

Evaluation of RPL-compliant routing solutions in real-life WSNs

Theodore Zahariadis^a, Lambros Sarakis, Stamatis Voliotis, Dimitrios Bargiotas, Panagiotis Karkazis

Technological Educational Institute of Sterea Ellada, Electrical Engineering Department, Psachna 34400, Greece

Abstract. Wireless Sensor Networks (WSN) applications continue to expand and already cover almost all our daily activities improving from security and environmental efficiency to gaming experience. The diverse applications running on top of WSNs have led to the design of an immense number of routing protocols. Few years ago, the IETF standardized the IPv6 routing protocol for low-power and lossy networks (RPL) which is based on routing metrics to build communication paths between a source and the destination node. While significant efforts have focused on the design of routing metrics that satisfy the various applications, limited work has been reported on validating their performance using real-life notes. In this paper, we focus on validating the simulation results (obtained through the JSim simulator) using TelosB motes for a set of routing metrics that have been proposed in our previous articles.

1 Introduction

The performance of any communication network strongly depends on the employed routing protocol. This is even more prominent in Wireless Sensor Networks (WSNs) where all nodes act as routers participating in the forwarding procedure to satisfy the requirement for infrastructure-less operation. Any routing protocol has to take into account the characteristics and the constraints of the target network, as well as the performance requirements of the application(s) to be supported.

In the case of WSNs, the design of such a protocol is a fairly complicated task, due to the distributed nature of the network and the need to support constrained devices (in terms of energy, processing and memory resources) communicating in a cooperative, multi-hop manner over unreliable wireless links. On the other hand, the diverse applications that WSN systems support nowadays range from simple environmental conditions monitoring to time-critical applications relevant to the operation of production lines in manufacturing environments, health or even military operations. Furthermore, the required cooperation between nodes (combined with their resource-constrained nature) raises concerns regarding security issues, especially in hostile or unattended environments. Focusing on network layer attacks, a “malicious” (i.e., non-fully cooperative) node may refuse to forward all or part of the received traffic towards the destination.

To achieve their goal, routing protocols rely on metrics to compute a weight (cost) for a path that interconnects an origin and a destination node. When more than one such path exists in the network, routing metrics are used by the routing protocols to select the most preferred among these paths. Depending on the

routing metric, the most preferred path can be the one with the minimum number of traversed hops, the maximum reliability, the minimum delay, the maximum energy efficiency etc. To defend against routing attacks in WSNs, an efficient approach has been pursued, namely trust management [3], capable of mitigating a wide set of attacks. Following this approach, WSN nodes establish trust relationships based on their expectation (trust) that their neighbors will sincerely cooperate on particular tasks (e.g. data forwarding).

Few years ago, the IETF standardized an IPv6-based routing protocol called routing protocol for low-power and lossy networks (RPL) [1], which provides a mechanism whereby traffic from devices inside the Low power and Lossy Network (LLN) is routed towards a central control point inside the LLN. RPL defines specific rules for the route cost calculation and the nodes’ operation but leaves the freedom to define the routing metric that will be used to the system designer/operator [2]. In [3], we have explored the concept of routing algebra that has been well-established from other networking systems to define the requirements that the routing metrics should satisfy to ensure optimal and loop-free operation of the routing protocol. In [4] and [5], we have defined routing metrics that satisfy these requirements and we have additionally defined alternative ways to combine them so that more than one performance aspects are jointly optimized. In these articles, to evaluate the achieved performance, we have used computer simulations and namely, the JSim simulation environment.

In this paper, we implement the RPL protocol in TelosB motes and perform experiments for three different routing metrics. We observe the operation of a real

^a Corresponding author: zahariad@teiste.gr

network of 16 nodes and compare it to the operation and results obtained through computer simulations.

2 The implemented routing solution

RPL constructs Directed Acyclic Graphs (DAG) and defines the rules based on which every node selects a neighbor as its preferred parent in the DAG, thus forming a tree. For the construction of the tree, each node calculates a rank value and advertises it in the so-called Destination Information Object (DIO) messages. The rank of a node is a scalar representation of the “location” of that node within a destination-oriented DAG (DODAG). In particular, the rank of the nodes must monotonically decrease from the leaves towards the DAG destination (root), i.e. the rank of every child is greater than the rank of its preferred parent. Each node selects as parent the neighbor that advertises the minimum rank value to guarantee loop-free operation and convergence. The rank has properties of its own that are not necessarily those of all metrics. It is the Objective Function (OF) that defines how routing metrics are used to compute the Rank. In this paper, we focus on evaluating the operation of a WSN when the adopted routing metric is the Hop Count (HC) or its combination with Packet Forwarding Indicator (PFI) which reveals the trust of a node on its neighbor, according to [4].

