Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea in Ingegneria delle Telecomunicazioni

A Multi-Hop 6LoWPAN Wireless Sensor Network for Waste Management Optimization

Laureando

Dario Cassaniti

Relatore

Prof. Michele Zorzi

Master Thesis in cooperation with

Fraunhofer Fokus Institute

Anno Accademico 2011/2012

I would like to dedicate my dissertation to my parents, my brother and my whole family, who supported me during my studies. They understood my dreams, my needs and my problems as no one else could do better. They believed in me and they allowed me to make very important experiences such as the Erasmus at the Technische Universität Hamburg Harburg and my Internship at the Fraunhofer Fokus Research Institute in Berlin which characterized not only my professional profile, but also my personal developement. With regard to my experience at the Fraunhofer Fokus Institute, I want to thank the Rescon (Resource Optimized Networks) team, in particular Dr. Thomas Luckenbach, who gave me this challenging opportunity of being part of the research group on the Outsmart European FP7 Project; Mrs. Gabriele Goldacker, who supervised me patiently during my six months internship and who gave me precious hints about how to organize, analyze and describe my research; Björn Riemer for his technical support regarding the Contiki Operative System and the creation of the platform for the AVR Atmel Atmega modules on this OS; Janis Sarikas for his studies on the sensors available on the market and his work on the point-to-point code between the waste basket sensor and the computer.
I also want to thank my supervisor, Prof. Michele Zorzi of the University of Padova, who gave me the opportunity to do an internship in a Research Institute abroad and to work on my Master Thesis about this project without any restriction or obstacle, but simply giving me suggestions and support during my work.
Moreover, I would like to thank my colleagues of the University of Padova and from the Technische Universität Hamburg Harburg, with which I shared moments of happiness, excitement, anxiety as well as common interests, hopes and hunger for knowledge.
Last but not least, a big thank to my friends of Padova, Hamburg and Berlin, with which I had many amazing times in my leisure.

Dario Cassaniti

Contents

1 Introduction **1**
1.1 OUTSMART Project - The Berlin Cluster 1
1.1.1 Research Goals 2

2 Wireless Sensor Network Technologies **5**
2.1 The ISM Band 6
2.2 Topologies 8
2.2.1 Reliability 10
2.2.2 Energy Efficency 11
2.2.3 Scalability and self-organization 13
2.2.4 Data latency 14
2.3 Power Consumption and Duty Cycle 14
2.4 IEEE 802.15.4 Standard 16
2.4.1 IEEE 802.15.4 Protocol Standard 17
2.4.1.1 Frame Format 18
2.4.1.2 Security 19
2.4.2 ZigBee 21
2.4.2.1 Network Layer (NWK) 23
2.4.2.2 Application Layer (APL) 25
2.4.2.3 Energy consumption 26
2.4.2.4 Security in ZigBee 26
2.4.3 6LoWPAN 28
2.4.3.1 6LoWPAN Network Architecture 28
2.4.3.2 6LoWPAN Protocol Stack 30
2.4.3.3 6LoWPAN packet format 30
2.4.3.4 Addressing 32
2.4.3.5 Header compression 35
2.4.3.6 Fragmentation and reassembly 37

2.4.3.7 Bootstrapping, Neighbor Discovery and Security . . . 38
2.4.3.8 Node, router and edge router operations . . . 42
2.4.3.9 Security in 6LoWPAN . . . 43
2.4.3.10 Mobility . . . 44
2.4.3.11 Routing . . . 49
2.4.3.12 Border Routing . . . 51
2.4.3.13 Application protocols . . . 53
2.4.4 IEEE 802.15.4a - Ultra Wide band . . . 55
2.4.4.1 IEEE 802.15.4a Frame Structure . . . 55
2.4.5 Wireless Hart . . . 57
2.5 Z-Wave . . . 59
2.5.1 Topology and Routing . . . 60
2.5.2 Security . . . 60
2.6 Bluetooth and Bluetooth Low Energy . . . 61
2.6.1 Bluetooth Low Energy . . . 64
2.7 Wavenis . . . 66
2.7.1 Wavenis Operation . . . 68
2.8 Dash7 . . . 69
2.8.1 Relation between frequency and distance range . . . 70
2.8.2 Drawbacks of the 433MHz ISM Band . . . 71

