Evaluation of ContikiRPL Local Repairs

Kevin Dominik Korte

Jacobs University Bremen
School of Engineering and Science
Campus Ring 1
28759 Bremen
Germany

Thesis for the conferral of a Masters of Science in Computer Science
Submission date: July, 26th 2011
Supervisor: Prof. Dr. J. Schönwälder
Second Reader: Prof. J. W. Wallace

Declaration

I, Kevin Dominik Korte, born 27.02.1988, hereby declare that this thesis is the result of my independent work, has not been previously accepted in substance for any degree and is not concurrently submitted for any degree.

.....

Place, Date Kevin Dominik Korte

Acknowledgments

I would like to express my gratitude to Prof. Dr. Jürgen Schönwälder and Anuj Seghal for their many helpful comments and their guidance through my Master's Thesis. I would also like to thank Mihaela Rusu for copy editing, Nikolay Melnikov for his encouragement and Vladislav Perelman for numerous insights into Contiki.

Abstract

Low-power sensor nodes have become an important topic in today's computer science and with the popularization of the Smart Grid and the Internet of Things they are now gaining industry's interest. After the hardware and communications oriented standards such as IEEE 802.15.4 and 6LoWPAN were widely accepted, the focus is now shifting towards the routing protocol. With the publication of the "Routing Protocol for Low Power and Lossy Networks", RPL, this has been standardized as well. Within this thesis the ContikiRPL implementation has been connected to an SNMP agent. Further, using this agent, the performance of the ContikiRPL local repair mechanisms has been evaluated on the RedBee EconoTAG hardware platform. This in the end allows a prediction on the behavior and on the extent the Contiki implementation is suitable for usage.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Home Automation, Sensor Networks and the Smart Grid are only some of the catchwords that come into mind when one thinks about smart objects in a network context. In the last year the idea of an Internet of Things has found its place not only in the research community, but it has also reached the everyday's life. Navigation systems interfaced with the car phones and stereos is certainly only one of the many examples where networks of small systems already work together to complete a task.

Smart objects are in general often described as a network of miniature computers connecting different physical items, including input and output devices, and providing communication services to the network. This dual role of communications provider and end system is one of the characteristics of the smart objects[1]. This characteristic is directly related to the limitations of their wireless interface, which is often only powerful enough to reach 30 meters as opposed to the 100 meters of an 802.11 network or the multiple kilometers reached by traditional radio and mobile phone technologies. This in turn means that for an efficient network a meshed routing is needed since otherwise the number of necessary base stations would be far too high. As a meshed network requires all participants to relay messages, the dual characteristic becomes obvious. An example is visualized in figure 1.1.

But the limitations in the network's range and transmission power are not the only restrictions smart objects have to face. The second fundamental restriction is the highly limited amount of computing power and storage available. This means that all the computations done by the devices are constrained to run with processing power in the kHz range. Therefore, one needs to restrict the computations to be done to the minimum to not block the processor. A second, possible more important, effect is that the network stack is in most cases not able to handle more than a single packet at a time, which means that anything that comes in has to get out very fast so that there is space for the next packet to come in. This sets requirements to the used routing, such as the rather fast availability of a working path,

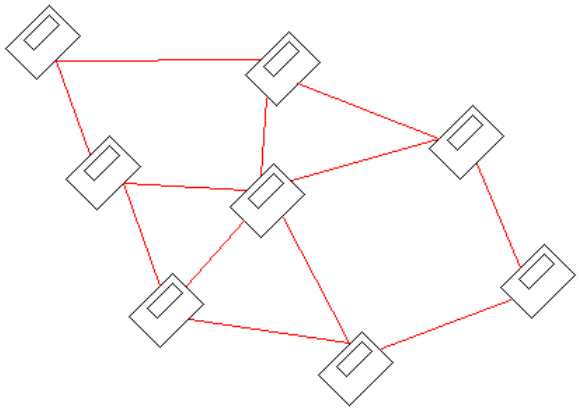


Figure 1.1: Meshed Network

which in turn consumes storage and processing power. The restrictions on the storage also set the requirement that not only the programs and the operating system have to be reasonably small, but also the routing tables should be both storage and computationally efficient.

The transmission power is not the only thing limited by the distances the radios can cover. It also has a considerable impact on the battery power. Thus the power consumption of all components is one of the most crucial issues. This has a great impact on the processing as well as on the network. It is apparent that any operation has a certain power foot print, from flashing an LED to computations and sending packets. As such it should be clear that one has to trade off aggregation of data on a node for sending multiple packets [2]. Another energy saving factor is the time spent in idle and in sleep mode.

If one looks at all the different hardware characteristics mentioned, it is noticeable that these are not static, but constantly changing[1]. Nevertheless, they always stay below the line of today's consumer hardware. Therefore, they will also always require special, standardized communication protocols.

With the increasing standardization around the Internet of Things and the wireless sensor networks it is based on, it became apparent that traditional routing schemes, even if they work well on wired networks, are not designed to operate on the constraints set by wireless sensor networks. Several manufacturers, especially in the field of building automation [3], have answered this problem by developing their own system on determining the routes between different motes. These systems are neither interoperable

nor in any way documented, so that there is no possibility of using devices outside the manufacturer's product line even if the remaining parts of the system rely on standardized mechanisms and protocols. Further, most of these systems depend on rather complex nodes, like PCs, to do the calculations, and thus the system is not self-contained or energy efficient.

To address the situation, the IETF has developed the "Routing Protocol for Low power and Lossy Networks"[4]. This does not only cover the problems caused by a vendor-specific solution, but also considers the constraints set by the limited resources of the IEEE 802.15.4 devices and the 6LoWPAN adaptation layer. With the approval of RPL in March 2011, it is now finally available to be used. This is only reinforced by the release of the Contiki OS implementation of RPL.[5]

As RPL is set out to consider any constraints or combination of constraints while choosing a route, with energy available on the path or looseness of the link being only two examples, it is clear that the system is flexible enough to perform reasonably well in any given situation and not just in sensor networks. Of course, this flexibility introduces the danger to lead into a situation where the device is drained of much energy or where these problems are avoided at an unreasonable price. To further enhance the functionality and resource conservation behavior of RPL, local repairs are possible, avoiding the need to rebuild potentially large routing trees.

Within this thesis the Contiki based ContikiRPL implementation was connected to the SNMP implementation [6] for Contiki using a Management Information Base (MIB) module which was refined in the course of this work. Using this SNMP integration, it will further be shown how performance-capable the local repair mechanisms are and which pitfalls occur when relying on RPL to deploy a stable routing tree. This thesis therefore integrates into the already existing theoretical and practical evaluations presented in 3.2 and extends their results by a practical analysis of the local repair schemes found within RPL. Chapter 1 and chapter 2 provide an introduction to the topic and the technical backgrounds of this thesis respectively. Chapter 3 presents the related works, including general information on RPL, further evaluation as well as the an detailed summary results of selected simulations. Thereafter chapter 4 presents the MIB module and its implementation. Chapter 5 presents the settings of the experimental evaluation and the evaluation itself and is followed by the conclusion in chapter 6.

Chapter 2

Technical Background

The following section will give an introduction into the technical details surrounding Low Power and Lossy networks and the working of the Routing Protocol for Low Power and Lossy networks. It will further provide a brief introduction into the Simple Network Management Protocol as well as into the Contiki Operating System and the hardware it was ran on.

2.1 IPv6 over Low-power and Lossy Networks

With the increasing market, the pressure on being able to use standard tools such as web-browsers to configure and control small devices has led to the development of IPv6 over Low-power and Lossy Networks standard, in short 6LoWPAN. 6LoWPAN provides an adaptation of the IPv6 standard to devices with limited resources, such as IEEE 802.15.4 based devices. As such it can be placed between the second layer of the Open Systems Interconnection Reference Model, which in this case would be the IEEE 802.15.4 specified link layer, and the third layer of the model, in this case the IPv6 protocol. RPL was then in turn designed to build up a routing tree over an existing 6LoWPAN network. Therefore, this section is dedicated to giving an introduction to the 6LoWPAN standard.

The main part of the 6LoWPAN layer is the dispatch header. This header is shown in figure 2.1. It is identifying the frame as a 6LoWPAN packet and informs the implementation about what type of address is to be expected next.

The dispatch header is always the first header in a sequence of headers. The short introduction of the components will follow the required sequence, even if most of the headers are optional.

The first header is the mesh routing header. Mesh routing is used for IEEE 802.15.4 devices to increase their range by requiring normal nodes to act as routers and transfer the packets. The mesh header defines the final destination and the original source, while the normal header is used for the

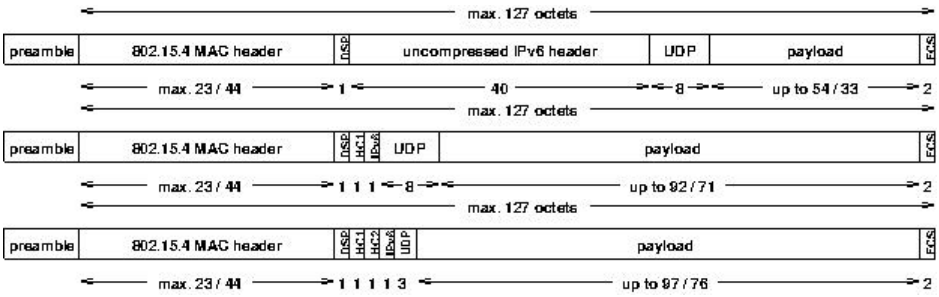


Figure 2.1: Uncompressed and Compressed 6LoWPAN Frames

current link. A meshed network was already shown in figure 1.1.

Apart from the limited range, the frame size given by the IEEE 802.15.4 standard is the biggest problem. A normal IEEE 802.15.4 frame is only 127 octets and the minimum maximum transmission unit of an IPv6 packet is required to be 1280 octets[2]. To resolve this discrepancy, the 6LoWPAN standard defines a mechanism of fragmenting and reassembling IPv6 packets into multiple frames.

Another method for keeping the size of the IPv6 packets small is by compressing the headers. This is true for the layer 3 as well as for the layer 4 header. The 6LoWPAN standard defines a stateless compression for the link local address of the IPv6 header. This compression scheme is referred to as HC1 and is done by simply transferring the onset of the address and constructing the full address from the node's own address and the transferred address part. For the higher layer protocol a HC2 compression scheme is only defined for UDP and works with a similar assumption of reconstructing the information from already known information.

The to-be-RFC 6282 [7] will replace these link-local compression schemes with a stateful method, working on link-local, global and multicast addresses. The compression is thereby utilizing the context information retrieved from the MAC address or the IEEE 802.15.4 device identifier together with the local prefix in order to allow an efficient compression of the IP layer. Moreover, the new compression method allows the mixture of compressed and uncompressed addresses, which means that the source within the 6LoWPAN network can be compressed to 0 bits using the context information, while the destination can be a full IPv6 address. The method described in the document also allows the compression of additional UDP ports and, if authorized by the application, the compression of the UDP checksum.

Another important part is the multicast system. Multicast is used for neighborhood discovery. The 6LoWPAN standard [8] defines a header for multicast over meshed routing in order to discover motes which are more than one hop away.

2.2 Low Energy and Lossy Networks and RPL

With the establishment of the IEEE 802.15.4 motes as the hardware standard and the maturing of the 6LoWPAN, it is important to look at the organizational side of the network. For the self-organization of the routing within the network the "Routing Protocol for Low power and Lossy Networks" [4] was approved in March 2011.

The following parts are now dedicated to an introduction into RPL. First, the challenges and goals of RPL will be introduced and then the technical solutions will be presented.

2.2.1 Challenges and Goals of RPL

While for a long time only server and desktop computers were participating in a network, the emergence of smart phones has shown that one could not ignore mobile devices anymore. While this has changed the aspects of mobility in the network, it has not changed the routing with many nodes grouped around a central router powered by a power line[1] . As said in section 1, this is going to change with the increasing number of smart objects. First of all, the limited range of their radios is not as big as the range of mobile phones or 802.11 radios, and thus the networks require meshed topologies. Further, for laptops and cell phones the limited number of devices per person makes keeping track of the battery state a manageable task, while the number of smart objects is potentially much bigger. Lastly, the already mentioned limitation of resources is something which most networks today would simply ignore.