3 Implementation in real-life motes

To explore the implementation requirements and investigate the performance of a RPL-compliant routing algorithm, we implemented it in real sensor nodes under TinyOS v2.1 operating system which is an open-source operating system designed for wireless embedded sensor networks. The motes we used were CrossBow TelosB which are equipped with ZigBee IEEE 802.15.4 PHY interface, an 8MHz MSP430 processor, 10K RAM and 48K ROM, thermistor, light and humidity sensors and supported NesC programming language.

3.1 TinyOS

TinyOS is an open source, BSD-licensed operating system designed for low-power wireless devices, such as those used in sensor networks, ubiquitous computing, personal area networks, smart buildings, and smart meters. A worldwide community from academia and industry use, develop, and support the operating system as well as its associated tools, averaging 35,000 downloads a year. TinyOS has a component-based programming model, codified by the NesC language, a dialect of C. TinyOS is not an OS in the traditional sense; it is a programming framework for embedded systems and set of components that enable building an application-specific OS into each application.

TinyOS implements a component-based architecture which enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks. It provides component library includes network protocols, distributed

services, sensor drivers, and data acquisition tools – all of which can be used as-is or be further refined for a custom application. The TOSSIM [7] framework was used to debug, test and analyze algorithms in a controlled and repeatable environment.

A TinyOS program is a graph of components, each of which is an independent computational entity that exposes one or more interfaces. Components have three computational abstractions: commands, events, and tasks. Commands and events are mechanisms for inter-component communication, while tasks are used to express intra-component concurrency. A command is typically a request to a component to perform some action, such as initiating a sensor reading, while an event signals the completion of that service. Events may also be signaled asynchronously, for example, due to hardware interrupts or message arrival. Commands and events cannot block: rather, a request for service is split phase in that the request for service (the command) and the completion signal (the corresponding event) are decoupled. The command returns immediately and the event signals completion at a later time.

Rather than performing a computation immediately, commands and event handlers may post a task, a function executed by the TinyOS scheduler at a later time. This allows commands and events to be responsive, returning immediately while deferring extensive computation to tasks. While tasks may perform significant computation, their basic execution model is run-to-completion, rather than to run indefinitely; this allows tasks to be much lighter-weight than threads. Tasks represent internal concurrency within a component and may only access state within that component. The standard TinyOS task scheduler uses a non-preemptive, FIFO scheduling policy.

TinyOS abstracts all hardware resources as components. While many components are entirely software-based, the combination of split-phase operations and tasks makes this distinction transparent to the programmer. For example, consider a component that encrypts a buffer of data. In a hardware implementation, the command would instruct the encryption hardware to perform the operation, while a software implementation would post a task to encrypt the data on the CPU. In both cases an event signals that the encryption operation is complete.

3.2 Performance validation

The focus of this section is the validation of the routing solutions (proposed in [4]) through the comparison of results obtained from the experimental setup of real-life nodes with results obtained from the simulation platform.

The testbed follows the topology comprising of 15 TelosB nodes, as depicted in Figure 1. Different configurations and parameters have been applied. In the considered configuration, the sink node (node id 1) was periodically sending application-layer CoAP messages (GET) to 7 nodes (nodes in green), which were periodically responding to CoAP messages transmitted from sink node. The configuration has been tested for

different number of malicious nodes (attackers), namely 0, 3 and 6.

These configurations have been validated for three different composite routing metrics, namely, HC, the lexicographic combination of HC and PFI (Lex(HC, PFI)) and the additive combination of these metrics (Add(HC, PFI)). For the latter composite metric, several weighting factors have been tested, concluding that the combination of (0.6, 0.4) was performing better than the others.