3 Contiki Operative System 2.5 73
3.1 System overview . . . 74
3.2 Kernel Architecture . . . 75
3.3 Events . . . 75
3.4 Processes . . . 76
3.5 Two level scheduling hierarchy . . . 77
3.6 Services . . . 77
3.7 Libraries . . . 78
3.8 Communication support . . . 79
3.9 The uIP TCP/IP stack . . . 80
3.10 RIME . . . 83
3.11 6LoWPAN Implementation . . . 84
3.12 Preemptive multi-threading . . . 85
3.13 Code size . . . 86
3.14 Contiki RPL . . . 86

4 RPL Routing Protocol 91
4.1 Protocol Overview . . . 91
4.2 RPL Instances . . . 95

4.3 ICMPv6 RPL Control messages 96
4.4 Sequence counter 100
4.5 Upward routes 101
4.6 Downward routes 103
4.7 RPL Security 106
4.8 Packet forwarding and loop avoidance 107
4.9 Maintenance of routing adjacency 108
4.10 Objective Functions 109
4.11 Initialization Mode 109

5 A Multi-Hop WSN implementation with ContikiRPL 111
5.1 Introduction 111
5.2 Hello World example 112
5.3 IPv6 Udp Sender & Receiver 113
5.4 RPL Collect: Compiling Sink node & Sender node 116
5.5 RPL Border Router 121
5.6 Multi-hop UDP - RPL Collect 129
5.6.1 Collect View 133
5.7 Multi-hop WSN for Outsmart: Sensor data collection 135
5.7.1 Level of fullness detection 135
5.7.2 Multi-Hop Sensor Network Tests 139

6 Conclusions and future implementations 145

7 Appendix 147

Bibliography 155

Abstract

In the first part of this Master Thesis several Wireless Sensor Network technologies, including the ones based on the IEEE 802.15.4 Protocol Standard like ZigBee, 6LoWPAN and UltraWideBand, as well as other technologies based on other protocol standards like Z-Wave, Bluetooth and Dash7, are analyzed with respect to relevance and suitability with the Waste Management Outsmart European FP7 Project. A particular attention is given to the parameters which characterize a Large Scale Wireless Sensor Network for Smart Cities, due to the amount of sensors involved and to the practical application requested by the project.
Secondly, a prototype of sensor network is proposed: an Operative System named Contiki is chosen not only for its portability on different hardware platforms, but especially for its Open Source license, for the use of the 6LoWPAN protocol and for the implementation, in the last version, of the new RPL routing protocol. The Operative System is described in detail, with a special focus on the uIP TCP/IP stack and on the 6LoWPAN and RPL implementation. With regard to this innovative routing protocol designed specifically for Low Power Lossy Networks, chapter 4 describes in detail how the network topology is organized as a Directed Acyclic Graph, what is an RPL Instance and how downward and upward routes are constructed and maintained.
With the use of several AVR Atmel modules mounting the Contiki Operative System a real wireless network is created and, with an Ultrasonic Sensor, the filling level of a waste basket prototype is periodically detected and transmitted through a multi-hop wireless network to a sink node.

