. (2) *Delivery ratio*, which is the number of data packets successfully delivered to the sink (node 1) compared to the total number of data packets generated by all nodes in the network. (3) *Average*

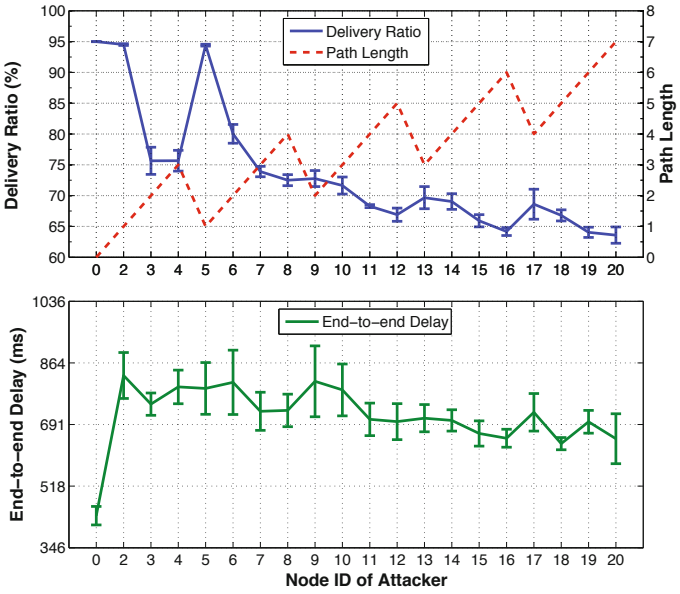

Fig. 5. The total delivery ratio and end-to-end delay for every location of the attacker. Path length, i.e. the hop count, to the attacker from root is also shown. (Node ID 0 indicates no attacker; the error bars show the standard deviation between the five simulation runs.)

end-to-end delay, which is the average amount of time it takes for all packets, from every node in the network, to be successfully delivered to the sink. Lost and dropped packets are not considered in this calculation. (4) *Inconsistencies*, which are the number of packets detected by a node that are destined for a descendant but also arrive from a child or vice versa. (5) *Loops*, which are the number of packets detected by a node that not only indicated an inconsistency but also have the 'R' flag enabled, i.e. a possible loop was previously detected on this path.

6 Results

The average incoming and outgoing packet overhead experienced by the entire network, for each location of the attacker, is shown in Figure 3. The incoming and outgoing overhead when there is no attacker (attacker ID 0 in the figure), both are about 1250 packets, which can be considered reasonable for a network of 20 nodes that functions for 50 minutes. However, as soon as an attacker is introduced, the overhead can increase by up to 18 times in the network. At first glance it appears that the overhead increases as the attacker moves into regions where it has more neighbors. Closer inspection of Figure 3 reveals that mostly nodes in the bottom row of the topology (4, 8, 12, 16 and 20) produce

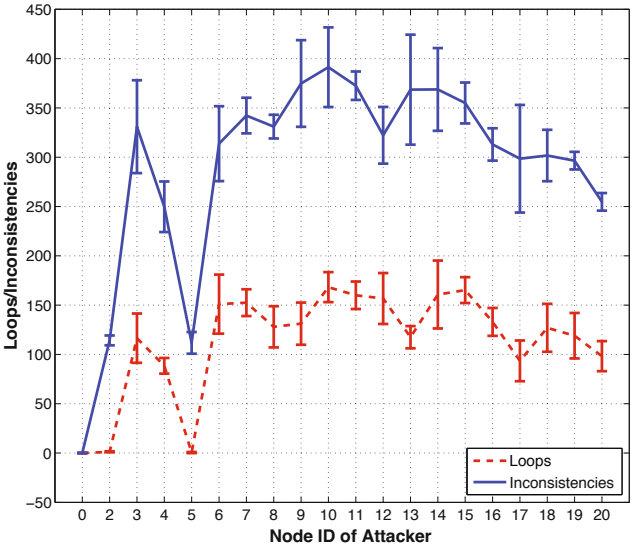

Fig. 6. The total number of loops and inconsistencies in the network, for every location of the attacker. (Node ID 0 indicates no attacker; the error bars show the standard deviation between the five simulation runs.)

localized maximums in their column of the topology. Since each of these nodes end up towards the tail-end of their section of the DODAG, it also implies that the further away a node is from the root, the more damage it can do because this provides it an opportunity to spread the damage further. The location of nodes 2 and 5 produces a topology that is analogous to each other, thereby leading to results that are similar in both their cases. As such, not only the number of neighbors, but also the distance from the root impacts the level of increase in the overhead.