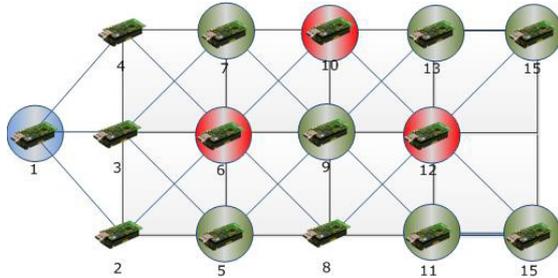


Figure 1. Test-bed scenarios, configurations and parameters for 7 data sessions and 3 attackers

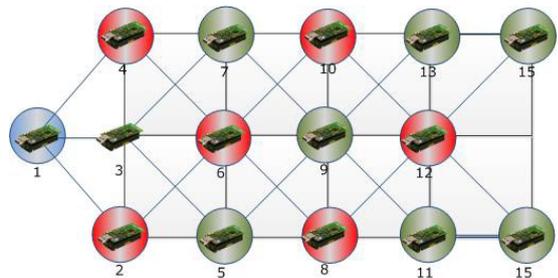


Figure 2. Test-bed scenarios, configurations and parameters for 7 data sessions and 6 attackers

In all aforementioned configurations, the data packet time interval was equal to 3 seconds, the number of requested data packets was equal to 500 and the payload equal to 20 bytes. It is noticed that fragmentation and reassembly on IP layer was not activated. Moreover, it is noted that in order to achieve multi-hop routing in the test-bed environment, nodes have been compiled in such a way that control as well as data messages processed by nodes were those transmitted by one-hop neighbours.

Finally, CoAP messages start being transmitted after the establishment of the network connections through the appropriate routing control messages. The performance metrics for these cases are Packet Loss percentage (denoted as “PL%”) and Round Trip Time (denoted as “Latency”).

3.3 Evaluation results in the absence of malicious nodes

Direct comparison between real testbed and simulation results under ideal conditions, in the sense that no malicious nodes are present, shows that there are differences between the simulation and the real environment. The results gathered from the simulation environment, present a PL% equal to zero for all nodes

due to the absence of packet collisions on network level, while for the real testbed case, PL% balances between 4% and 14%, proportionally increasing with number of hops. It must be noted that, as it will be proved in the subsequent sections, the PL% is quite higher from other configuration settings as all nodes are transmitting data packets to the sink node, increasing the possibility of collisions at layer 2. In short, comparison between PL% values (heavy traffic conditions) and values (without attackers) reveals that the main reason for the increased PL% values is mainly due to the larger number of nodes transmitting in network level simultaneously. A second (expected) conclusion is that latency increases proportionally to the number of hops. Differences between the latency values between simulation and real test-bed environment can be explained by the fact that JSim simulation platform uses different values for the various RPL protocol timers that cannot be compared to real conditions.

3.4 Evaluation results in the presence of malicious nodes

Moving to the case where malicious nodes exist in the network, the results from the experiments with the real motes confirm the simulation results qualitatively. Starting from the case where the HC metric is used to decide the route to the sink, when the topology of either Figure 1, or Figure 2 is set up in the simulation, the packets of the same flow have the same experience: they either find their way to the sink or not. This is also confirmed in the experiments with the real life motes. The reception of data packets is in a “binary” state: either the sink node receives all data packets or none of them. It must be highlighted that in both cases, the selection of the first node to interact with solely depends on the neighbor first sending DIO message. Thus, this node is the first to populate the list of potential forwarding nodes and is selected in case where the metric value among all neighbors is the same. The nodes cannot distinguish between two paths of the same length in hops that include different number of malicious nodes.

Combining the HC with the PFI which is a metric built on the trust each node has on its neighbor in a lexical manner, it is expected that the PL% will be reduced compared to the previous scenario, as the nodes will be able to identify nodes acting maliciously and react accordingly, by selecting other neighboring nodes for forwarding data packets towards sink node. This is verified by the experimental results included in Table 1. The selection of the composite metric does not affect either PL% or Latency in the absence of malicious nodes. Of course, there exists PL% due to failures and collisions in the network. As previously mentioned, comparing the PL% values of this configuration with those obtained under the HC metric, one can observe that the inclusion of PFI has a positive impact in both real test-bed and simulation platform environments. Focusing on the real testbed results, which are of primary concern, it is evident that the utilization of the lexical composite metric leads all involved nodes to discover paths towards the

Table 1. Results from experiments with TelosB motes for different routing metrics and malicious nodes presence