Chapter 1

Introduction

The optimization of the waste management has always been a challenging problem, especially in big cities and State capitals like the city of Berlin. The knowledge not only of the level of fullness of every single waste basket, but also of the weight of the whole content and the knowledge of the kind of objects inside the waste baskets, can give a remarkable improvement in the waste management quality, in the safety of the dustmen and in the cleanness and tidiness of the city. Moreover, an accurate statistical study of the frequency of usage of every single waste basket can give interesting information about the typical use not only of a single waste bin, but of the whole surrounding area.
The use of a Wireless Sensor Network for the waste management optimization would lead to a remarkable improvement in the waste collection route plan in order to avoid the accumulation of trash around the basket when it becomes full (which might last for a few hours or even for several days), as well as in order to avoid a frequent route to the areas where the public waste bins are rarely full. A route plan optimization leads, in the long term, to a considerable save of gasoline (and consequently pollution), but also to a remarkable improvement of the city cleanness and, in general, in the quality of life of the city.

1.1 OUTSMART Project - The Berlin Cluster

Outsmart is an FP7 European project which goal is to contribute to the Future Internet by aiming at the development of five innovative ecosystems (clusters) in urban areas. The Berlin cluster is one of these and it has to do with the waste management optimization with the use of a wireless sensor

network. This cluster is expected to develop the platform and main building blocks for data collection, preprocessing and provision of information regarding the waste management in the city of Berlin. This will include the specifications of the architecture from sensor/actuator nodes and networks up to the IP world.
The Fraunhofer FOKUS Institute in Berlin is responsible for the Work Package 4 of the Berlin cluster: this work package is supposed to cover all the aspects related to sensor technologies and sensor/actuator networks required to reach the overall goals of the Outsmart project. Moreover, Work Package 4 is supposed to analyze routing protocols over wireless lossy links in order to enhance end-to-end IP networking with an energy-efficient transmission mechanisms and is supposed to analyze auto-configuration, device management and node failure handling requirements in the selected application scenarios. Main aspect in the work of Work Package 4 is the integration of carefully selected communication mechanisms and protocols of existing state-of-the-art technologies considered appropriate for supporting scalable Future Internet applications based on Wireless Sensor Network.

The Fraunhofer FOKUS Institute is working on the Berlin cluster of the Outsmart project side by side with another partner of the project: the Berliner Stadtreinigung (BSR - the Berlin City Cleaning Services at a glance), a public company fully owned by the State of Berlin, which is the largest municipal refuse collection service provide in Germany, covering an area of $890km^2$ and 3.4 millions of inhabitants. The information provided by the BSR company regarding the location of the four waste management depots, the five regional centers of area cleaning and the 15 recycling depots across the whole city of Berlin (Figure 1.1) were very useful to get an idea about the problem as well as the information about the location of the 20.000 waste bins and the concession of a waste basket in order to test some prototypes.

1.1.1 Research Goals

Scope of this research was first of all to analyze the characteristics of the wireless sensor network technologies available in the market, with a special focus on the energy efficiency, the scalability, the reliability, but also on the routing protocols adopted and the topologies supported. In particular the IEEE 802.15.4 standard (which defines only the physical layer and the MAC layer of low power networks) and the technologies which uses this standard as lower part of their stack, like ZigBee, 6LoWPAN, Ultra Wide Band and Wireless Hart were studied, as well as other technologies like Z-Wave, Dash7 and Bluetooth.

