A Study of the RPL Repair Process Using ContikiRPL

Kevin Korte, Anuj Sehgal, Jürgen Schönwälder

A Study of the RPL Repair Process using ContikiRPL

Kevin Dominik Korte, Anuj Sehgal, and Jürgen Schönwälder

Computer Science, Jacobs University Bremen
Campus Ring 1, 28759 Bremen, Germany
{k.korte, s.anuj, j.schoenwaelder}@jacobs-university.de

Abstract. The IETF developed the RPL routing protocol for Low power and Lossy Networks (LLNs). RPL allows for automated setup and maintenance of the routing tree for a meshed network using a common objective, such as energy preservation or most stable routes. To handle failing nodes and other communication disturbances, RPL includes a number of error correction functions for such situations. These error handling mechanisms, while maintaining a functioning routing tree, introduce an additional complexity to the routing process. Being a relatively new protocol, the effect of the error handling mechanisms within RPL needs to be analyzed. This paper presents an experimental analysis of RPL's error correction mechanisms by using the Contiki RPL implementation along with an SNMP agent to monitor the performance of RPL.

Keywords: Low-power Lossy Networks, RPL Routing, RPL SNMP MIB module, Experimental Evaluation

1 Introduction

The emergence of embedded devices capable of wireless communications is leading to a materialization of an Internet of Things. However, since most such embedded devices do not use an IEEE 802.11 WiFi interface [1], interconnecting them with the existing Internet infrastructure requires the development of protocols and systems that take into account the low-power and lossy radio standards currently used in embedded networks. To avoid the emergence of devices that are not interoperable and neither documented, the IETF developed the 6LoWPAN standard [2,3] to enable IPv6 networking over the IEEE 802.15.4 radio standard, which is commonly used in embedded networks [4].

The emergence of a standardized IP-based wireless sensor network framework made it apparent that traditional routing schemes, even if they work well on wired networks, are not designed to operate well under the constraints set by wireless sensor networks. Several manufacturers, especially in the field of home automation [5], have addressed this issue by developing their own proprietary routing solutions. Integrating such systems with the largely standards reliant Internet poses quite a challenge and as such, the IETF developed the RPL

routing protocol to address routing in low-power lossy networks in a standard way [6]. In fact, RPL has been developed keeping in mind the constraints posed by the 6LoWPAN adaptation layer, thereby making it a good choice for routing in embedded IPv6 networks.

RPL is designed to consider any specific objective functions, or a combination of objective functions, while choosing a route. Energy available on the path or looseness of the link are two examples of such objective functions that can be imposed in order to obtain routes via RPL. Since RPL is flexible enough to be extended to suit any objective functions, it is possible to attain reasonably good performance in any given situation. However, this flexibility also introduces the peril of encountering a situation where a complex objective function could make the decision making process cost more energy and time than a simple but fixed approach. These combination of advantages and possible pitfalls associated with using RPL make it important to fully understand this routing protocol, which is expected to become the default method for routing in 6LoWPAN networks.

While a number of studies on RPL [7,8,9,10] have already addressed the issues relating to the efficiency of routes discovered and the protocol overhead, this study focuses on node failures within networks using the RPL routing protocol and the ability of the network to recover from these situations. For the purpose of this analysis the RPL implementation provided as part of the Contiki OS [11] has been utilized. An implementation of the RPL MIB objects [12] within the Contiki SNMP implementation [13,14] allows us to access the necessary data and monitor the routing tree state while a network is deployed.

The following sections of this paper present a discussion of the related work in performance evaluation of RPL, followed by an overview on the RPL routing protocol. An analysis of the RPL rebuild process, along with results, is then presented before conclusion.

2 Related Work

The authors of [9] presented an experimental evaluation of the TinyOS RPL stack named TinyRPL. In their study they come to the conclusion that RPL is not less efficient than the Collection Tree Protocol currently used by TinyOS. To arrive at this conclusion they look at both, the packet reception ratio and the control packet overhead. They also note that RPL, while in storing mode provides backwards routes, which enable communication both from the leaf to the root and vice versa. While this study is interesting, it does not cover the recovery performance of RPL in case of node failures.