Apart from the limited resources and the topology of the network, the loss rate of the network is another challenge to be faced. This means that the links between two nodes are not to be seen as being as reliable as higher powered wireless networks and the traditional wired networks. As you can see in section 3, routing schemes for these networks are focused on hop count, bandwidth or similar metrics, but do not take the loss rate into account.

The last major problem that RPL has to take into account is the limited frame size of the devices. It can utilize at most 92 octets inside of a 6LoWPAN packet. As sending packets can quickly drain the devices out of energy, it is clear that the packets can not be simply fragmented when sending larger amounts of data.

This already shows that RPL was designed to be flexible enough to find the best compromise between these constraints in any of the given situations. To achieve this, RPL is able to use any metrics available on the device [4], which is fundamentally different from traditional routing protocols which mostly use hop counts for building up their routing tree. Apart from this flexibility in choosing a suitable metric, RPL has many other features, such as local subtrees, which are explained below in section 2.2.2.

2.2.2 The Workings of RPL

The following sections are dedicated to the inner workings of RPL. They will give a comprehensive overview over the DODAG building process, routing within an RPL tree, error handling, node joining and movement.

DODAG Building

In the initial state there is only one or multiple root nodes. When considering multiple root nodes it is important that these nodes already have a working connection to each other. Whether this is again realized using RPL or another protocol isn't important. One should also note that this procedure is repeated periodically to rebuild the RPL tree as an additional error prevention mechanism.

When the root node initializes a tree building procedure, it broadcasts a DIO, a DODAG Information Object. This DIO contains information about the sending node, such as its rank and root node, and information about the DODAG, such as the ID and the Objective Code Point (OCP), which represents the Objective Function. When a node receives a DIO it first checks whether it already knows about this DODAG and, if not, it calculates its rank within the DODAG, informs the parent node that it chooses it as parent and by itself broadcasts a DIO. If the node already knows about the DODAG, it also calculates its rank. If the DIO was sent by its current parent, it replaces the rank and sends out a DIO with the new rank. If the DIO is not from its parent, it only replaces its parent if the node's rank is improved, in which case the node informs the old and new parents about the changed relation and then sends out a DIO.

As any non-root node has only one primary parent, a loop free graph can be constructed with the completion of the build process. Since the node communicates to the parent during the build process, it is also ensured that any node in the multi-hop range of the root can reach its parent.

To finish the building process, a node needs to send DODAG Destination Advertisement Objects, DAO, up the tree. The DAO message contains the prefix which the sending node has knowledge of. As it is a prefix advertisement, one can aggregate the prefixes if the node is reasonably sure that there are no other nodes outside its subtree having this prefix.

One of the important parts in the build process is the rank calculation. The rank in itself determines the position of the node in the DODAG and is highly dependent on the Objective Function. While the root node is always the level zero node, its children cannot be assumed to be level one nodes. This fact is important when talking about the RPL definition of siblings. They are defined as nodes of the same rank, which, together with the previous mentioned rank calculation, means that nodes of the same parent do not have to be siblings, while siblings are allowed to have different

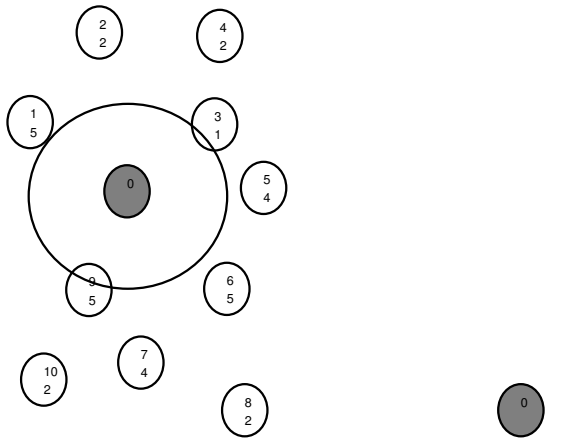


Figure 2.2: RPL Routing Tree Building Step 1: The root starts the building

parents.

An example can be seen in figures 2.2 to 2.5. In all four figures the (a) part marks the network, where each node has a node ID and, below, a metric. The gray node is the root node. The right side (b) shows the current state of the routing tree while going through the figures. It also shows the rank below the nodes.

In figure 2.2 the root node starts by transmitting the first DIO. This is then received by all the nodes within the range of the root node. These will then reply so that the root knows about them.

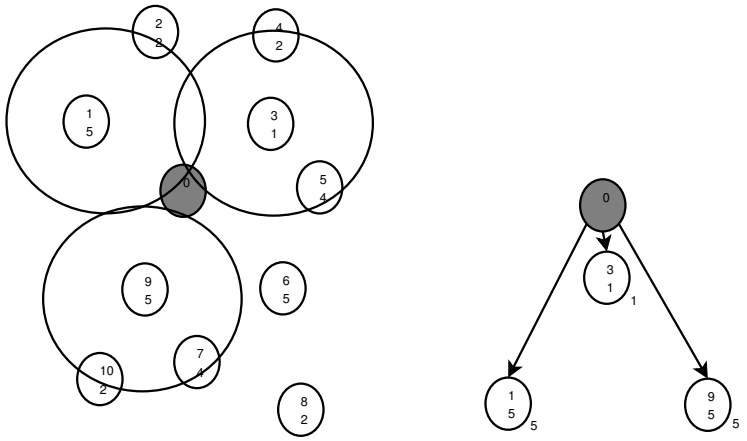
Figures 2.3 and 2.4 show the continuous build process. All nodes that have received the messages in the previous step transmit a DIO to their neighbors. In figure 2.3b one can see the nodes added in figure 2.2. It is also noticeable that not all the children of the root are siblings as not all of them have the same rank.

Finally, in figure 2.5 the last nodes got added to the RPL tree and the full tree can be seen. Figure 2.5a shows that even if the build is completed, the last nodes as well transmit a DIO each as the nodes don't know that there are no more nodes left. Further one can see in Figure 2.5b that, due to a better rank, node 2 is changing its primary parent.

Routing

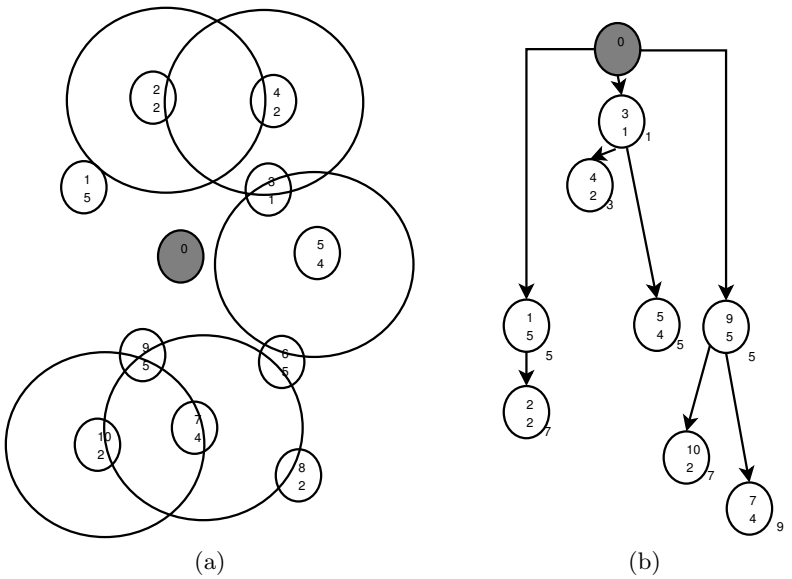
In the more simple case of global routing each node now has from this tree a defined path to its descendants and a default route up to the root node. Thus routing can be said to work in the way of sending traffic upwards until a node has a straight path to the destination.

The RPL Instance and the respective DODAG should be chosen depending on the quality of service settings within the IP packet or should follow



(a) (b)

Figure 2.3: RPL Routing Tree Building Step 2: The first level of nodes relay the DIO



(a) (b)

Figure 2.4: RPL Routing Tree Building Step 3: The next level of nodes relay the DIO

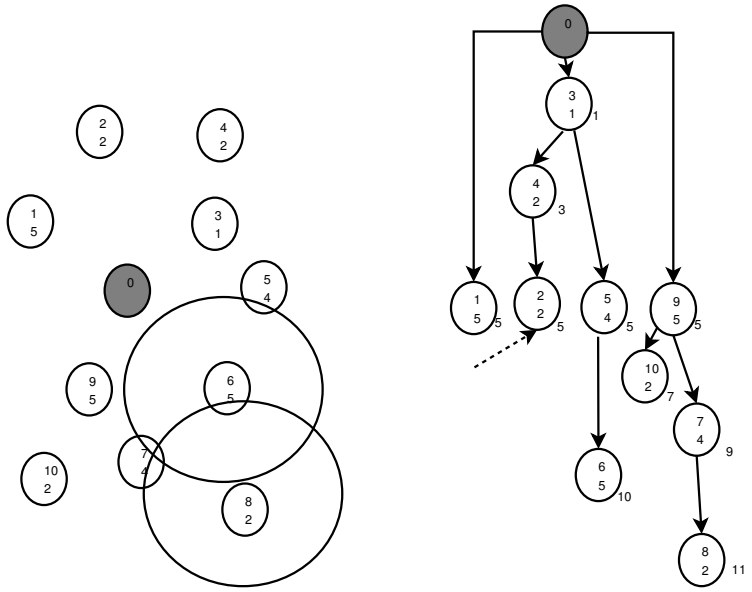


Figure 2.5: RPL Routing Tree Building Step 4: All nodes are in the routing tree

the set default DODAG and its routes. The default DODAG can be changed by a DIO from the current root node for the whole network or when a considerable change in the node's internal state occurs e.g. depletion of the main power source. Apart from children, RPL also allows optional routing to siblings of a node to shorten the path.

Besides this simple case, RPL has also a mode of always sending traffic to the root node. From the root node the traffic should then be sent to the respective nodes. This is especially useful if the network is rather flat, with many nodes connected to a grounded root, and if the root at the same time performs aggregations over the data for the destination node. In this case one can reduce the amount of energy lost on sending the headers compared to the energy used on sending data.

Apart from the routing according to the routing tables, RPL offers routing with an additional header [9]. This header may then specify either a part of the route or the full route. This is especially helpful when a node is on a path which does not have the capabilities to store a route other than the default one.

The difference between subtree routing and routing through the root can be seen in figure 2.6, which uses the tree built in section 2.2.2. In figure 2.6b the dashed path between the two black nodes shows the path using the subtree, while the dotted route shows the path trough the root node.

In figure 2.7 the difference between tree routing and sibling routing can

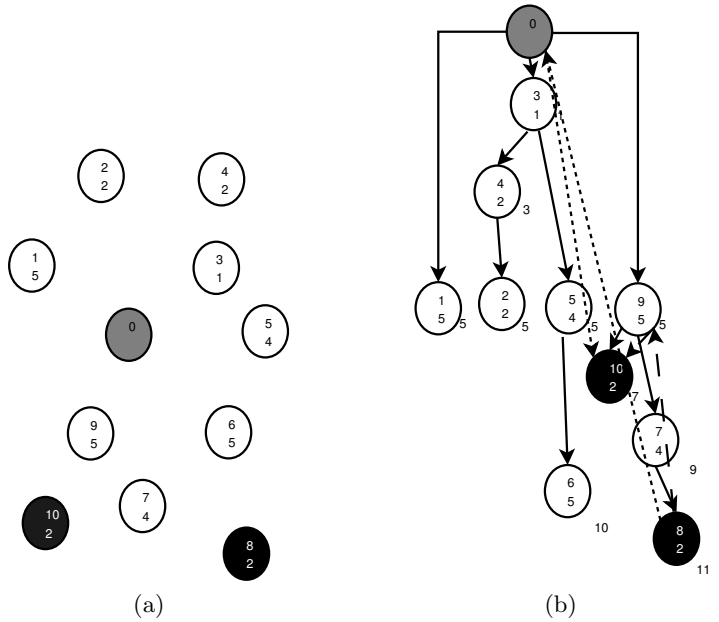


Figure 2.6: Routing through the Root and Subtree Routing

be seen. The dotted path between the black nodes is an example of sibling routing while the dashed path is the normal path trough the tree.