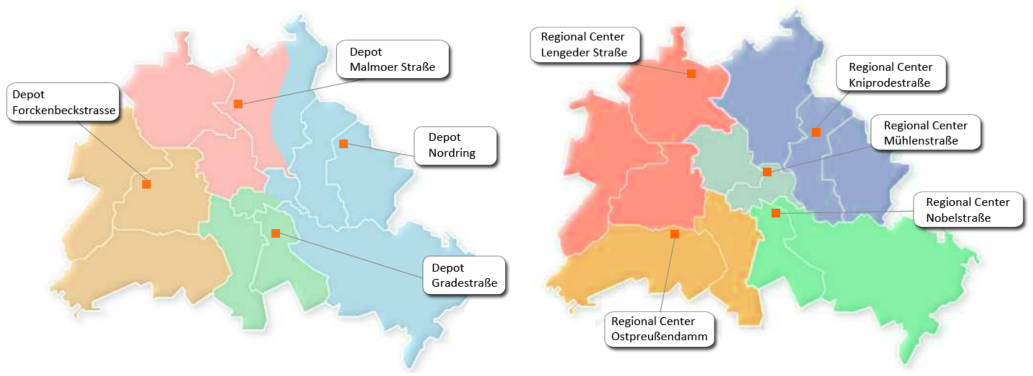

Depot Forckenbeckstrasse
Depot Malmoer Straße
Depot Nordring
Depot Gradestraße
Regional Center Lengeder Straße
Regional Center Kniprodestraße
Regional Center Mühlenstraße
Regional Center Nobelstraße
Regional Center Ostpreußendamm

Figure 1.1: The BSR Depots and Regional Centers location in the city of Berlin

Secondly, a prototype of a wireless sensor network was developed: after some discussions with the other researchers in the Fraunhofer FOKUS Laboratories regarding the possible sensors which could be used for detecting the filling level of the waste basket, a wireless network implementing multi-hop was created in order to save energy for the transmission to a central node and to reduce the complexity - and consequently the cost - of every single transmitter. An Open Source operative system (Contiki OS) was chosen for different reasons, including the introduction, in the last version of the operative system, of an innovative routing technique named RPL, a routing protocol designed specifically for Low Power Lossy Networks based on the concept of the Directed Acyclic Graph and perfectly integrated with the 6LoWPAN standard.

Chapter 2

Wireless Sensor Network Technologies

A wireless sensor network (WSN) consists of spatially distributed and autonomous sensors that cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants. Wireless sensor network applications typically require network components with average power consumption that is substantially lower than currently provided in implementations of existing wireless networks. The overall cost plays a fundamental role in applications adding wireless connectivity to inexpensive or disposable products, and for applications with a large number of nodes in the network. In order to meet this objective, the communication protocol and network design must avoid the need for high-cost components. The network should be ad hoc and capable of self-configuration and self-maintenance in order to be a true low-cost system. An "ad hoc" network in this context is defined as a network without predetermined physical distribution, or logical topology, of the nodes. "Self-configuration" is defined to be the ability of network nodes to detect the presence of other nodes and to organize into a structured, functioning network without human intervention. "Self-maintenance" is defined to be the ability of the network to detect and recover from faults appearing in either network nodes or communication links, again without human intervention.
The Outsmart project needs to take into account, in order to create an efficient wireless sensor network for the waste management, the fact that in the city of Berlin over 20000 waste bins are present. Therefore, it is necessary to consider a specific WSN sub-field named Large Scale Wireless Sensor Networks, where several thousands of nodes are distributed in hundreds of square meters. The small communication range of sensor nodes (in general between 10 and 100m), due to the necessity to prolong battery life by reducing energy consumption, makes the use of a multi-hop communication

necessary in order to build a Large Scale Wireless Sensor Network.

2.1 The ISM Band

Most of the wireless sensor network technologies operate in one (or more) of the ISM band(Industrial, Scientific and Medical radio band). These frequencies were internationally reserved for the use of RF electromagnetic fields for industrial, scientific and medical purposes other than communications.
The ISM bands are defined by ITU-R in 5.138, 5.150, and 5.280 of the Radio Regulations. Because communication devices using the ISM bands must tolerate any interference from other ISM equipment, these bands are typically given over to uses intended for general licensed operation, since they need to be tolerant of interference from other devices anyway. A table with the ISM bands and a list of typical uses for each frequency range follows:

Start Frequency	**End Frequency**	**Typical Uses**
6.765 MHz	6.795 MHz	Radio broadcast Weather report Aeronautical radio service
13.553 MHz	13.567 MHz	Radio services Telecommunications (PTP) Remote control system
26.957 MHz	27.283 MHz	Used by CB
40.66 MHz	40.70 MHz	Television broadcaast Telemetry/remote control
433.66 MHz	434.79 MHz	Amateur radio services Backscatter rfid system Baby intercoms Relemetry transmitter Cordless headphones - microphones