In [10] J. Ko et al. evaluate the interoperability of TinyRPL and ContikiRPL implementations. Their work shows that both implementations perform quite well independently. However, they do not perform as well when interoperating. As a result of this outcome they define achieving fast interoperability between the two implementations as one of their future goals. Interoperability of standardized protocols is quite important and this study brings forth important results for the two most popular sensor network operating systems thereby highlighting

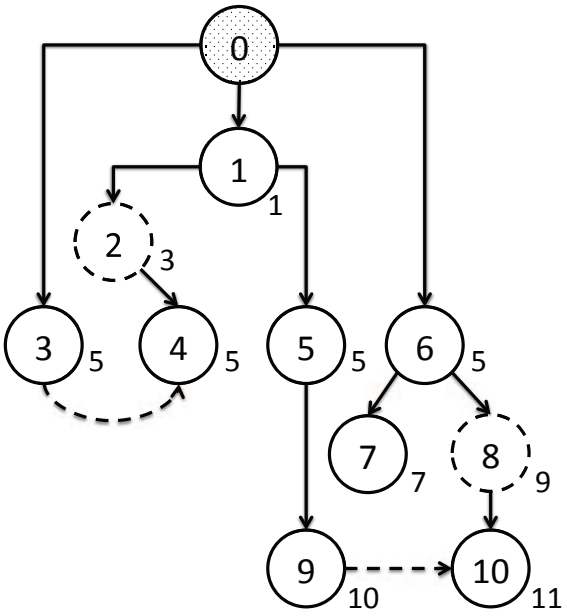

Fig. 1. A DODAG before the repair process. Borders of inactive nodes and routes are dotted. The shaded node (Node 0) is the sink. (Numbers inside the circles represent the node IDs, while the numbers adjacent to the circles are the rank of each node.)

performance issues that will be addressed in future revisions. However, this study too falls short of analyzing the effect of rebuilds in an RPL tree forced by node failures.

An overview of the weaknesses and strengths of the multiple facets of RPL is provided in [15]. In this work, the authors analyze all the different operations and features of the RPL routing protocol in order to provide a better understanding of the strengths and weaknesses this protocol might have. They have also supplemented their study with simulations performed in ns2 and on a wireless sensor network testbed. Their experimental investigation, however, unlike ours is focused upon the multiple timers that assist the trickle algorithm used by RPL and this is used in order to perform a critical evaluation of the workings of RPL. In contrast, our work is focused upon the actual performance of the RPL rebuild mechanisms, rather than a critical assessment of it.

According to Tripathi et al. [7], the results obtained from a simulated network using the RPL routing protocol are also very promising. They come to the conclusion that a network relying solely on global repairs will not be efficient and that therefor the focus has to be on local repair mechanisms. This is the only other study, thus far, which discusses the rebuild process of RPL, however, it is not the main focus. Furthermore, since these tests were performed using a sim-

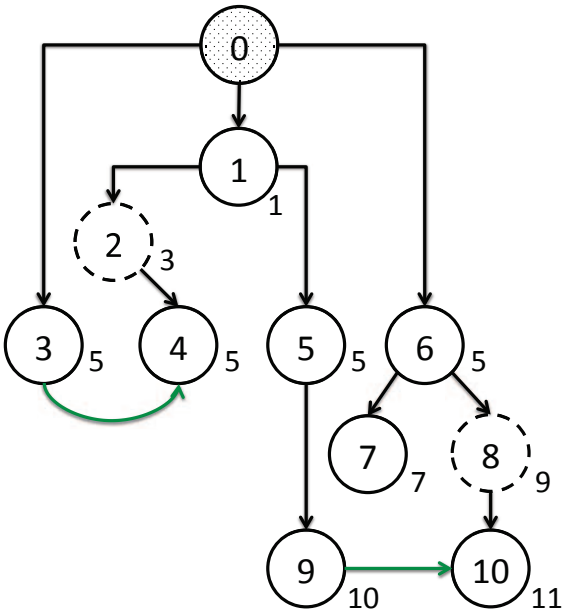

Fig. 2. A DODAG after the local repairs to a DODAG. Borders of inactive nodes and routes are dotted, repaired routes are marked Green. The shaded node (Node 0) is the sink. (Numbers inside the circles represent the node IDs, while the numbers adjacent to the circles are the rank of each node.)

ulator and did not consider the different repair mechanisms, our paper focuses on experimental evaluation of these local repair mechanisms.