Node ID	TelosB-ADD(HC,PFI)						TelosB-LEX(HC,PFI)					
	# of attackers = 0		# of attackers = 3		# of attackers = 6		# of attackers = 0		# of attackers = 3		# of attackers = 6	
	PL(%)	Latency (ms)										
5	0	132	0	124	0	125	4	132	4	119	8	121
7	0	152	0	153	0	122	4	128	8	142	8	116
9	0	168	0	169	0	258	6	168	10	168	10	267
11	0	218	0	289	0	219	2	225	10	283	18	218
13	0	224	0	254	0	230	6	224	18	250	10	230
14	0	224	4	230	4	260	4	278	24	331	18	270
15	0	247	4	250	4	280	6	272	20	310	20	290

destination node, in contrast to the utilization of HC metric. The PL% values in the presence of 3 and 6 attackers can be explained as the initial interactions of the nodes ignore the behavior of the neighboring nodes. As the number of interactions increases, each node “builds” a confidence factor for neighboring nodes that assists on selecting the proper path towards destination node. It must be noticed that in the cases of the presence of 3 and 6 malicious nodes, the simulation results also present a non-negligible PL%, increasing as the number of malicious nodes increases.

Finally, adopting the *additive routing metric* that combines HC with PFI, the performance is significantly improved as indicated in Table 1. Comparing the columns corresponding to the additive combination of HC and PFI with the columns corresponding to their lexical combination, it is evident that all data flows experience packet loss below 5% when the additive routing metric is adopted. In case the lexical combination is used, the packet loss ranges between 8 and 24% in the presence of malicious nodes. The latency may slightly increase in the case of the additive routing metric, which comes as no surprise since the packets may follow a longer path in order to avoid the malicious nodes.

4 Lessons learned

In general, it has been verified that qualitatively simulations and experiments with real life motes lead to the same conclusions although there are significant quantitative differences. The main reasons for the differences in the two different environments are the following.

First, in JSim simulation platform, the memory size and the execution (run-time) environment does not reflect in a realistic way the restrictions of the real sensor nodes running the single-threaded TinyOS. Also, collisions that happen in the real network might affect differently the network performance than estimated in simulation, especially if the transmission rate of the periodic messages is high. Moreover, the radiation pattern of the sensor nodes is different than that assumed in the simulation environments.

Second, the connectivity graphs of real test-bed deployment and simulation platforms, even for the same topology, are different. The reason is that transmission signal strength fluctuates in real-test bed scenario (asymmetric or unstable links), while in JSim or TOSSIM

the transmission loss is constantly defined. Consequently, this has a direct impact on the neighbor list of the nodes and thus at the routing decision process.

Third, due to the fact that all motes are using the same channel, sometimes collisions occur. The number of collisions as well as their random existence differs from simulation to real testbed cases. This has an impact on sniffing functionality of the code (forwarding metric and reputation response validation) and consequently to the routing decision process.

Finally, despite these discrepancies regarding measured data in the two environments the general qualitative conclusions about the performance and the behavior of the protocols (very good responsiveness with prompt change of routing paths) are the same for both environments.

References

1. T. Winter, P. Thubert (Eds.), “RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks”, IETF RFC 6550, March 2012
2. JP. Vasseur, M. Kim (Eds.), “Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks”, IETF RFC 6551, March 2012.
3. P. Trakadas, T. Zahariadis, S. Voliotis, P. Karkazis, Terpsichori Helen Velivassaki and L. Sarakis, “Routing metric selection and design for multi-purpose WSNs”, IIWSSIP 2014.
4. P. Karkazis, P. Trakadas, H. C. Leligou, L. Sarakis, T. Velivassaki, I. Papaefstathiou, Th. Zahariadis, “Evaluating routing metric composition approaches for QoS differentiation in low power and lossy networks”, Springer, Wireless Networks, Vol. 19, Issue 6 (2013), Page 1269-1284, DOI 10.1007/s11276-012-0532-2.
5. P. Karkazis, I. Papaefstathiou, L. Sarakis, Th. Zahariadis, T. H. Velivassaki, D. Bargiotas, “Evaluation of RPL with a Transmission Count-Efficient and Trust-Aware Routing Metric”, in Proc. 2014 IEEE International Conference on Communications (ICC), Sydney, Australia, June 2014.
6. <http://www.tinyos.net/>
7. <http://tinyos.stanford.edu/tinyos-wiki/index.php/TOSSIM>