868 MHz	869 MHz	Short range devices RFID applications
902 MHz	928 MHz	Not available in Europe
2.4 GHz	2.5 GHz	Backscatter (RFID) Telemetry transmitter WiFi/Bluetooth Skype Phones Car Alarm
5.725 GHz	5.875 GHz	Amateur radio Radiolocation services Movement sensors (door openers) Contactless toilet flushing Backscatter RFID system
24 GHz	24.25 GHz	Amateur radio Radio location services Earth resources services via satellite Movement sensors Directional radio system for radio transmission
61 GHz	61.5 GHz	
122 GHz	123 GHz	
244 GHz	246 GHz	

These bands have been shared with general licensed error-tolerant communications applications such as Wireless Sensor Networks in the 865 MHz, 915 MHz and 2.450 GHz bands, as well as wireless LANs and cordless phones in the 915 MHz, 2.450 GHz, and 5.800 GHz bands. Because these devices are already required to be tolerant of ISM emissions in these bands, low power uses are generally able to operate in these bands without causing problems for ISM users. The ISM bands are also widely used for Radio Frequency Identification (RFID) applications with the most commonly used band being the 13.56 MHz band used by systems compliant with ISO/IEC 14443, including those used by biometric passports and contactless smart cards.

Most of the wireless sensor network technologies suitable for the OUTSMART project in order to realize a scalable, energy efficient and marketable sensor network in urban environments use the ISM radio bands to create a WPAN (Wireless Personal Area Network).

Technology	**Frequencies**
IEEE 802.15.4 - ZigBee/6LoWPAN	868 MHz 915 MHz 2,4 GHz
ZWave	868 MHz 915 MHz
IP500 Alliance	868 MHz 915 MHz
Wavenis	433 MHz 868 MHz 915 MHz
Bluetooth - Bluetooth Low Energy	2.4 GHz
KNX	868 MHz
UMTS/3GPP developments (LTE)	1800 MHz

Each of these frequencies is regulated by the European Communication Office in accordance with the Radio and Telecommunications Terminal Equipment (R&TTE) Directive (1999/5/EC) of the European Commission. The main restrictions in the ISM band rules are the transmit power, which influences the maximum distance achievable, and the duty cycle, which influences the time that can be used for transmission by each node.

2.2 Topologies

The basic issue in communication networks is the transmission of messages to achieve a prescribed message throughput and Quality of Service (QoS). QoS can be specified in terms of message delay, message due dates, bit error rates, packet loss, economic cost of transmission, transmission power, etc.

Depending on QoS, the installation environment, economic considerations, and the application, one of the several basic network topologies may be used. A communication network is composed of nodes, each of which has computing power and can transmit and receive messages over communication links, wireless or cabled. The basic network topologies are shown in Figure 2.1 and include fully connected, mesh, star, ring, tree, bus. A single network may consist of several interconnected subnets of different topologies.