Since the position of the attacker can impact the overhead, it is interesting to also investigate which nodes a particular position of the attacker can effect. Only the per node outgoing packet overhead is plotted in Figure 4, because the incoming and outgoing packet overhead is closely related. While it is intuitive to assume that the largest increase in overhead would be contributed by nodes neighboring the attacker, because these are most likely to form loops, the results from Figure 4 indicate otherwise. The version number attack, by design, is propagated across all neighbors, even if they are not relatives of the current node. This causes a significant increase in control packets to cascade all across the network, leading to the observed results. As such, a version number attack is worse than many others because it does not only impact the attacker's neighborhood but also the entire network.

This increased overhead decreases channel availability, thereby impacting delivery ratio. The delivery ratio, averaged over five runs, for the entire network,

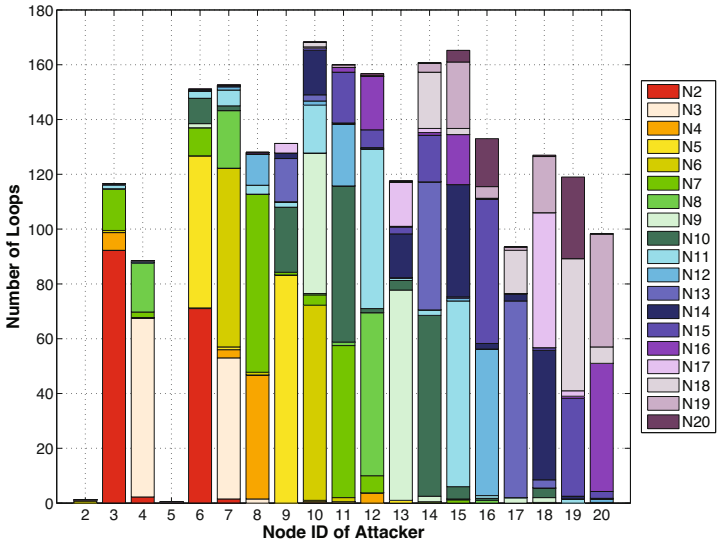

Fig. 7. The total number of loops detected per node, for every location of the attacker

with respect to the location of the attacker, is shown in Figure 5. It is immediately apparent that the version number attack can have a significant impact on delivery ratio, with it being reduced by up to about 30%. More interestingly, a similar pattern can be seen in the effect on delivery ratio, as was on the overhead. As such, the path length of the location of the attacker is also shown in Figure 5. This makes it clear that there is a strong correlation between the distance of the attacker from the root and the effect of the version number attack on delivery ratio. A good example of this is that the delivery ratio when the attacker is located on nodes 2 and 5 is exactly the same, and they both also have a path length of 1 to the root. In fact, the correlation between the delivery ratio and path length can be seen across all positions of the attacker, with the attacker located at the bottom of the topology in Figure 2, i.e. farthest from the root, leading to worst impact on delivery ratio.

The end-to-end delay is also a good measure of a network's performance. The average end-to-end delay for different attacker locations can also be seen in Figure 5. Any packet that was not delivered did not contribute towards calculation of the end-to-end delay. As with other metrics, it is obvious that an attack significantly impacts end-to-end delay, by almost doubling it as against no attack within the network. Unlike overhead and delivery ratio, there is no strong correlation between location of the attacker and the delay. This is because the delay is effected by a number of things, such as the channel availability, number of loops, possible alternate routes, neighborhood density, etc. The large number of factors that can influence this also cause a high variation in the delay results across the set of simulations, as is also evident from Figure 5.