Local DODAG

A local DODAG can be built up by any node within an RPL Instance and contains this node and a few neighbors. It is mostly used to optimize trees, for example by allowing a connection between two deep branches of a DODAG. It should also be used when avoiding routing large flows up to the root of the DODAG and then down again to the destination.

Building up and routing work like with normal DODAGs with the exception that local DODAGs have only one root node.

Loop Detection

Loops are avoided already due to the method by which the trees are built. However, this does not consider the case when a node is missing a rebuild cycle or when nodes are moving. RPL transports its control information within the headers of ordinary data packets' external extension headers such as the one in "RPL Option for Carrying RPL Information in Data-Plane Datagrams", the main idea being that, when there is no need for a route to be established, it is not of importance whether or not it is loop-free.

When a loop detection [10] is performed on a packet, a node sending a packet will prepend its own IP header. A receiving node will check whether

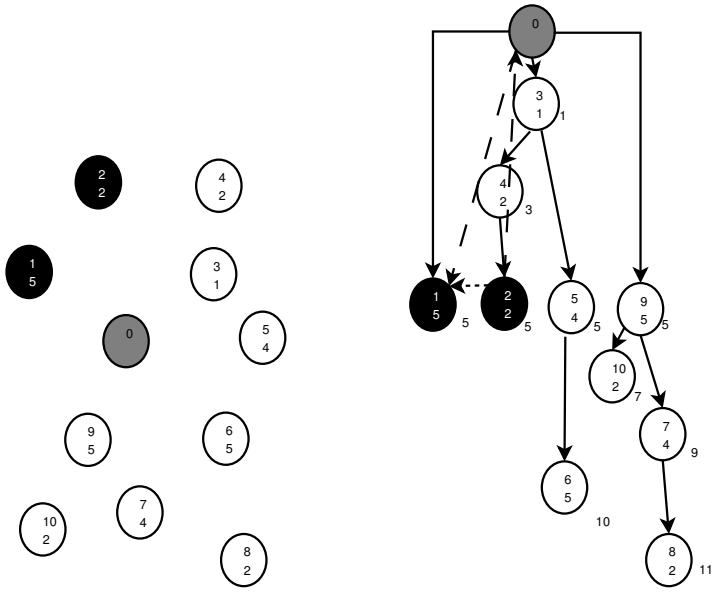


Figure 2.7: Routing through the RPL Tree and Sibling Routing

any of the headers has its own IP address as the source of one of the packets. If this is the case, it sends an error back to the original source and initiates the repairs as outlined in 2.2.2.

Local and Global repairs

RPL knows of two repair methods, local and global repairs. As it is the case with many protocols, RPL prefers local repairs, which affect only broken elements, to global repairs affecting all nodes.

Global repairs can only be triggered by the root node and work in the same way as the DODAG building as outlined in 2.2.2. Consequently, after a global repair, all nodes will be again at their most optimal position.

The local repair is designed for finding quickly an alternative path without giving much consideration to the optimal path. When a node detects that it cannot reach its parents, it first tries to shift its default root to the best of its siblings until either its parent comes up again or the next rebuild is performed. If the node has no reachable siblings or none of them is responding, the node will behave towards its parets as if it had been newly added. Depending on the node's settings it will thus either wait for the next periodic sending of a DIO by its neighborhood or send a DODAG Information Solicitation, DIS, message requesting a new DIO from the DODAG. Afterwards the node will find its new best parent and rejoin the network.

One of the more important tasks to prevent loops in the routing tree is

to poison the subtree after a local repair. Poisoning the tree hereby refers to announcing its own rank to be infinity and thereby triggering a change of parents or a local repair in the children. When they are rebuilding their tree they will not continue to announce their rank, which is coming from the faulty parent. This is especially important if their old rank is better than the new rank of the parent, as that would make it possible for the parent to attach itself to its own child and create a loop.

In figure 2.8 one can see the two alternative local repair strategies. The dotted nodes are going down. In response, node 2 is choosing a temporary path using its sibling node, while node 8 is choosing one of his secondary parents as alternative.

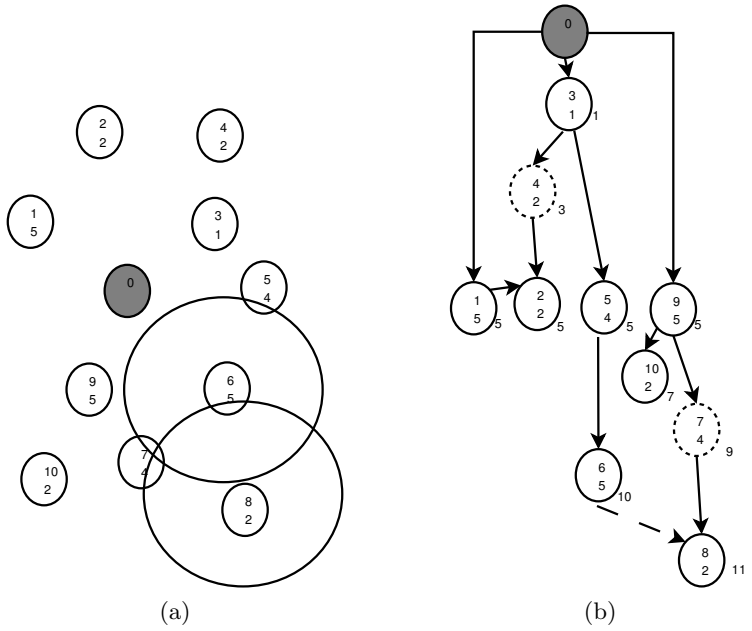


Figure 2.8: Local Repairs

Node Movement

When looking at the node movement, one has to differentiate between moving a node between two DODAGs and within a DODAG.

Within the same DODAG a node may only move upward to improve its Objective Function. This means a node may choose only a parent which would allow the Objective Function to improve the node's own rank. This should avoid greedy situations when one of the node's children has a higher or equal rank compared to the parent.

The node movement is done by sending a DIO to the new parent, notifying it about the intention to join. After having successfully joined a new

parent, the node informs its former parent that it is not anymore a child of his. Both parents notify then the upward tree about the change of routes. The node should also announce the change to its children who then either can choose a new parent or stay with the old one and notify in turn their children about the change in structure.

The procedure of changing between two DODAGs of the same RPL Instance is similar to the above described one. The difference as such is that the node may attach itself to a node of any rank if it has not been previously a member of this DODAG in the current version. If the node has been a member of this DODAG in the current version it is treated as movement within the DODAG.

If it is not possible to join a DODAG in accordance with these rules, the node may join the DODAG by performing a local repair. This includes poisoning its subtree as outlined in the previous section.

In general, a better way to cope with node movement, especially when a lot of movement can be expected, is the periodic rebuild of the whole RPL instance with all its associated DODAGs. The rebuild not only avoids problems raised by the delay in updating the tree, but it also helps to conserve energy as a potentially large DODAG is not traveled upwards to notify about many individually moving nodes.

Node Joining

If a node wants to join a network after being powered on, it has two options which it can follow. The first option is to wait for a periodic rebuild. In this way the node can join in a way that not only permits it to find a good position, but also allows the network to integrate the node well into its structures. Waiting for one of the rebuilds also provides the great advantage for the network that no additional messages are required to be sent as they are transmitted in any case during the rebuild.

A quicker way for the node to be included in the network is to request a DIO from the network, by sending a DODAG Information Solicitation, DIS, message and attach this node to a parent. As said, the node broadcasts an information request to all reachable nodes. Thereby it can either include a DODAG ID if it had knowledge of the available DODAGs or ask for all available DODAGs and then choose any of them. It joins the DODAG by informing its chosen parent about this action, which in turn sends the messages upwards. The new node does not need to add any route but the default one as it does only have a route to its parent node.

2.3 The Simple Network Management Protocol

In the beginning of computer networks no one has imagined that one day networks with hundreds of small devices need to be managed, including

switches, routers and environmental sensors. When one now considers that each of these devices provides a multitude of parameters to monitor and configure, the need for a standardized network management protocol can easily be seen. The Simple Network Management Protocol (SNMP) [11] provides the methods to exchange the management and configuration data in an application layer protocol in a standardized and vendor independent way.

The most recent version of the protocol, 3, was released in 2002. The original version, released in 1988, is still widely used, but is suffering from the lack of adequate security considerations.

Since summer 2010 there is now an SNMP implementation for Contiki [6] developed at Jacobs University Bremen. This implementation should be extended to include the RPL MIB.

The next section will describe the concepts behind SNMP, while section 2.3.3 speaks about the advantages of the used implementation.

2.3.1 Communications Architecture

The communication takes place between managed nodes, which are also known as SNMP agents, and network management stations. The network management station thereby acts as the monitoring and control node, while the SNMP agent is running on the managed device, where it responds to the commands as set by the respective management station. Both the agent and the management station need access to the same Management Information Base (MIB) to interpret the transmitted data. More information on the MIB can be found in section 2.3.2.

Communications is driven by a periodic polling by the network management station, which retrieves the data from the respective agents. In case of special events, the SNMP agent can send a notification, called Trap, to the management station. However the agent should not send it more than once, but rather rely on the management station to retrieve the information in the normal polling cycle.

The communications between the agent and the network management station take place over UDP, which allows SNMP to take full advantage of the compressions defined within 6LoWPAN.

2.3.2 Management Information Base

The Management Information Base specifies the variables for the device and features or protocol which can be monitored or modified. This of course means that an SNMP agent, running on a device with multiple monitored protocols enabled, will have more than one MIB module to use. In order to read or to write the different variables, SNMP only needs to traverse through the tree structure formed by the different MIB modules and is then

able to access the different variables. This allows for a standardized access to all parameters and thus for a simple way of configuring and monitoring different devices.

Within the MIB module there are multiple Object Identifiers (OID). OIDs define a consistent numbering of the different variables and enables the network management stations to access all parameters by using these OIDs as a consistent and unique numbering scheme.

MIB modules are written in the Structure of Management Information (SMI) Definition Language. Currently both IEEE and IETF maintain large collections of MIBs.

The structure, implementation pitfalls and remarks within Contiki of the "Definition of Managed Objects for the IPv6 Routing Protocol for Low power and Lossy Networks (RPL)" [12] can be seen in section 4.

2.3.3 SNMP for Contiki

Within his Master Thesis at Jacobs University Bremen, Siarhei Kuryla developed an SNMP application for Contiki [6]. It supports both Version 1 and 3 style message processing for the Get, GetNext and Set operations. On the authentication side it provides user-based security with the HMAC-MD5 authentication as well as CFP128-AES-128 encryption.

The implementation allows adding MIB modules on demand by using the add_scalar and add_table functions. Therefore, the extensions to the respective SNMP program can be done without interfering with the program as a whole.

A further advantage of this implementation is that it is well tested within the thesis and known to work on the chosen hardware platform. In addition, the size of the implementation is known to be small enough to run side by side with additional applications.

2.4 The Contiki Operating System

The Contiki Operating System [13], used within this project, has the goal to provide developers with an operating system that can be easily used to build software for smart objects. The following section will give a short introduction into the Contiki OS as well as into the hardware used at Jacobs University Bremen.

2.4.1 Introduction to Contiki

The Contiki Operating System was started in 2003 by Adam Dunkels and is now developed by an industry and academia funded team of developers. It has been ported to twelve different platforms, including the AVR platform from Atmel. The Contiki Operating System is using protothreads [14, 15]

for the event-driven scheduling. These protothreads provide, for the cost of 2 bits per state, an abstraction around the state machine, which by now is so famous that it has been ported to many different applications, including high performance clusters and set-top boxes, but also miniature systems such as weather sensors[1].

Contiki is also well known for using the world's smallest IPv6 stack, the uIPv6 stack[16], which was developed in 2006. This IP stack is also widely used outside of the Contiki Operating System. The uIPv6 stack, requiring only 1KB of RAM, was developed especially to accommodate the restrictions on the processing power and on the memory that emerge when using smart objects.