3 A Brief Overview of RPL

An analysis of the routing requirements in the low-power and lossy networks area [5,16,17] led to the development of the RPL routing protocol [6] by the IETF's RoLL working group. To fulfill the multitude of requirements for routing protocols applied in low-power and lossy networks, RPL supports three traffic patterns, namely, point-to-point, point-to-multipoint and multipoint-to-point, as well as an extendable number of objective functions. These objective functions can target the different requirements of the routing tree, such as energy conservation or fastest delivery.

The RPL routing tree is a Destination Oriented Directed Acyclic Graph, in short a DODAG. The construction of a DODAG is started by the root node by broadcasting a DODAG Information Object (DIO). The DIO message contains the configuration of the DODAG, such as its traffic pattern, objective function or rebuild cycle. A node receiving the DIO checks whether it is already a part of this particular DODAG. In case the node is already aware of the DODAG,

Table 1. Computing Resources of Embedded Platforms

Platform	RAM	Flash	Clock Speed	Architecture
MICAz	4 kB	128 kB	8 MHz	8-Bit AVR
Atmel RZ-Raven	16 kB	128 kB	8 MHz	8-Bit AVR
Atmel RZ-Raven USB	8 kB	128 kB	8 MHz	8-Bit AVR
TelosB / TMote Sky	10 kB	16 kB	8 MHz	16-Bit MSP430
Redbee Econotag	96 kB	128 kB	24 MHz	32-Bit ARM7

but the newly received DIO gives it a better rank in the tree, it deletes the old information and proceeds as if it had no prior information regarding the DODAG. On the other hand, in case the node does not know about the DODAG, it will join the DODAG by storing the information provided by the DIO, computing its own rank using the objective function and informing the parent about this action by a message. The node then propagates the DIO further to its neighborhood, which repeats the action until all nodes have joined the DODAG.

In order to prevent loops from being created in the routing tree, a node is forbidden from selecting a parent with a rank lower than its own current rank. While this does a good job at keeping loops out of the DODAGs, for completeness, RPL is nonetheless equipped with a method to detect loops within the DODAG. In order to detect loops, every routing node in a path prefixes it's own IP header to the packet and every receiving node checks whether any of the prefixed headers has its own IP address as the source address. If the node finds its own IP address a loop is detected.

However, unlike loops, one of the main problems in low-power lossy networks is the possibility of nodes disappearing from the network, for example, because they run out of battery power or the link conditions degrade significantly. RPL has a system of methods to route through multiple parents and through siblings to repair the network in order to combat such scenarios of nodes disappearing from a network. The simplest, most useful, but also the most costly repair method, as such, is the global repair of the routing tree. During this process the root node increases the version number of the RPL DODAG and forces a rebuild of the entire DODAG. As is the case with the build process, this rebuild method also guarantees a loop free and best possible routing tree according to the objective function in use.

Since it would be counter productive to rebuild the entire DODAG each time a single node disappears, for smaller problems or between rebuilds, RPL has two prominent local repair methods. The first one allows routing through siblings in range, when a sibling is a node with the same rank. The second method is to switch parents. In Figures 1 and 2 one can see the two repair processes, where nodes 2 and 8 are failing, and node 4 is routing through its sibling, while node 10 changes its parent.