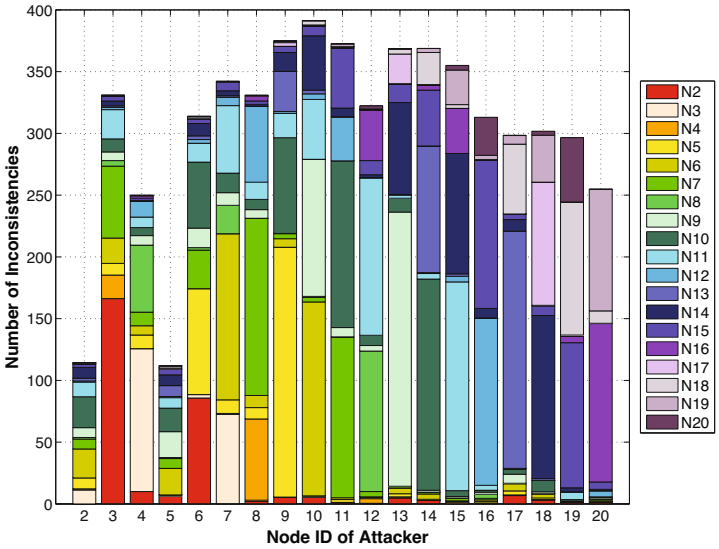

Fig. 8. The total number of inconsistencies detected per node, for every location of the attacker

Since the version number attack creates loops (packets encountered with the 'R' flag) and rank inconsistencies (packets that mismatch actual direction and have 'O' flag set) in the network, it is important to also understand how these are effected. The number of such inconsistencies the network can be seen in Figure 6. The pattern of rank inconsistencies and loops are closely related because loops are a form of rank inconsistency and also counted into it. Unlike before, attacker locations farthest from the root generally lead to the least number of inconsistencies and loops in the network. On the other hand, attacker locations closest to the root, but with most amount of neighbors lead to the highest number of inconsistencies and loops. For example, nodes 3 and 6 create more inconsistencies than nodes 2 and 5, which are closer to the root node. This is because both nodes 2 and 5 have fewer neighbors than nodes 3 and 6. As such, the number of loops are closely related to the number of neighbors an attacker has and increases with proximity to the root. Closer proximity to the root likely has this behavior because it forces a rebuild from the root faster as well, causing this to cascade into the rest of the DODAG before a new attack cycle can begin.

This interesting relationship of inconsistencies and loops with the location of the attacker means that it would be useful to understand where in the network most of this effect is centered. As such the number of loops and inconsistencies per node are plotted in Figures 7 and 8 respectively. It is immediately apparent from these plots that as the location of the attacker shifts towards higher node IDs, the loops and inconsistencies also shift the area of impact towards them. Closer analysis reveals that while there might be some loops created in nodes that are farther away from the attacker, the majority

of them are located within the direct neighborhood. In fact, the bulk of these loops and inconsistencies are detected by the parent, and alternate parents, of the attackers. This is because most of the packets will be routed towards the preferred parent, or the alternate parent in case of the preferred parent being unavailable. The next highest quantity of loops and inconsistencies is detected at the children of the attackers. For example, when the attacker is located at node 11, the highest number of loops and inconsistencies are detected by parent nodes 7 and 10. The children, nodes 12 and 15, account for the majority of the rest of these anomalous situations.

7 Conclusions

A study was performed to quantify the effects of version number manipulation attacks within RPL networks. Through simulations it was discovered that control overhead can increase by up to 18 times, thereby impacting energy consumption and channel availability. This in turn can reduce the delivery ratio of packets by up to 30% and nearly double the end-to-end delay in a network. A strong correlation between the position of the attacker and the effect on the network was also observed. An attacker located as far away from the root as possible causes the highest increase in overhead, and similarly a higher path length between the attacker to the root also causes the higher packet loss. It was also discovered that loops and rank inconsistencies created by the attack are generally located around the neighborhood of the attacker, with parents or alternate parents experiencing the maximum loops, followed by the descendants.

Such impact caused by the version number attack makes it important to develop mechanisms that can be used to mitigate it, especially since the integrity of the version field in the DIO message is not verified even in secure operation mode of RPL. However, since RPL is likely to be used in low-power lossy networks comprised mostly of constrained devices, these mitigation strategies should be as simple as possible.