This includes not only the IP protocol, but also the ICMP, UDP and TCP protocols, and is not just a proof of concept implementation of an IP stack on smart objects, but it is also used in many real life scenarios. In the 6LoWPAN stack most platforms, including the ones presented below, are using the new stateful compression. However, this is not true for all platforms.

2.4.2 Hardware

From the hardware platforms available, two were used during course of this project. One is the already mentioned AVR Raven produced by Atmel [17], the other one is the RedBee EconoTAG from Redwire [18].

Atmel AVR Raven

The AVR Raven is an IEEE 802.15.4 device. It is equipped with 2 of Atmel's own AVR processing chips as well as with a custom segmented LCD display, speakers, a microphone, sensors and serial ports. The LCD display, which is in its own uncommon for sensor motes, is segmented, which means it is addressed as a serial device rather than a frame buffer. This reduces the need for a graphics library to be included when using the display, saving a considerable amount of space. This LCD display, together with the other input and output devices, are driven by the ATmega3290PV. The second processor, the ATmega1284PV, is driving the radio communications stack. Other tasks can be decided between the two units. This co-working is displayed in figure 2.9.[17]

Apart from the Ravens, the demo kits contain a USB stick, which is used to connect a computer to the IEEE 802.15.4 network. When attaching the USB stick, flashed with an RPL border router, the interface will be using the Microsoft RNDIS driver.

For flashing Contiki onto Raven motes and the USB stick, a programmer, which transfers the information over the JTAG connection onto the motes and the USB stick, is needed. The full set can be seen in figure 2.10.

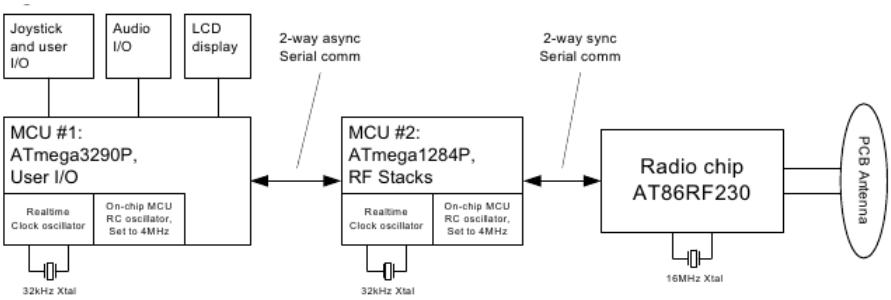


Figure 2.9: Raven Boards Micro Controller Setup[17]

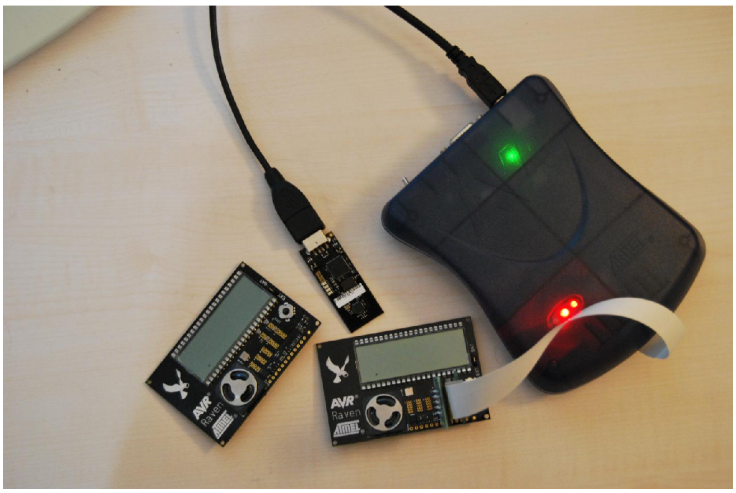


Figure 2.10: Raven Motes, USB Stick and Programming Kit

2.4.3 RedBee EconoTAG

The ARMv7-based RedBee EconoTAG is a more powerful smart object and can be seen in figure 2.11. The ARMv7 processor from Freescale is equipped with 96kB of RAM and 128kB of Flash ROM, with the image being transferred fully from ROM to RAM before execution. The board provides UART and JTAG access over the USB interface. Therefore it is possible to both use the motes and debug it using standard JTAG tools at the same time without the need for external equipment.[18]

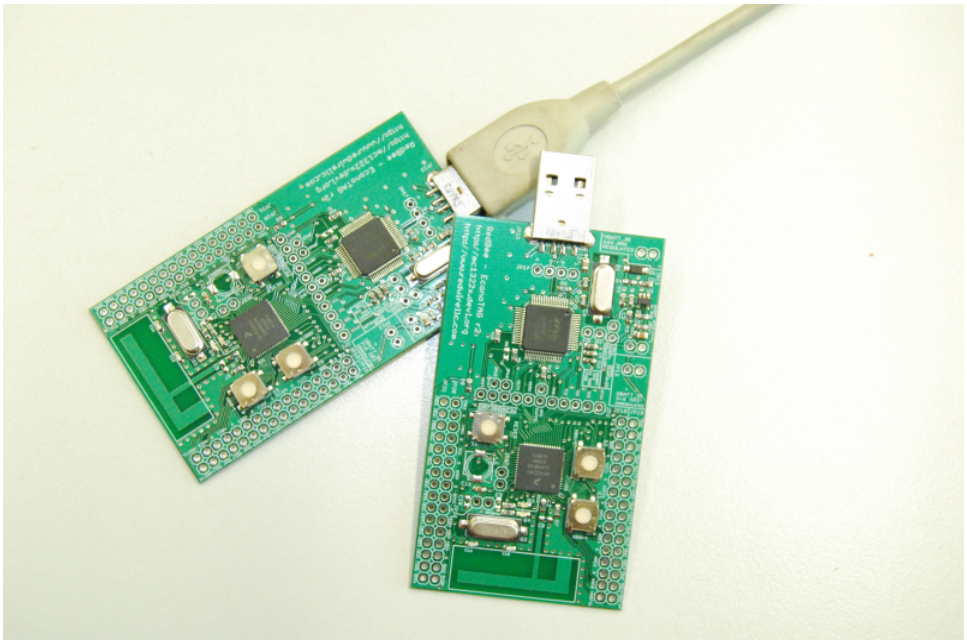


Figure 2.11: Two RedBee EconoTAGs

Chapter 3

Related Work

3.1 Overviews and Discussions of RPL

The best point for getting not only into RPL, but also into the field of smart objects, is the book released in 2010 by Jean-Philippe Vasseur and Adam Dunkels "Interconnecting Smart Objects with IP - The Next Internet" [1]. The book is divided into 3 major sections. In the first section the authors consider the underlying architecture, such as the definition of a smart object, the ideas behind IP and the routing on the current Internet. It is also explained what are the possible difficulties encountered when applying these technologies to LLNs. After the analysis in the first section of the book, the second section introduces the technologies currently used with smart objects such as the Contiki Operating system 6LoWPAN and ZigBee. In the last section there follows an analysis on how smart objects are or could be used in different scenarios. For the purpose of this paper, the most important chapter of the book is chapter 17, where a detailed introduction into RPL was presented. Apart from the introduction to RPL, the book also presents an extensive analysis of an OMNET++ simulation of RPL, where the analysis not only tackles the question whether RPL creates efficient paths, but also looks into the size of the routing table, the control traffic and the failure tolerance. While the authors admit that simulation results can never replace a real life deployment, it shows at least that RPL can be expected to to perform well in LLNs.

As routing is one of the most prominent topics in networking, there are many different algorithms available for it. For a starting point one might want to look at the Internet-Draft "Overview of Existing Routing Protocols for Low Power and Lossy Networks" [19]. It gives not only a very good point of entrance in the whole topic by presenting a very nice introduction to the problems which one has to consider when dealing with low power and lossy networks, but also an introduction to the routing protocols developed within the IETF. Thereby they follow the common subdivision between Link State

and Distance Vector Protocols. By listing the shortcomings of the different protocols, the Internet draft of the ROLL working group comes to the final conclusion that none of the commonly used protocols satisfies the needs of a multi-hop routing scenario. From that they conclude the need for RPL.

Many of the other documents published by the ROLL working group are considering applications for Low-power and Lossy Networks. The first of these informal documents is the "Industrial Routing Requirements in Low Power and Lossy Networks" [20]. As the name suggests, the draft is mainly considering routing situations within industrial environment, such as factory monitoring. In the document a clear statement is made that, due to the stability required within industrial processes, it is unlikely that the wired networks will get replaced by wireless ones in the near future. For all networks, the authors make a clear distinction between the different reliability and latency levels needed and how this could be achieved using RPL.

Along the same line, the draft "Building Automation Routing Requirements in Low-Power and Lossy Networks" [3] describes the usage in building automation, especially in HVAC. The interesting difference to the previous mentioned paper is that they consider that the configuration accepted by the installing people, as well as by the end user, is much lower than what is accepted in the industrial environment. The paper also specifies clearly the distinctions between the different building networks such as HVAC, security and lighting. It is also explained how they would profit from interacting over a common network and what the security considerations are when multiple networks of this type work together. In a last step it moreover explains how mobility and device tracing could be included into the analysis based on the idea of tracing the wheelchairs in a retirement castle. Another important aspect is the ease of replacement of a broken node. The authors clearly state that in most cases a member of the janitor staff will be called for a failing light switch and he should be able to repair it without involving the usually much higher paid IT staff. This of course calls for a zero conf network, where also the role of a device is easy to be determined. All in all, this paper gives an accurate overview of all the requirements a combined system has to fulfill.

The next report in this row is the one on "Home Automation Routing Requirements in Low-Power and Lossy Networks" [21]. While many requirements and applications are similar to the report on "Building Automation Routing Requirements in Low-Power and Lossy Networks", there are also unique characteristics to the Home Automation field. One difference is the experience of the maintainers. While in bigger buildings a janitor which can be trained for basic maintenance is usually available, at home the user has only a manual or has to rely on external support staff which comes with extra costs. This accounts for the stronger emphasis on a self-configuring system. Another interesting idea presented is to replace the many fixed

switches around the house by a remote control per inhabitant. On the one hand, this would make maintenance easier as the number of devices would greatly decrease if there are no switches, but on the other hand it would require quick rebuilding of the routing tree as the remote control is moved through the house. Another aspect targeting the reliability of the routing is the question whether such a network could be used to monitor the health status of sick or elderly people and thus allow them to live at home. Taken together, this shows the many aspects one has to consider when looking at RPL in home automation.

On the implementation side there is of course the abstract to the poster presentation of the RPL algorithms in Contiki. In "Low-power wireless IPv6 routing with ContikiRPL" [5] the authors do not only give a good introduction to RPL, but also explain how the implementation works. Even more interesting is the evaluation of used memory and the energy efficiency of the ContikiRPL implementation. The only shortcoming is that most of the results are based on their Cooja simulator, which means they might need some validation. All in all, this is certainly a good starting point for RPL in Contiki.

The paper "A Survey on Routing Metrics TIK Report 262" [22] targets more the question of how to classify the different routing metrics. This taxonomy was used for grouping the different metrics used within the target function of RPL. Furthermore, this grouping would later allow to find comparable metrics between RPL and more traditional routing methods.

An interesting discussion on the different routing protocols for mobile ad hoc networks can be found in the "A Comprehensive Comparison of Routing Protocols for Large-Scale Wireless MANETs" [23] paper. It compares the Ad hoc On-demand Distance Vector, AODV, the Temporally-Ordered Routing Algorithm, TORA, and Location-Aided Routing, LAR, in terms of delay and packet delivery ratio. It finally comes to the conclusion that LAR is the best scalable of the three routing protocols and that TORA is nearly not usable for large scale networks.

The paper "RPL Based Routing for Advanced Metering Infrastructure in Smart Grid" [24] is looking at the RPL from the perspective of using it in a smart metering infrastructure. The paper includes both a custom objective function and some slight modifications to adapt to the requirements of smart metering in both the tree construction as well as the routing behavior. While their simulations in the NS-2 can be seen as a general validation of the concepts behind RPL, it remains unclear whether their implementation would be able to work together with an implementation following the RPL recommendation by the IETF.