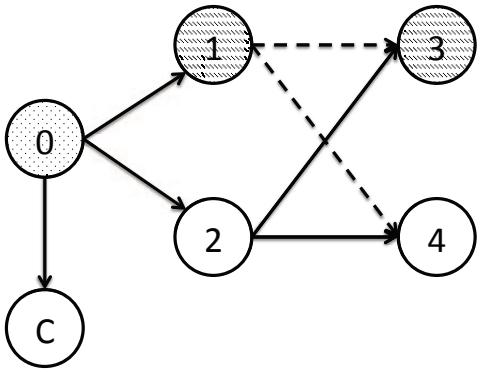

Fig. 3. The topology; Node 0 is the sink node, nodes with diagonal shading have a modified objective function, the dotted line indicates secondary routes and arrowheads indicate children relations.

4 Analysis of the ContikiRPL Rebuild Process

4.1 Experimental Setup

For the analysis of the RPL rebuild process the RPL implementation provided with the Contiki OS was used. Contiki supports many embedded development platforms, of which some of the popular ones can be seen in Table 1. It is quite clear that each platform offers highly variant capabilities in terms of RAM, flash, clock speed and architecture utilized.

While it might be the first instinct to choose the platform that appears to have the most RAM and flash memory available, this may not be the best strategy as the platform architecture influences the amount of actual resources available. For instance, the Redbee Econotag has a 32-bit ARM7 architecture, which causes it to load the entire program code into RAM before execution unlike the AVR or MSP430. Also, a 32-bit architecture means that machine code produced is generally larger in size than on the 8-bit AVR platform. However, a more thorough analysis of ContikiRPL's resource usage is needed before a platform can be chosen.

The ContikiRPL implementation takes up a lot of memory resources on target platforms. For instance, ContikiRPL on a MSP430 processor based RPL router like the TelosB or TMote Sky platform occupies close to 50kB of flash memory [15]. Similarly, on an AVR based platform ContikiRPL also occupies nearly 50kB of flash storage, but more importantly takes up nearly 6kB of RAM, which exclude the MICAz and RZ-Raven USB platforms. While this means that the Atmel RZ-Raven platform does have some RAM left to accommodate other applications as well, in reality the services provided by Contiki itself occupy most of the available RAM thereby not leaving much for a custom application.

As such, a Redbee Econotag was utilized as the target hardware platform, since it provides enough resources to run RPL without issues. The Econotag has

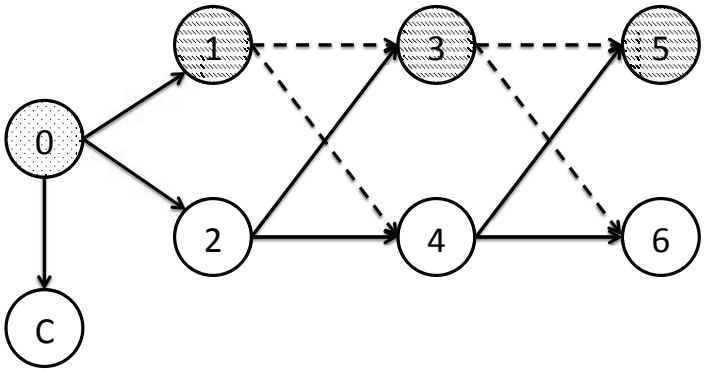

Fig. 4. The extended topology; Node 0 is the sink node, nodes with diagonal shading have a modified objective function, the dotted lines indicate secondary routes and arrowheads indicate children relations.

a Freescale MC13224v ARM7 microcontroller with 96kB RAM and 128kB Flash and a 802.15.4 radio interface, all of which leave enough resources available even after the basic Contiki installation is used on the device.

Furthermore, all nodes in an RPL network need to function in the router mode and as such the Econotag, due to its high RAM and Flash availability, was the most suitable target platform.

For the first three experiments, a network of six Econotags was set up in a linear topology indicated in Figure 3. In this topology, a simple objective function was used, which adds one to the parent rank for white nodes, for nodes with diagonal shading it adds one in the fallback parent testing and 8 in all other cases. This makes the previous white node in the tree the preferred parent to the following couple and secondary parent to its partnering diagonally shaded node. In the parent testing scenario, it also makes the diagonally shaded node the secondary parent to the following nodes and sibling in all other tests. This allows for an analysis of the working of sibling and parent relationships on leaf nodes, as well as their preferential treatment.