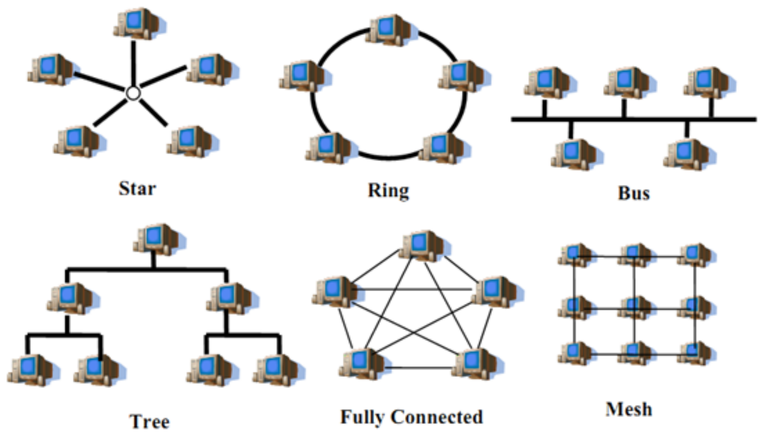


Figure 2.1: Different topologies

The most common topologies in wireless sensor networks are the star, mesh and tree networks. The star networks are a very simple and common form of communication topologies. The star topology organizes all the peripheral nodes around the central hub (sink), which is logically (and sometimes physically) at the center of the network. The central hub can be either the base station itself or a gateway node that is in direct communication with the base station. A natural and logical extension of the star topology is the tree topology, where the sink node is the root and nodes at different levels in hierarchy are connected via direct links.
The mesh networks are multi-hop local area networks, in which each sensor node not only sends and receives its own message, but also works as a router to relay messages for its neighbors through the network. Mesh topology facilitates multiple communication paths from the sensor nodes to the base station. A special case of mesh is the grid topology.
In clustered hierarchical topology, all nodes in the Wireless Sensor Network are joined at the lowest level. The sensors use their local neighborhood information to form a set of clusters and elect a cluster head (CH) for each cluster. The CH election process can be based on various parameters such as available energy resources, proximity to the base station, and number of

neighbors. The CH in the lowest level are arranged into clusters in a higher level, and a CH is assigned for each cluster at this level. The process is repeated for each level in the hierarchy. The number of hierarchical levels depends on several criteria, including coverage requirement, deployment region, node density, and transceiver sensing range. The clustered hierarchical architecture maintains a tree rooted at the sink node, with a hierarchy of CH as the internal nodes and the sensor nodes as leaf nodes of the tree, for network addressing and organization. Nevertheless, different from the tree topology, it still maintains the multi-hop mesh routing for actual data communication. While this hierarchical scheme is not optimal for traditional networks, it is still effective for Wireless Sensor Networks because most of the communication is directed from the sensor nodes to the sink or vice versa, and the higher level CH are usually closer to the sink.

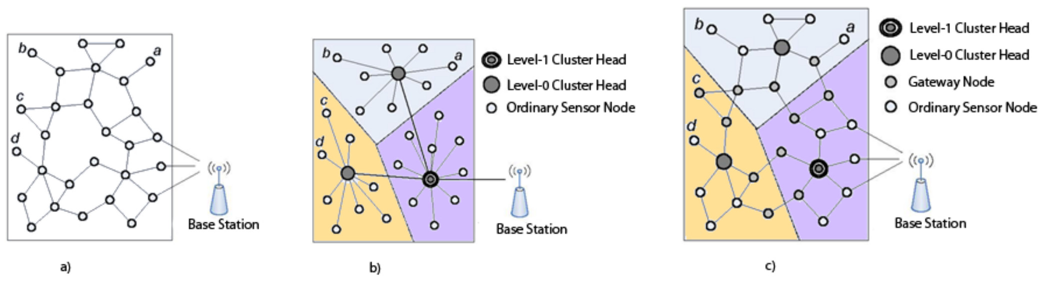


Figure 2.2: a) Mesh Configuration; b) Hierarchical Tree Configuration; C) Clustered Hierarchical Configuration

The choice between different wireless sensor network topologies is influenced mainly by four parameters : reliability, energy efficiency, scalability and latency

2.2.1 Reliability

Reliability is generally defined as the probability that the system will perform its intended function under stated conditions for a specified period of time. The WSN reliability can be studied for three different scopes of data delivery, known as the infrastructure communication:

- users send their interest to a single sensor node
- users send their interest to a subset of nodes in a sub-area and the message needs to be delivered to all sensors in the particular group

- users send their interest to the entire sensor network and the message needs to be delivered to all sensors in the network.