Acknowledgments. This work was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Program.

References

1. Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, J., Alexander, R.: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. IETF RFC 6550 (March 2012)
2. Seeber, S., Sehgal, A., Stelte, B., Rodosek, G.D., Schönwälder, J.: Towards A Trust Computing Architecture for RPL in Cyber Physical Systems. In: IFIP/IEEE International Conference on Network and Service Management (CNSM), Zürich, Switzerland (October 2013)

3. Sehgal, A., Perelman, V., Kuryla, S., Schönwälder, J.: Management of Resource Constrained Devices in the Internet of Things. IEEE Communications Magazine 50(12), 144–149 (2012)
4. Dunkels, A., Gronvall, B., Voigt, T.: Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In: 29th Annual IEEE International Conference on Local Computer Networks (LCN), Tampa, FL (November 2004)
5. Ko, J., Dawson-Haggerty, S., Gnawali, O., Culler, D., Terzis, A.: Evaluating the Performance of RPL and 6LoWPAN in TinyOS. In: Workshop on Extending the Internet to Low Power and Lossy Networks (IP+SN), Chicago, IL (April 2011)
6. Baccelli, E., Hahm, O., Günes, M., Wählisch, M., Schmidt, T.C.: RIOT OS: Towards an OS for the Internet of Things. In: 32nd IEEE INFOCOM Poster Session, Turin, Italy (April 2013)
7. Cheneau, T.: SimpleRPL (February 2014), `https://github.com/tcheneau/simpleRPL`
8. Tsao, T., Alexander, R., Dohler, M., Daza, V., Lozano, A., Richardson, M.: A Security Threat Analysis for Routing Protocol for Low-power and Lossy Networks (RPL). IETF Internet Draft <draft-ietf-roll-security-threats-06> (December 2013)
9. Chugh, K., Aboubaker, L., Loo, J.: Case Study of a Black Hole Attack on LoWPAN-RPL. In: Proc. of the Sixth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE), Rome, Italy, pp. 157–162 (August 2012)
10. Weekly, K., Pister, K.: Evaluating Sinkhole Defense Techniques in RPL Networks. In: 20th IEEE International Conference on Network Protocols (ICNP), Austin, TX, pp. 1–6 (November 2012)
11. Wallgren, L., Raza, S., Voigt, T.: Routing Attacks and Countermeasures in the RPL-Based Internet of Things. International Journal of Distributed Sensor Networks 13(794326) (2013)
12. Landsmann, M., Perrey, H., Ugus, O., Wählisch, M., Schmidt, T.: Topology Authentication in RPL. In: 32nd IEEE INFOCOM Poster Session, Turin, Italy (April 2013)
13. Dvir, A., Holczer, T., Buttyan, L.: VeRA - Version Number and Rank Authentication in RPL. In: 8th IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS), Hangzhou, China, pp. 709–714 (October 2011)
14. Perrey, H., Landsmann, M., Ugus, O., Wählisch, M., Schmidt, T.: TRAIL: Topology Authentication in RPL. arXiv preprint arXiv:1312.0984 (2013)
15. Levis, P.A., Patel, N., Culler, D., Shenker, S.: Trickle: A Self Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In: 1st Symposium on Networked Systems Design and Implementation (NSDI), San Francisco, CA (March 2004)
16. Hui, J., Vasseur, J.: The Routing Protocol for Low-Power and Lossy Networks (RPL) Option for Carrying RPL Information in Data-Plane Datagrams. IETF RFC 6553 (March 2012)
17. Korte, K.D., Sehgal, A., Schönwälder, J.: A Study of the RPL Repair Process Using ContikiRPL. In: Sadre, R., Novotný, J., Čeleda, P., Waldburger, M., Stiller, B. (eds.) AIMS 2012. LNCS, vol. 7279, pp. 50–61. Springer, Heidelberg (2012)
18. Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., Voigt, T.: Cross-Level Sensor Network Simulation with COOJA. In: 31st IEEE Conference on Local Computer Networks (LCN), Tampa, FL, pp. 641–648 (November 2006)