3.2 Evaluation of RPL

This section will briefly look into the different simulator evaluations of RPL, before presenting detailed results of two of the papers in the next section.

An evaluation of RPL can be found in the Internet-Draft "Performance Evaluation of Routing Protocol for Low Power and Lossy Networks (RPL)" [25]. As it is the case in the book, the Internet-Draft evaluates the performance of RPL based on data produced with the OMNET++ simulator and, in the same manner, the metrics evaluated are the path quality, the control plane overhead, the end to end delay, the coping with instability and the resource requirements. In contrast to the book, however, the Internet-Draft provides a high level of details both on the settings of the simulations as well as on the results. The simulations consider both indoor and outdoor node placements and movement patterns as well as the typical home automation scenario where most destinations are only two hops away. While the simulations show that RPL is not finding the shortest path in all cases, it seems that the results justify the usage of RPL in LLNs.

In the paper "A Performance Evaluation Study of RPL: Routing Protocol for Low Power and Lossy Networks" [26] the same authors as in the previously presented Internet-Draft did another evaluation of RPL. This paper, however, considers only one scenario which is slightly bigger than the networks in the Internet-Draft. Again, the OMNET++ simulator was used for generating the data. The results are not fundamentally different from the ones of the other simulations and can thus be seen as another confirmation of the design of RPL.

As already said, "Interconnecting Smart Objects with IP - The Next Internet" [1] by Jean-Philippe Vasseur and Adam Dunkels contains an evaluation as well. The results, which were obtained by using the OMNET++ simulator, coincide with the previously mentioned findings.

J. Ko et. al have implemented and evaluated RPL on the TinyOS platform and presented the results in "Evaluating the Performance of RPL and 6LoWPAN in TinyOS" [27]. Thereby they compare the performance of TinyRPL to the performance of the current default routing algorithm within TinyOS, CTP. They reach the conclusion that none of the two is offering a significantly better performance but the features RPL adds to the system, like providing roots down from the root to the nodes, make it preferable to CTP.

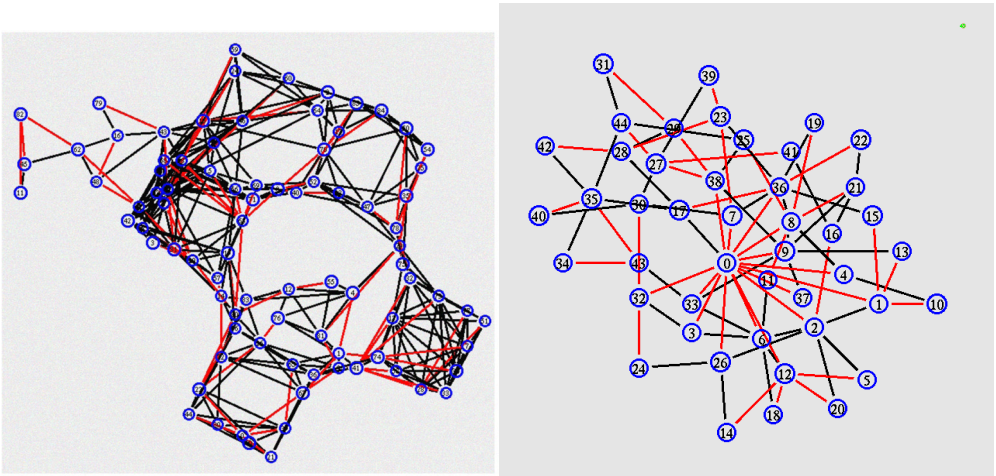
A very interesting practical evaluation is presented in "ContikiRPL and TinyRPL: Happy Together" [28]. When the two RPL implementations, with their individual rather good performance, interoperate, the resulting network is not nearly as optimal and reliable as a network out of only either of them would be. In the end the conclusion is reached that while both of them follow the standards in RPL as well as the standards in their lower stack, the small uncertainties in all the layers combined, from IEEE

802.15.4 over 6LoWPAN to RPL, yield enough tension points to slow down the network.

3.3 Simulation Results

This section is dedicated to presenting the simulation results from Joydeep Tripathi et. al [26] and [25] introduced in section 3.2.The simulations papers use networks of 45, 2242 and 86 nodes respectively which should cover most application scenarios from small environmental monitoring scenarios to automation of a full building floor. These results should help to build an understanding of the findings obtained when evaluating the Contiki-based network in section 5.

The network with 86 nodes from [26] can be seen in figure 3.1a and the one with 45 nodes, from [25], in figure 3.1b. A summary of the data can be seen in table 3.1.



(a) from [26] (b) from [25]

Figure 3.1: The Topologies

3.3.1 Hop Count

The hop count is simply the count from one node to every other node within the same network. The hop count can then be evaluated against the theoretical shortest path within the network. In the [25] an RPL routed packet could reach 90% of all nodes in 5 or less hops, while the optimal shortest path could do so in 4 or less. This shows that despite RPL being centered around root nodes, the impact of 1 hop more is not significant. The hop counts versus hop distances from the networks in figure 3.1 can be seen in figure 3.2.

Metric	Joydeep Tripathi et. al [26]	Joydeep Tripathi et. al [25]
Number of Nodes	86	45
Number of hops to reach 90% of the nodes using shortest distance routing	5	4
Number of hops to reach 90% of the nodes using RPL	8	5
Control traffic in the network compared to all traffic	1%	1%
Control traffic at a leaf of the overall traffic	10%	10%
Time till 90% of the nodes are reconnected with global repairs only	200s	220s
End to end delay to cover 90% of the nodes	N/A	0.05s

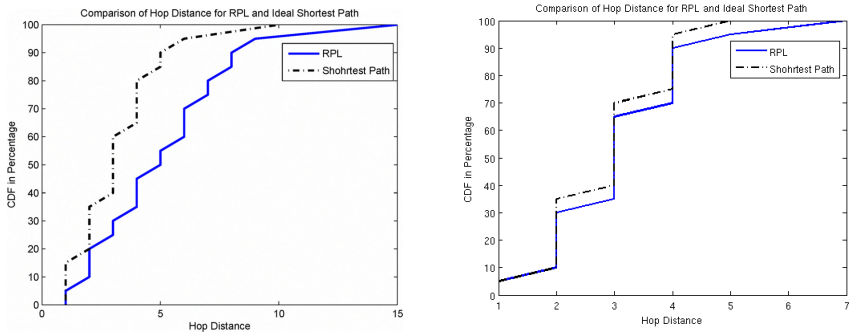
Table 3.1: The Results for the RPL Simulations from [26] and [25]

3.3.2 Control Packet Overhead

When considering the hardware behind LLNs, it is noticeable that the number of packets makes an impact on the battery life of the nodes. When looking at the network's topology and considering that DIO messages are transmitted in a broadcast fashion it is clear that the amount of control traffic is different on leaf and routing nodes. From the simulations the conclusion is reached that the amount of control traffic for non-leaf nodes is smaller than the amount of simulated data traffic, which generates 1 packet every 10 seconds. On leaf nodes the control data overhead highly depends on the amount of data the node creates as well as on the settings of the network, but in all cases control traffic overhead is more significant than on routing nodes. For the networks from the simulations, presented in figure 3.1, the control packet overhead based on the repair schemes is visualized in figure 3.3.

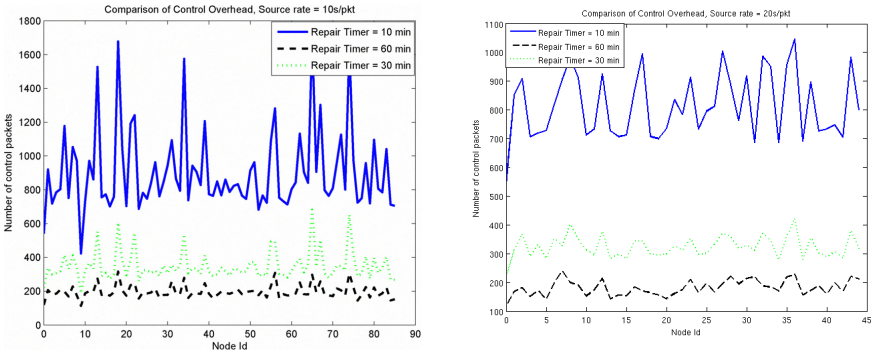
3.3.3 Routing Table Size

To have a comparison of the burden RPL puts on the different nodes, the simulations have recorded the average size of the routing table within the RPL nodes. This shows how much memory within a node is used to store this information. However, the size of the routing table is highly dependent on the node distribution pattern and on whether during the deployment phase one optimizes the IP addresses of the nodes according to the location so that a routing by a long prefix is possible.



(a) from [26] (b) from [25]

Figure 3.2: The Hop Count versus the Hop Distance



(a) from [26] (b) from [25]

Figure 3.3: The Control Packet Overhead based on the Repair Schemes

During the simulations, the number of routes in the routing table of most nodes amounts to less than 10 entries [25, 26]. This shows that they used a rather flat topology which was much spread out at the root nodes.

3.3.4 End to End Delay

Another aspect within a network is the end to end delay, especially in scenarios where users are involved and where they are accustomed to rather fast responses such as when pressing a light switch. The delay between two endpoints is mostly influenced by the hardware, for example by the transmission speed of the interface or the speed with which look ups on the routing table are performed. This already hints at the influence the routing algorithm has on the transmission speed. If the routing algorithm constructs a network with many hops or long routing tables on low powered devices, the performance of the network will decrease and the latency will increase.

However, the simulations rarely reached times of a second. Most of the

packets arrived at the destination after not more than 0.1 seconds, but this is connected to the number of hops the packet had to travel. The end to end delay for the network from [25] can be seen in figure 3.4.

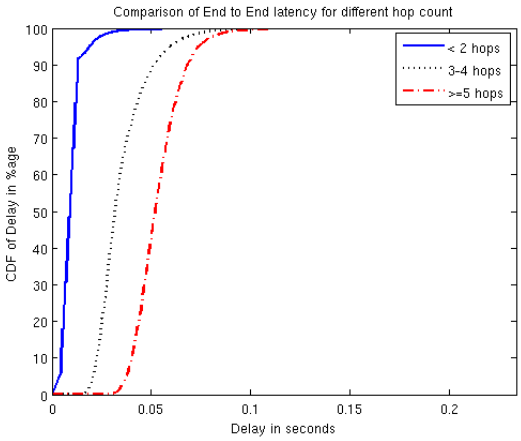
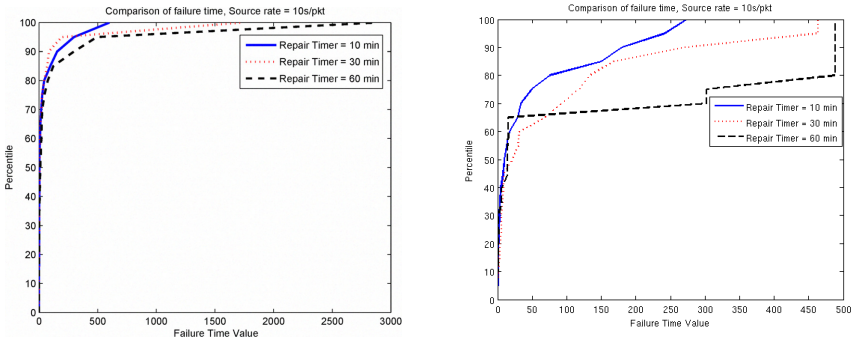


Figure 3.4: The End to End Delay for the Network from [25]

3.3.5 Loss of Connectivity

In the simulation a loss of connectivity is considered the state, where a node does not have any connections to parents or siblings and as such is unable to send packets towards the root node. As one can see from section 2.2.2, the loss of connectivity is greatly influenced by the global rebuild times and the local repair schemes applied. During the simulations, the results show that local repairs significantly improve the connectivity within the network while not noticeably increasing the amount of control traffic within the network. For the global repairs the conclusion is reached that the repair time influences the connectivity and the amount of control traffic to the point that it highly depends on the usage scenario and there can be no general rule on the time which should be used. In figure 3.5 the loss of connectivity based on the repair methods used during the simulations is visualized for the simulated networks.