In Star WSN the reliability is simply the product of the reliabilities of communication between each node in the subset S with the sink node.
In Mesh WSN the reliability is the k-terminal network reliability among the nodes in the subset S and the base station.
In Clustered Hierarchical WSN the reliability is the probability that there exists an operational path from the sink to top hierarchical level CH, then to the next hierarchical level CH and so on to the destination group's CH and finally to all sensor nodes in that group. Note that the sensor nodes in the group may belong to a single cluster or a subset of contiguous or non-contiguous clusters.

For all the scenarios, mesh topology offers the highest reliability due to multiple paths through the network. A mesh network is highly fault tolerant as it offers multiple redundant paths throughout the network. If a routing node fails or the link between two nodes becomes unavailable, the network automatically reconfigures itself around the failed component. In a mesh WSN, the degree of redundancy, and in general the reliability of the network, is essentially a function of node density.
Tree topology has the lowest reliability due to the use of only a single direct link between nodes at successive levels in the hierarchy. Clustered hierarchical topology is a compromise between the two extremes. It is better than tree as it still maintains multi-hop paths, while it has lower reliability than mesh because each communication between nodes at different clusters must route through affiliated cluster heads.

2.2.2 Energy Efficency

While traditional networks aim on achieving high quality-of-service provisions, WSN focus primarily on energy awareness in every aspect of hardware and software design and operation to prolong the useful lifetime of each sensor node and in the entire WSN. The energy required for communication scales with distance (d) as d^2 to d^4. Since the radio signal attenuation scales with distance in a greater-than-linear fashion, the multi-hop communication in mesh networks consumes less power than the traditional single-hop long distance radio communication in star networks.

$P(h,d) = (h-1)P_{RxElec} + h\left[P_{TxRad1}\left(\frac{d}{h}\right)^r + P_{TxElec}\right]$

where d is the distance between node and base station, h is the number of hops, P_{rxElec} is the power required to receive and P_{txElec} is the one to transmit electronics, P_{txRad1} is the radiated power required for a successful one-meter transmission and r is the path loss exponent.

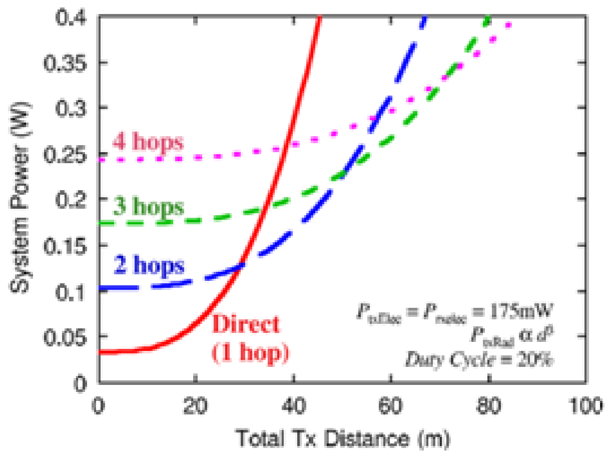


Figure 2.3: As the distance between source node and base station increases, the number of intermediate hops increases

It is possible to calculatean optimal characteristic distance and an optimal number of hops that balances the reduced transmission energy and increased receive energy. The optimal number of hops (K_{opt}) can be calculated as $K_{opt} = \left| \frac{d}{d_{char}} \right|$ where D is the distance between the node and the base station and $d_{char} = \sqrt[n]{\frac{\alpha_1}{\alpha_2(n-1)}}$, with α_1 the energy/bit consumed by the transmitter electronics (including energy costs of imperfect duty cycling dueto finite startup time), α_2 accounts for energy dissipated in the transmit op-amp (including op-amp inefficiencies) and n is the loss index.