For the further experiments on changes within the mesh, the topology was extended to include an additional row of two nodes, as seen in Figure 4. The marking of the nodes and the objective function follow the same pattern as for Figure 3. The additional node marked as “C”, in Figures 3 and 4, was used to confirm that the local repair did not effect other branches of the network. As such, the local repair counter, as well as the global repair counter were monitored on this node.

The different aspects of the RPL rebuild process that were analyzed during this study are discussed further within their own sub-sections.

Table 2. Parent Fallback Time

Monitored Value	**Average**	σ
ICMP Runtime before disconnect	43.4ms	0.8ms
ICMP Runtime after fallback	43.8ms	1.0ms
ICMP Timeouts	211s	19.4s

4.2 Fallback Parent

The aim of the first analysis was to look at the time it took for a node to switch from a failed parent to a secondary parent. Therefore, node two and four were switched off, forcing node three to switch to node one as its only available parent. To monitor the time it took till node three was reachable again, node three was constantly contacted using ICMP echo requests. Additionally, the counter for the parent switch, which is provided by Contiki RPL, was checked afterwards to ensure the switch over had occurred as expected. To ensure that no rebuild took place, SNMP was used to monitor the DODAG version number of node three, as well as its local and global repair counters. The experiment was repeated until the standard deviation was one tenth of the average or less and until at least 20 values have been collected.

Table 2 shows the timeouts, runtimes and the standard deviations for this experiment. As one can see, the average unreachable time of the node during a rebuild was around three and a half minutes.

4.3 DIO Timer and Fallback Parent

In an additional step, the DIO Timers were altered to check whether this would influence the behavior of the nodes. After building a DODAG with the new values, the experiment from the previous section was repeated by switching off node 2. Once again, SNMP was used to monitor the version number and counters to ensure that no global repair was taking place.

An overview of the results can be seen in Table 3. These results show that the time it takes for a child to switch from one of the parents to another is not dependent on the time between two DIO messages. Since Contiki relies on its own neighborhood discovery methods to detect failing parents and does not rely on counters between two DIO messages, this can be taken as a confirmation that the Destination Advertisement Object (DAO) and DIO exchange between the node is not affected by the failing node. It also means that extra care has to be taken when selecting the parameters of Contiki's RIME stack in different situations.

4.4 Route Lifetime

Another counter that affects the expiration of the downwards routes is the route lifetime counter. By default Contiki sets it to infinity meaning that only when a DAO announcing a change arrives, the nodes will change the routes. Setting it

Table 3. DIO Timer and Fallback Time

DIO Timer	Average ICMP Timeout
2^3	203s
2^5	215s
2^7	214s
2^9	214s
2^{12}	211s

to any other value means that the route is only valid for this number of seconds and that an update is needed afterwards or the route is deleted.

If the time taken for route establishment and parent failure is known in advance, it would be possible to have small timeouts here. For example, the speed of many mobile nodes and their transmission range is known beforehand. This information might be a useful tool to set the timeout to a value forcing a reset when nodes are likely to leave the transmission range.

However, when dealing with static nodes failing due to energy problems or external influences, changing the route lifetime will only drain the nodes of energy as it forces them to send unnecessary announcements and in the worst case the node will fail right after renewing the lifetime. It should also be noted that in some scenarios, for example when having an energy aware objective function, the objective function might poison its rank when routes should be disregarded and the functionality might be doubled.

4.5 Fallback Sibling

The secondary fallback method provided by RPL is to route packets through a connection with a sibling. To force a node to use a connection through a sibling, nodes two and three were switched off, leaving node four with only the option to switch to its sibling, node one, in order to establish a working connection. Again ICMP echoes were sent to determine the time it took to find a working route. The values for the DODAG version number and the local repair counter as well as the global repair counter were retrieved. This again helped to ensure that no global repair had occurred but the local repair had taken place.