(a) from [26] (b) from [25]

Figure 3.5: The Loss of Connectivity

Chapter 4

RPL MIB Module and its Implementation in Contiki

In section 17 of the "Routing Protocol for Low power and Lossy Networks" [4] a number of monitored and configurable objects were defined. These variables were then transferred within this research project into [12]. The structure of this Management Information Base can be seen in figure 4.1. The following chapter describes its design along the implementation in Contiki.

4.1 Implementation Details

The Management Information Base was used on top of the Contiki SNMP implementation presented in Section 2.3.3 to implement an SNMP monitoring solution for ContikiRPL. This section will talk in detail about the functions of the objects and the difficulties and restrictions faced during the implementation within ContikiRPL.

4.1.1 Write Access in Contiki

As indicated in figure 4.1, a number of attributes are marked as being writable in the MIB module. In general, it is possible to have write access when handling ContikiRPL variables and it can be enabled inside the source code of the MIB modules implementation. However, when a new DIO is received within the DODAG, all changes are reversed. Worse even is that most writes will be considered to be modifications to the existing DODAG and thus the DIO timer will be reset, triggering an increase of control traffic. Therefore write access is disabled by default.

If further restrictions apply to a variable in the MIB module, they will be mentioned in the respective subsection.

4.1.2 Testing and Verification

For the first testing, a single RedBee mote was used and its values were retrieved using the snmpget and snmpwalk commands from the NET-SNMP[29] collection. This allowed to confirm the testing of all data structures without having to deal with problems that might arise from a changing routing tree. Simultaneously, the results were outputted over the JTAG interface of the RedBee motes, allowing to confirm the data. By using malformed OIDs within a get-next request, the table handlers were tested on whether they find the correct next entry.

4.1.3 Additional MIB Modules

In addition to the RPL MIB, the routing table from the [30] has also been included. This allows access to the downwards roots, which are not handled by the RPL MIB module, as well.

4.2 MIB Module Parts

This section will follow the layout of the MIB module while explaining both its content and its implementation. The layout of the MIB module can be seen in figure 4.1.

4.2.1 General and Active RPL settings

The first two groups of objects within the RPL MIB show the general settings of the RPL implementation as well as of the currently active RPL instance and DODAG. The RPL DIS Mode setting indicates whether or not RPL should actively try to find a DODAG to join when the node powers up. This value is fixed at compile time using a special define statement which is also used within the MIB modules implementation to set the correct values. As the RPL DIS mode is fixed at compile time, it is not writable.

The second group indicates the active RPL Instance and the used DODAG. As Contiki only supports one instance, the values for the Active Instance and the Active DODAG are always the one of the only instance available. Further, as it is not possible to switch between different instances, there is no write access, preventing one from switching off the DODAG and thus damaging the routing tree.

Following, one can also find here the DAO Sequence number, which indicates the number of DAO messages sent to the root node, and the DAO Trigger Sequence Number, DTSN, giving the number of hops it takes to reach this node from the root node. The value for the current DAO sequence is provided by a global variable, made available through a minor modification of the RPL code. The value from the DTSN is provided by the active instance.

4.2.2 Objective Function Table

After the general information on RPL and the Active DODAG the table following below provides information on the available Objective Functions, identified by their Objective Code Points, OCP, as well as the function to switch them on and off. Unfortunately, in Contiki it is possible to compile only one Objective Function into the code. Thus, this Objective Function is the only one Contiki knows about and the only one available within the SNMP module. It also means that, when switching the Objective Function at compile time, one has to adapt the external variable within the SNMP code pointing to this Objective Function. Obviously the OCP is always marked as active.

4.2.3 RPL Instance Table

This table has been designed to describe the different RPL instances a node knows about. The instances are defined by their instance ID which is part of the OID indexing the values.

The first value, OCP, indicates the Objective Code Point and thus the Objective Function used within the RPL instance. It is important to notice that the OCP can be different from the ones available on the system, in which case the node will only be able to join the RPL instance as a leaf node and cannot act as a router.

Following the OCP comes the per Instance DIS mode. Similar to the general DIS mode RPL can be set to send or not send DIS messages when wanting to rejoin an already known RPL Instance. As it is with the global DIS mode, the result is set at compile time. Since Contiki has only one DODAG at a time, this value has little practical implication. As it is determined at compile time, the object is not writable.

After the Dis follow the DAO acknowledgment flag and the Mode of Operation, MOP. The MOP determines whether routes from the root down to a node should be stored and, if so, whether multicast groups should be stored as well. The DAO acknowledgment, on the other hand, determines whether these routes should be confirmed by the parent node or whether a sent DAO should be assumed to be received without an acknowledgment. Contiki does not support not acknowledging DAOs when downwards routes are maintained. Thus, both values are determined at run time using the mode of operation of the RPL Instance. While the MOP is writable, the option to acknowledge it is not, as it is determined by the MOP.

4.2.4 DODAG Table

As the name suggest, the DODAG table stores the information about all DODAGs known to the system. They are indexed by the RPL Instance ID the DODAG belongs to as well as by the DODAG ID. As RPL only stores

one DODAG at a time, this table also holds only one at the time on the Contiki nodes.

The DODAG version number and the Rank are the first two values in this table. The former is giving the current version of the DODAG, which is increased every time a global rebuild has occurred, as mentioned in section 2.2.2. The later gives the rank of the node within the DODAG.

The next column is taken by the DODAG state. The state of the DODAG describes whether the DODAG is a local DODAG or grounded with an RPL root node or associated with an RPL Instance. The difference between associated and grounded is that a predetermined root is creating the DODAG by itself in a grounded state while in the associated state the information regarding the instance variables is shared between multiple roots and set by a higher instance.

The DAO Delay in the following column is giving the time a routing node should wait for its children to send up DAO messages. In an ideal scenario, that would allow the routing node to congregate IP addresses into prefixes and thus shorten the routing tables.

Next is the preference of the DODAG, which denotes how preferable a DODAG is and hence whether a node should switch to it. While Contiki reads, writes and stores this setting, the value is ignored when Contiki is selecting a DODAG.

Following in the next column is the minimal increase a child must have from the rank of its parent. This helps to ensure that there is a constant increase between two steps.

The last column in the table holds the path control size. The path control size gives the size of the path control field. The path control field is used for ordering the preference of the parents within a DAO. This means that a node can only advertise it downwards routes to as many parents as the path control field can hold and thus as many as the path control size.

4.2.5 DODAG Parent Table

After the DODAG in the last table, the next object table lists the parents of the current node. They are indexed by the RPL instance and the DODAG they belong to as well as by their own IP address. The only object which is therefore shown in the table is the interface they can be contacted over. While the access to the parents is rather easy using Contiki's own list functions, it came to the attention that Contiki does not delete the parents from the list if they are worse then the maximum rank increase, but sorts them out when it comes to using them. While in a black box test ContikiRPL thus behaves as expected, it might create problems for the management system if it assumes that there are backup parents available while they are not. To deal with this situation and to create an additional ordering, an updated MIB might consider the inclusion of the parents' rank among the submitted

variables.

4.2.6 DODAG Child Table

Similar to the DODAG Parent Table, the DODAG child table is considered with the successors of the node. In the same fashion, the objects are again indexed using the RPL instance and the DODAG they belong to, as well as their own IP address. As they do not contain other objects, their IP address is also the only object in the table. In the next version, it should be considered to include the interface the child can be contacted over, as that might be helpful for monitoring border routers with multiple interfaces.

Contiki currently does not have child entries within RPL. Instead it is relying on a combination of the routing table and the neighborhood table to fulfill the needed tasks. Hence, the child table is not included in the current SNMP implementation.

4.2.7 DODAG Prefix Table

In the same manner, the DODAG prefix table is indexed by the RPL instance and the DODAG they belong to, followed by the actual prefix and their length. The prefix table allows to access the prefixes announced within an RPL DODAG. These prefixes may then be used for a stateless auto configuration of the IP address of the nodes.

ContikiRPL currently supports the usage of only one prefix which is then revealed to the user.

4.2.8 RPL Statistics

The last group of monitored objects is concerned with the error counters found within the RPL definitions. These are of the most important when finding errors within a routing tree. To enable access to the error counters one has to change the “RPL_CONF_STATS” flag within “rpl.h” to 1 which then enables the monitoring at the same time.

The first counter is monitoring the memory overflows within the RPL implementation. Memory overflows can happen when the node allocates new DODAGs, parents or routes.

The next counter is counting the time where no valid parent has been available to the node. This is especially useful as the default route allocated by RPL is always considered to be through a parent. Thus, no parent means no default route. ContikiRPL is instead counting the numbers of times it switched between parents, which could be considered sufficient, assuming that one cannot access the node if it is not connected to the root node.

Following is a counter which is not included in ContikiRPL, the counter for the wrong instance ID. When an RPL node receives a packet with a defined instance ID, as specified in [9], and it does not know about this

instance ID, the implementation should increase this counter to indicate that there is a node with a wrong topology within its neighborhood.

The two counters that follow are the ones giving an overview on the number of local and global repairs a node has experienced. As indicated in section 2.2.2, a global repair is triggered by the root node and starts a full rebuild of the whole RPL routing tree, while a local repair is initiated and executed by the node itself, whenever it finds that it has no more contact to any of its parents.

The last implemented counter is counting the parsing errors within the RPL messages. In Contiki's RPL implementation it only counts parsing errors found within the DIO and not within DIS or DAO messages. While the DIO messages are undoubtedly the most important messages when maintaining a routing tree, it should be extended to include incoming DAO messages as well.

The following two counters count the seconds an RPL node is left without a parent and the seconds it is without a parent while trying to forward a packet. In theory these counters allow some evaluation of whether the chosen Objective Function is creating a stable tree within the given environment or whether a change to it should be considered. Neither of the two have a matching variable within Contiki.

The last four counters are considered with the control bits found in [10] and thus should help to determine whether the node has encountered loops within its forwarding table. As Contiki does not include the header specified in [10] these counters are not included either.

Figure 4.1: Structure of the MIB Module

Chapter 5

Experimental ContikiRPL Evaluation

The following will present an evaluation of the RPL implementation of Contiki.

5.1 Installation

In general, the installation of Contiki with RPL on the Atmel Raven boards worked without problems. Only the activation in the contiki-conf.c file in the platform folder was needed. As the 2.4 release could not integrate well with the rest of the system, the git version from the 20th of April 2011 was used

In contrast to other platforms, RPL is not yet the default algorithm for the Raven USB stick. Thus, similarly to the Raven boards, RPL required an activation in the contiki-conf.c file in the platform folder. Further changes to the RPL ICMP files and the make files are needed in order to get it running. All of which are documented in contiki-conf.c. A large problem is the memory of the USB stick, as it does not allow the inclusion of the web server for visualizing the routes on the border router.

Not only does the memory size of the USB stick prevent the inclusion of the web server, but it does not even allow the inclusion of the TCP stack. During RPL operations, the Contiki installation on the border router is responsible for the routing and can in itself be contacted over the tun-slip interface. This is fundamentally different from the usage of routing advertisement messages, where the stick only converts IPv6 to 6LoWPAN and does not affect the routing.

The situation on the RedBee EconoTAGs was easier. RPL was activated by default and no changes to the ICMP files were needed. In comparison to the Raven USB stick, the RedBee EconoTAGs provided enough RAM to allow the inclusion of the web server application to show all nodes detected

within the routing tree as well as the neighborhood of the node.

5.2 Connecting RedBee EconoTAGs and Atmel Ravens

After installing ContikiRPL onto both RedBee EconoTAGs and Atmel Raven, both types of motes were turned on to test whether they would interconnect. While when using one type of nodes the RPL tree build rather fast, including multihop and meshed routing networks, it was not possible to build a routing tree using both nodes regardless which one acted as border router. When checking the neighborhood of the border router it came to the notice that the other nodes were not discovered here either. Despite the fact that the experiment could not provide an insight on whether RPL works when using multiple hardware platforms, it could still reveal that the problems occurred in the UIP and RIME layers, where packets from one hardware platform to another are detected as being malformed.