Table 4 gives the monitored values as well as their standard deviation. The time to rejoin is longer as it involves sending out a solicitation object and DIOs before the node is able to rejoin the tree.

4.6 Greediness of Nodes

Another problem when considering the routing tree is that a node may be trying to optimize itself to have as many parents as possible. To control how Contiki handles the greediness of the nodes, nodes one, two and four were switched on in different orders and the status of the RPL parents recorded in node four. The rank of the nodes was also monitored using the SNMP agent running on the nodes.

Table 4. Sibling Fallback Time

Monitored Value	**Average**	σ
ICMP Runtime before disconnect	43.5ms	0.8ms
ICMP Runtime after fallback	43.6ms	0.9ms
ICMP Timeouts	236s	13s

The obtained results showed that ContikiRPL optimizes its rank and is not greedy about its parents. For the node sequence two, four and one, the status of node four with a rank of two and node two as its parent was correct. For the node sequence one, four and two, however, the rank was still correct but the nodes did not forget about the parent. Upon a failure in node two, however, node one was not chosen as a fallback but a local repair was triggered instead. While this behavior means that the node appears to function properly, it makes automated monitoring difficult since the internal state is not consistent with the expected behavior.

4.7 Poisoned Tree

The RPL specification defines that upon performing a local repair, a router must poison its routes to indicate that it is moving and to prevent a parent attaching itself to one of its children. To check whether the routers properly poison the downwards routes, the network was set up as shown in the extended network in Figure 4, with node one being turned on after the network is stable. This way node four did not consider one as a parent. Then node two was turned off forcing node three and four to change their parents and thus change their rank, which required them to poison the routes to trigger a full rebuild of their subtree.

During this process nodes three, four and five were monitored. To check the time needed for the local repair to be completed and the message runtimes before and afterwards, ICMP echo messages were used. To verify the repair process, the repair counters for both the global and the local repair on these three nodes were monitored.

Using SNMP it was discerned that the local repair counter on node two increased. The resulting data from this experiment can be seen in Table 5, which illustrates the speed of local repair in case the nodes receive a poisoned rank message. Seeing that the time to reach node five is more then twice as long as the time needed to reach nodes three and four, it is confirmed that Contiki RPL poisons and rebuilds its subtrees as required by RPL.

In a second step, node two was turned on again. This step confirmed that Contiki is not poisoning its subtree wrongly, since a node should only do so if it cannot maintain its current or higher rank. This confirms the behavior found in the greediness analysis, since the node did not forget the old parent.

Table 5. Poisoning the Sub-DODAG

Monitored Value	**Node 3**	σ	**Node 4**	σ	**Node 5**	σ
ICMP Runtime before disconnect	44.3ms	0.7ms	44.8ms	0.3ms	63.4ms	0.9ms
ICMP Runtime after fallback	44.2ms	0.8ms	44.9ms	0.6ms	62.3ms	1.2ms
ICMP Timeouts	235s	12s	234s	6s	265s	18s

4.8 Rebuild Build Memory

The last experiment focuses on the global rebuild of the DODAG. During a rebuild, a new DODAG is constructed basing itself on the objective function. Since a simple counter function is used, the current state of the DODAG should not influence the new DODAG. In our experiment, the DODAG rebuild was triggered after node two was switched off and the preferred parents and rank of each node was recorded using SNMP. Afterwards, node two was switched on again and preferred parents and rank recorded again.

As expected, it turned out that the DODAG was rebuilt without the consideration of the state of the previous DODAG. Thus, the optimal DODAG, according to the objective functions was reached. It was also possible to confirm the result of the greediness analysis, as in all cases, that the best parent was chosen for the rank calculation.