As a result of the initial analysis it was decided to use the RedBee EconoTAGs for the better border router they provide.

5.3 Routing Tree Setup

5.3.1 Routing Metrics

As presented in [19] and [23], one of the most traditional routing metrics has been the hop count. As such, one of the metrics used during the experiment will be the hop count between the border router and the respective node. Here the objective function is the parent node's rank plus one. While in the experimental analysis this approach was still taken, a modification was made to ensure the constant choice of primary and fall back parents. The metric therefore was increased more per hop on some nodes.

5.3.2 Mote Setup

For building up a working RPL tree it is not enough just to place the nodes next to each other, but one is required to move them one or multiple hops away. To reach that, the base station will be set up and one of the motes will be moved until it does not have a connection to the base station anymore. Then a node between the two other nodes will be inserted making the connection between the first two nodes. Therefore, one can be sure that the connection between the first node and the base station will be established over the second node. When adding further nodes the same algorithm can be applied. One simply switches off the nodes in the middle. In this

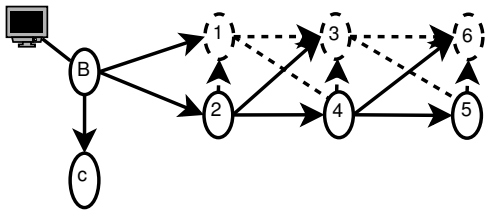


Figure 5.1: The Experiments Topology: B is the border router, dashed nodes have a modified objective function, dotted line indicates secondary routes, arrowheads indicate children relations

way, a mesh can be built up without facing the danger of having a directly connected network.

8 of the RedBee EconoTAGs were distributed in the building to allow a small scale testing of the RPL implementation. The distance between two hops was 12 meters. The 24 meters resulting between two nodes when skipping a hop was enough to ensure that a hop could not be skipped.

5.3.3 Routing Tree

The established topology can be seen in figure 5.1. The border router, connected to the monitoring computer, is marked red. Node c is a control node that was used to ensure that local repairs in the experimental subtree did not wrongly affect the remaining RPL tree. Dashed and solid nodes indicate the objective functions. While the basic objective function used was an increase by the minimal rank increase per hop, the dashed nodes were modified to give the rank a higher increase to ensure the same topology when repeating the experiment.

As one can see, the solid node is always the primary parent of the solid and dashed nodes in the next couple as well as the backup parent to the neighboring dashed node. A dashed node might be backup parent to the following nodes depending on its setting. For most of the experiments the increase was one unit higher than the minimum increase used on the solid nodes, which means the dashed node were parents. To ensure that a dashed node only becomes a backup parent when desired, the maximum increase of the rank is set to be two which is as high as the default increase of the dashed nodes. Whenever different settings were used it will be mentioned in the respective section.

It is also possible to confirm that in all cases ContikiRPL has built the routing tree as expected. It always saved multiple parents whenever they were available. However, as mentioned in section 4.2.5, it also saved parents that were not parents and ignoring them when switching parents.

5.4 ICMP Echo Reply Time

Firstly the time it takes for an ICMP echo request and reply to travel through the routing tree to a node and back was recorded. This number helps to see not only the speed reached within the network, but also the time per hop given RPL is used for routing.

In table 5.1 one can see the time an ICMP echo request and reply needs from the computer attached to the border router over one or more hops to the node. The results from nodes with the same number of hops from the border router were averaged. The experiment was repeated until the standard deviation was one tenth of the average or less and until at least 100 values have been collected. The ICMP packets did not contain any payload and were sent every second.

Destination	Round Trip Time	σ
border router	24ms	0.89ms
1&2	33ms	0.96ms
3&4	42ms	1.42ms
5&6	55ms	1.54ms

Table 5.1: ICMP Echo Reply Time from the Controlling Computer

From the times seen in table 5.1, one can conclude that the round trip time per hop is about 10ms and thus the time needed for one hop in one direction is 5ms.

5.5 DODAG Building Time

The next value to be considered is the time ContikiRPL needs to build up a working routing tree. We thereby consider three different scenarios. In the first one both the border router as well as the tree nodes are starting up at the same time. In the second scenario the border router will already be up when the tree starts. In the last scenario the nodes will already have started when the border router starts. Table 5.2 shows the time which is needed for building the routing tree in each of the scenarios for each pair of nodes. Again, nodes at the same distance from the border router were taken together. A timeout of 1 second between two empty ICMP echo requests was used.

When adding to the considerations that it takes about 5 seconds after the powering of the border router until it is reachable from the host computer, one can see that a tree is constructed in about 10 to 12 seconds per hop. Table 5.2 also shows that it makes no difference whether DIS messages are used or whether a node is waiting for the next periodic DIOs during the initial build.

Scenario	Node 1&2	Node 3&4	Node 5&6
complete cold start	17s	29s	40s
cold start of non border routers	12s	21s	32s
cold start of the border router	18s	29s	42s

Table 5.2: Tree Building Time

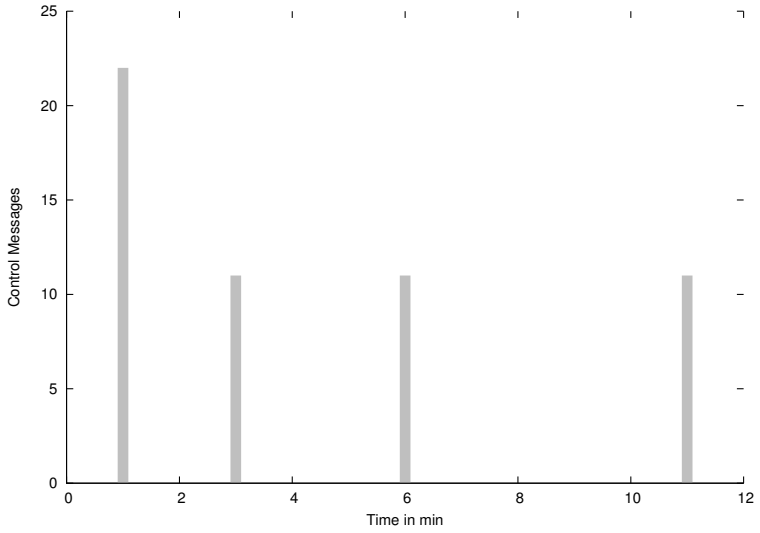


Figure 5.2: Number of RPL Control Messages received at the router

5.6 Control Messages

For building DODAGs, RPL is sending out DIS, DIO and DAO control messages. For efficiency reasons, RPL is decreasing the number of messages when the tree is stable. In figure 5.2 it is visualized how the control messages arrived at the border router. One can see that the time between the control messages is doubling and, while in a more lossy environment this might not yield the same results, it proves that RPL's concept of slowing down the control traffic in a stable setting is working.

5.7 Packet Reception Ratio

During the evaluation, the packet reception ratio became a focus point, since a high packet loss is one of the rather noticeable events. The current subsection covers the general operations after the routing has been established, checking whether it is free of any additional errors. Error handling and its impact on the packet reception will follow in section 5.8.

While it first looked promising, it soon proved to be more and more difficult to make a qualified remark about the packet reception ratio. In an

empty corridor the loss rate was at 0.5%, or out of 200 packets only one was lost. However, when a notebook on wireless LAN was moved through the corridor, packet loss rates of 50% could be observed. This merely confirms that our nodes are too close to each other to provide a sufficiently different fall back route.

In the comparison between a non-RPL enabled Contiki node and a ContikiRPL node running at the same time with one hop at the same distance to the base station, and thus suffering the same disturbances, none of them performed better or worse than the other. This is not unexpected, but it confirms that the RPL control traffic per hop is not affecting the quality of the link.

5.8 Repairs in ContikiRPL

After having looked first into the general performance of the ContikiRPL implementation, the following sections will address the performance of the local and global repair mechanisms. The focus will be first on the parent and sibling fall back as well as on the rebuild of the subtree of nodes conducting a repair. In the second part global repair parameters and the greediness are analyzed. The detailed repair procedures are thereby outlined in section 2.2.2.

5.8.1 Fall Back Parent

The first local repair procedure analyzed is the usage of a secondary parent. As one can see in section 5.3.3, the node 4 has node 2 as a primary parent and node 1 as secondary parent, while node 3 has node 4 as additional secondary parent.

To analyze the behavior of the nodes when the primary parent fails, node 2 was switched off while node 3 and 4 were monitored. The failure of node 2 should force both the other nodes to switch over to node 1 as their main parent.

For monitoring the time elapsed until node 3 and 4 had completed the switch over, both nodes were contacted using empty ICMP echo requests at 0.5Hz. Additionally, the parent switch counter, provided by ContikiRPL, was recorded before the disabling node 2 and after the reconnection using multiple SNMP requests, to ensure that the switch over had occurred as expected. Further SNMP values that were monitored as well were the DODAG version number and the local and global repair counter, as any change in these would have hinted to a repair instead of a simple parent switch over.

Relying on ICMP echo request to determine the timeout rather than on constantly polling SNMP provides two advantages. First of all, the messages are smaller and thus create less impact on the network and lower the chance of a malformed message due to external interference. Secondly, the ICMP

processing takes place fully within the UIP stack, meaning less message parsing and less processing time is needed, which could potentially influence the results. However, using SNMP would not change the time needed for the repair as all processing power consuming operations are done on the unreachable node.

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 5.3 shows the timeouts and run times and the respective standard deviation. As one can see, the average unreachable was around 3.5 minutes. Further it was also noticed that node 3 chose, as expected, node 1 as new primary parent and not node 4.

Monitored Value	Average	σ
ICMP run time before disconnect	43.4ms	0.8ms
ICMP run time after fall back	43.8ms	1.0ms
ICMP Timeouts	211s	19.4s

Table 5.3: Parent Fall Back Time

5.8.2 DIO Timer and Fall Back Parent

ContikiRPL provides multiple counters to take care of the maintenance of the routing tree. One of them, standardized in [4] are the different DIO trickle timers which take care of the periods in which regular DIO messages are send and the maximum increase of the period. As the increase is always done by doubling the previous value, it is sufficient to alter the DIO trickle start time.

To monitor the influence the timers have on the behavior of the nodes, the starting time was altered and the previous experiment from section 5.8.1 was repeated. As the ICMP run time has been shown to not change, only the timeout was measured. Again, SNMP was used to monitor DODAG version number and the local and global repair counters to ensure no repair taking place.

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

Table 5.4: DIO Timer and Fall Back Time

The results can be seen in table 5.4, which shows that the time it takes for a child to switch from one of the parents to the other one does not depend

on the time between two DIO message. It also means that the discovery of a failing parent is not affected by the timing of the DIO messages, but is solely done by the RIME stack of Contiki.

5.8.3 DAO Timeout and Route Lifetime

A counter that affects the timeout of the routes is the DAO expiration timeout giving the lifetime of the DAO messages and thus the routes. By default, Contiki sets the route lifetime to infinity, meaning that only when a DAO announcing a change is received the nodes will change the routes. If the parent is not responding to a DIO message, the lifetime is set to the DAO timeout, after which, if no new DAO is received, the route is deleted.

When knowing the time between route establishment and parent failure in advance, it is possible to generate very short timeouts. When dealing with predictably moving nodes, the lifetime of the default route and the DAO timeout can help to maintain a working routing tree. However, when dealing with static nodes failing due to energy problems or external influences, changing the route lifetime might have the opposite effect. As deleting routes requires reacquiring a new one, the nodes are forced to send additional messages which drain more energy. It also needs to be considered that in some scenarios, for example when having an energy-aware objective function, the objective function might increase the nodes' rank when routes should be disregarded. Thus, the objective function might provide a much more finely tuned system to handle energy critical situations.

5.8.4 Fall Back Sibling

The second fall back method provided by RPL is to change its parent connection to one of its siblings, see section 2.2.2 for details. To force a siblings connection, the rank increase for the dashed nodes in the setup, see section 5.3.3, was set to two, making it worse then the maximum parent rank. Then node 2 was switched off while node 4 was monitored. As node 4 thus had no working parent anymore, it had to perform a local repair and change its parentage.