5 Conclusion

In order to perform this study we designed and implemented an RPL-MIB using the Contiki SNMP agent. The RPL-MIB was utilized in order to study the behavior of the local repair mechanism of RPL. Our analysis has shown that the proposed RPL repair mechanisms work fairly well when given the time needed and that Contiki follows the required path poisoning whenever a local repair takes place, or the rank of the node worsens. It could also be seen that there is little difference between the time it takes to switch from the preferred parent, to the time it takes to switch to a completely new subtree. This leaves the time it takes to find a dead parent as the more important focus point when considering the maximum time allowed for an RPL DODAG to be repaired after a nodes failure or movement. While route timeouts and global repairs offer a working solution in some scenarios, it has to be carefully weighed against the extra energy used.

References

1. M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels, "Making Sensor Networks IPv6 Ready," in *Proceedings of the Sixth ACM Conference on Networked Embedded Sensor Systems (ACM SenSys 2008), poster session*, Raleigh, North Carolina, USA, November 2008.

2. J. Hui and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," *IETF RFC 6282*, September 2011.
3. G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," *IETF RFC 4944*, September 2007.
4. N. Kushalnagar, G. Montenegro, and C. Schumacher., "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement and Goals," *IETF RFC 4919*, August 2007.
5. J. Martocci, P. Mil, N. Riou, and W. Vermeylen, "Building Automation Routing Requirements in Low-Power and Lossy Networks," *IETF RFC 5867*, June 2010.
6. T. Winter and P. Thubert, "RPL: IPv6 Routing Protocol for Low power and Lossy Networks," *IETF Internet-Draft draft-ietf-roll-rpl-12.txt*, October 2010.
7. J. Tripathi, J. Oliveira, and J. Vasseur, "A Performance Evaluation Study of RPL: Routing Protocol for Low Power and Lossy Networks," in *44th Annual Conference on Information Sciences and Systems (CISS)*. Princeton, NY: IEEE, 2010.
8. ——, "Performance Evaluation of Routing Protocol for Low Power and Lossy Networks (RPL)," *IETF Internet-Draft draft-tripathi-roll-rpl-simulation-07*, August 2011.
9. J. Ko, S. Dawson-Haggerty, O. Gnawali, D. Culler, and A. Terzis, "Evaluating the Performance of RPL and 6LoWPAN in TinyOS," in *Workshop on Extending the Internet to Low power and Lossy Networks (IP+SN 2011)*, Chicago, IL, USA, April 2011.
10. J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggety, A. Terzis, A. Dunkels, and D. Culler, "ContikiRPL and TinyRPL: Happy Together," in *Workshop on Extending the Internet to Low power and Lossy Networks (IP+SN 2011)*, Chicago, IL, USA, April 2011.
11. A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, ser. LCN '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 455–462.
12. K. Korte, J. Schönwälder, A. Sehgal, T. Zhou, and C. Zhou, "Definition of Managed Objects for the IPv6 Routing Protocol for Low power and Lossy Networks (RPL)," *IETF Internet-Draft draft-sehgal-roll-rpl-mib-02*, October 2011.
13. S. Kuryla, "Implementation and Evaluation of the Simple Network Management Protocol over IEEE 802.15.4 Radios under the Contiki Operating System," Master's thesis, Jacobs University Bremen, July 2010.
14. S. Kuryla and J. Schönwälder, "Evaluation of the Resource Requirements of SNMP Agents on Constrained Devices," in *5th Conference on Autonomous Infrastructure, Management and Security (AIMS 2011), Springer LNCS 6734*, June 2011.
15. T. Clausen, U. Herberg, and M. Philipp, "A critical evaluation of the IPv6 Routing Protocol for Low Power and Lossy Networks (RPL)," in *Proc. IEEE 7th Int Wireless and Mobile Computing, Networking and Communications (WiMob) Conf*, 2011, pp. 365–372.
16. A. Brandt, J. Buron, and G. Porcu, "Home Automation Routing Requirements in Low-Power and Lossy Networks," *IETF RFC 5826*, April 2010.
17. K. Pister, P. Thubert, S. Dwars, and T. Phinney, "Industrial Routing Requirements in Low-Power and Lossy Networks," *IETF RFC 5673*, October 2009.