Again ICMP echos were send to determine the time it took to find a working route. The RPL values for the DODAG version number and the local repair counter as well as the global repair counter were retrieved using SNMP to help to ensure that no global repair had taken place, but a local repair occurred.

The table 5.5 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. It is, however, noticeable that the timeout is only slightly longer than in 5.8.1, indicating that the time it takes to discover a broken parent is of a much higher

significance than any repair conducted.

Monitored Value	Average	σ
ICMP run time before disconnect	43.5ms	0.8ms
ICMP run time after fall back	43.6ms	0.9ms
ICMP Timeouts	236s	13ms

Table 5.5: Sibling Fall Back Time

Looking closer at the nodes by using the JTAG interface, the exchange of an additional DIS message from node 4, requesting to join the DODAG, and the DIO message sent by node 1 with the information about its rank and DODAG, could be held responsible for the additional delay. The DAO exchange was the same as for falling back to the parent. This could also be confirmed when reading the DAO counter over SNMP, with its increase being the same in both cases. Both observations hold true only if one ignores cases with message drops and other external influences which were observed in both experiments.

5.8.5 Poisoned Tree

As explained in section 2.2.2, poisoning the subtree is important when trying to prevent loops during local repairs. Thus, the following experiment will test whether or not ContikiRPL correctly performs the route poisoning.

The previous experiment has been repeated to check whether the poisoning of the routes properly triggered a local repair. Therefore nodes 3, 4 and 5 were monitored. To check the time needed for the local repair to be completed and the message run times before and afterwards, ICMP echo messages were sent to all targets. As three nodes were contacted this time, the timeout was increased from the previous one second to two seconds between two ICMP messages. To verify the repair process, the repair counters for both the global and the local repairs on these three nodes as well as their DODAG ranks were monitored using SNMP agents.

Using SNMP it was observed that the local repairs counter on node three, four and five increased. The data from the ICMP round trips can be seen in table 5.6, which again illustrates the speed of the local repair after the nodes receive a poisoned rank message. The increase in the local repair counters as well as the timeout length of node five confirms that the poisoning of the tree took place as expected by nodes three and four.

In a second step, node two was turned on again to confirm that Contiki is not poisoning its subtree wrongly as it should only do so if it cannot maintain its current or a higher rank. A behavior that could be confirmed upon retrieving the counters.

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

Table 5.6: Poising the Sub DODAG

5.9 Greediness of Nodes

A further problem when considering the difference between parent changes and local repairs within a routing tree is that a node may be trying to optimize itself to have as many parents as possible to switch faster between them. To control how Contiki handles the greediness of the nodes, node one, two and four were switched on in different order, with the rank increase of node one being 2 again, and the status of the RPL parents recorded in node four as well as the rank of the node being monitored using the SNMP agent running on it.

When switching on the nodes in the sequence 2, 4 and then 1, node 4 correctly set its rank to two and had chosen node 2 as its parent. When switching them on in the order 1, 4, 2, the same rank could be observed and node 2 was chosen as parent again. However, the parent table still contained node 1 as a parent and only upon the failure of node 2 did node 4 discover that node 1 was too bad, discarded it and triggered a local repair. While this behavior means that it looks like a properly behaving node from the outside, it makes automated monitoring difficult as the internal state is not consistent with the expected behavior.

To sum it up, ContikiRPL is behaving as expected, optimizing for the best parent while not being greedy about the number of parents. It is also shown that, while ContikiRPL does not forget about previous parents, it does not use them if their rank is too bad.

5.10 Rebuild Build Memory

The last step is focusing on the global rebuild of the DODAG. During a rebuild a new DODAG is constructed solely based on the objective function. The objective function may thereby consider the state of the DODAG and other connection-related variables such as the stability of the route. However, the actual rebuild process must not include them. As a function solely based on the hop count was chosen, the global rebuild should not consider the old state at all.

Therefore, the DODAG rebuild is triggered multiple times and the preferred parents and rank of each nodes were recorded using SNMP. To test whether the rebuild occurs without the usage of the memory, the rank in-

crease is set to three times the minimum hop rank increase, while the solid nodes change their rank increase. For odd DODAG versions the solid nodes used an increase of the minimum hop rank increase, while for even ranks the increase was five times the minimum hop rank increase.

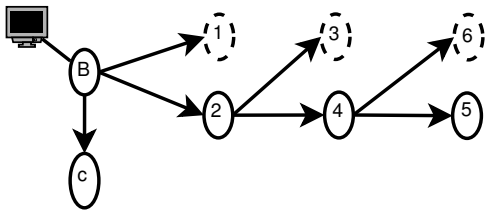


Figure 5.3: Rebuild Build Memory Topology 1: Solid nodes with rank plus one and dashed plus three

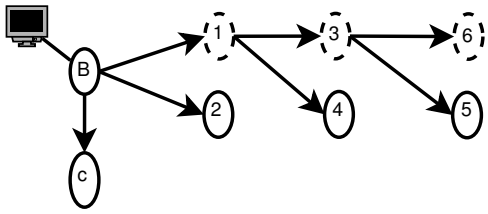


Figure 5.4: Rebuild Build Memory Topology 2: Solid nodes with rank plus five and dashed plus three

The resulting typologies can be seen in figures 5.3 and 5.4, which confirm the expected results. The DODAG rebuild happened without the consideration of the previous DODAG, based only on the objective function. The experimental findings also confirm the results of the greediness analysis, as in all cases the best parent was chosen.

Chapter 6

Conclusions

In the implementation part of this thesis ContikiRPL has been successfully connected to our SNMP implementation using the MIB module updated within this thesis and while ContikiRPL does not allow write access to the variables as defined in the MIB module, read access works flawless within the transmission limits set by lossy networks. With the help of the SNMP implementation it was possible to show a few limitations within ContikiRPL. The most important of these is the storage of non-parents in the parents table. The missing of the child table is the second most pressing problem. While at a first glance the table can be substituted by the routes table, it, nevertheless, does not give the option to associate a hop with a particular DODAG.

In the experimental and analytic part of this project it was possible to show that the RPL repair mechanisms work as required by [4]. Special care has to be taken only about the neighborhood discovery, as the time it takes to discover failures is the only restriction within the process. Considering the two to four messages needed to conduct a local repair, it is clear that global repairs, with 11 messages at the root node, are not comparable in terms of efficiency.

Seeing the little time difference between a parent switch and a local repair, it might make sense not to store additional parents on nodes with very limited memory. However, this would mean that the tree is poisoned for every switch and it thus would be a trade off between memory and energy available.

In terms of the time it takes to repair the DODAG, keeping in mind the network had most of the traffic coming from the root node as it might be found in a building automation scenario, most of the consideration has to be given to the neighborhood discovery mechanisms. This is especially valid when considering that users are expecting immediate reactions to controls, such as a light switch.

To sum it up, it was possible to use SNMP and ICMP tools to confirm

that ContikiRPL's error correction algorithms work as expected. It was also possible to show that the right choice of neighborhood discovery and objective function remains a difficult matter and that in another scenario the results might be different.

It remains a question whether the approach of the child node choosing its parent is preferable to a method where the routing node chooses its preferred path between two possible paths, at least given the setting with the root-based traffic.

Bibliography

[1] Jean-Philippe Vasseur and Adam Dunkels. *Interconnecting Smart Objects with IP - The Next Internet*. Morgan Kaufmann, 2010.

[2] N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919 (Informational), August 2007.

[3] J. Martocci, P. De Mil, N. Riou, and W. Vermeylen. Building Automation Routing Requirements in Low-Power and Lossy Networks. RFC 5867 (Informational), June 2010.

[4] Tim Winter, Pascal Thubert, Anders Brandt, Thomas Clausen, Jonathan Hui, Richard Kelsey, P Levis, Rene Struik, and JP Vasseur. RPL: IPv6 Routing Protocol for Low power and Lossy Networks. Internet-Draft (Standards Track), March 2011.

[5] Nicolas Tsiftes, Joakim Eriksson, and Adam Dunkels. Low-power wireless ipv6 routing with contikirpl. In *IPSN '10: Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 406–407, New York, NY, USA, 2010. ACM.

[6] Siarhei Uladzimiravich 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, 2010.

[7] J. Hui and P. Thubert. Compression Format for IPv6 Datagrams in Low Power and Lossy Networks (6LoWPANs). RFC 6282 (Proposed Standard)(Awaiting Approval), 2011.

[8] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard), September 2007.

[9] J. Hui, JP. Vasseur, D. Culler, and V. Manral. An IPv6 Routing Header for Source Routes with RPL. Internet-Draft (Standards Track), March 2011.

[10] J. Hui and JP. Vasseur. RPL Option for Carrying RPL Information in Data-Plane Datagrams (Work in Progress). Internet-Draft (Standards Track), March 2011.

[11] D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411 (Standard), December 2002. Updated by RFCs 5343, 5590.

[12] K. Korte, J. Schönwälder, A. Sehgal. Definition of Managed Objects for the IPv6 Routing Protocol for Low power and Lossy Networks (RPL) (Work in Progress). Internet-Draft (Standards Track), March 2011.

[13] The Contiki Operating System - Home. http://www.sics.se/contiki/.

[14] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)*, Boulder, Colorado, USA, November 2006.

[15] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Workshop on Embedded Networked Sensors*, Tampa, Florida, USA, November 2004.

[16] uIP. http://www.sics.se/ adam/uip/index.php.

[17] Atmel. http://www.atmel.com/.

[18] Redwire. http://www.redwirellc.com/store/node/1.

[19] P. Levis Stanford University, A. Tavakoli, and S. Dawson-Haggerty UC Berkeley. Overview of Existing Routing Protocols for Low Power and Lossy Networks (Work in Progress). Internet-Draft (Informational), April 2009.

[20] K. Pister, P. Thubert, S. Dwars, and T. Phinney. Industrial Routing Requirements in Low-Power and Lossy Networks. RFC 5673 (Informational), October 2009.

[21] A. Brandt, J. Buron, and G. Porcu. Home Automation Routing Requirements in Low-Power and Lossy Networks. RFC 5826 (Informational), April 2010.

[22] Baumann, Rainer and Heimlicher, Simon and Strasser, Mario and Weibel, Andreas. A Survey on Routing Metrics, 2007.

[23] Ioannis Broustis, Gentian Jakllari, Thomas Repantis, and Mart Molle. A comprehensive comparison of routing protocols for large-scale wireless manets. In *IEEE International Workshop on Wireless Ad-hoc Networks (IWWAN)*, 2006.

[24] Di Wang, Zhifeng Tao, Jinyun Zhang, and A.A. Abouzeid. RPL Based Routing for Advanced Metering Infrastructure in Smart Grid. In *2010 IEEE International Conference on Communications Workshops (ICC)*, pages 1–6, Capetown, 2010. IEEE.

[25] Joydeep Tripathi, Jaudelice Oliveira, and JP Vasseur. Performance Evaluation of Routing Protocol for Low Power and Lossy Networks (RPL) (Work in Progress). Internet-Draft (Informational), January 2011.

[26] Joydeep Tripathi, Jaudelice Oliveira, and JP Vasseur. A Performance Evaluation Study of RPL: Routing Protocol for Low Power and Lossy Networks. In *2010 44th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6, Princeton, NY, USA, 2010. IEEE.

[27] JeongGil Ko, Stephen Dawson-Haggerty, Omprakash Gnawali, David Culler, and Andreas Terzis. Evaluating the Performance of RPL and 6LoWPAN in TinyOS. In *Workshop on Extending the Internet to Low power and Lossy Networks*, 2011.

[28] JeongGil Ko, Joakim Eriksson, Nicolas Tsiftes, Stephen Dawson-Haggety, Andreas Terzis, Adam Dunkels, and David Culler. ContikiRPL and TinyRPL: Happy Together. In *Workshop on Extending the Internet to Low power and Lossy Networks*, 2011.

[29] NET-SNMP. http://www.net-snmp.org/.

[30] B. Haberman. IP Forwarding Table MIB. RFC 4292 (Proposed Standard), April 2006.