The existence of a characteristic distance has two practical implications for micro-sensor networks. First, it is often impractical to ensure that all nodes are spaced exactly d_{char} apart: the deployed nodes may be placed in a line of nodes and a base station separated at a distance of either d or $2d$, with $d < d_{char} < 2d$. The second practical implication of a fairly large d_{char} is that there are a large class of applications for which the entire network diameter will be less than d_{char}. For these applications, the best communication policy is not to employ multi-hop at all: direct transmission from each node to the base station is the most energy-efficient communication scheme. For today's radio hardware, the typical d_{char} is around 20 meters.
For the scenario where the network size is less than the characteristic distance, direct communication in star networks is the most energy efficient because star topology facilitates energy savings for the sensor nodes by al-

lowing them to enter in sleep mode independently and to turn on briefly for sensing and communicating the sensed data back to the base station. In a mesh WSN, nodes closer to the base station handle more traffic and deplete their energy faster than the nodes away from the base station, thereby disconnecting the base station from the whole WSN, which might still have adequate resources and infrastructure.
Clustered hierarchical architecture is inherently amenable to in-network processing. It decreases communication traffic and communication frequency via data aggregation progressively at each CH in the hierarchy by processing and filtering the possibly redundant data received from its member nodes. To prevent unfairly taxing the battery power of the CH, LEACH algorithm uses randomized rotation of high-energy CH responsibility among all the nodes to distribute the energy load evenly among the sensors in the network. Simulations show that LEACH can achieve as much as a factor of 8 reduction in energy dissipation compared with direct communication when using the optimal number of cluster-heads (5%).

2.2.3 Scalability and self-organization

In star WSN, addition of nodes increases load on the base station, which results in increased power consumption and complexity. Also, as node density increases, the increase in collision greatly degrades performance. It is difficult to scale star WSN to more than a few nodes.
Mesh WSN are self-configuring networks that dynamically optimize routes through the network, based on the best link quality between neighboring nodes. The propagation of sensor data through the mesh allows WSN to be extended, in theory, to an unlimited range. However, the transmission and duty-cycle scheduling of sensor nodes should be planned in a very careful way. Therefore mesh networks work well for medium sized networks, but have scalability limitations that degrade performance for larger or densely deployed WSN.
Clustered hierarchical networks improve the scalability of the mesh networks by assigning CH to manage the local neighborhood of sensor nodes. For example, LEACH uses localized coordination to enable scalability and robustness for dynamic networks. Also, the adaptive self-organizing capabilities of clustered hierarchical WSN allow the periodic reformation of hierarchical clusters of sensor nodes in the event of environmental or topology changes as sensor nodes fail or new sensor nodes are added to improve connectivity and coverage.

2.2.4 Data latency

Star networks have the least data latency because there is nodelay due to buffering at routers along the path, but they are not scalable and moreover there may be more loss due to collision as the network density increases.
Mesh network has higher data latency than star but lower data loss because keeping the transmission power lower reduces the packet collision rate. Depending on the number ofnodes and the distances between them, mesh network may endure increased latency as message moves along multi-hop route to the base station. Also, mesh network can cause the nodes closer to base station to overload with the increase in node density which might cause latency in communication, and at the worst case create a black hole of overloaded (or dead) nodes around the base station.
In hierarchical tree topology, as the data moves from the lower level to a higher level, it moves a greater distance, thus reducing the travel time and data latency. However, as the distance between cluster levels increases, the energy dissipation, which is proportional to square of the distance, increases. Clustering is a design approach to minimizing energy consumption and minimizing communication latency In clustered hierarchical topology, only CH (along the hierarchy) perform aggregation; while in mesh topology, (every) intermediate nodes perform aggregation. As a result, clustered hierarchical architecture has lower latency than mesh topology.

2.3 Power Consumption and Duty Cycle

There are two important parameter in the ultra-low power sensor network: the sleep current, which affects the power consumption of the device while it is sleeping, and the time needed by the sensor to wake up after the sleeping time. Ultra-low power sensors spend most of their life asleep, waking up as a result of an external event or an internal timer. When they are sleeping, they need to consume negligible current, ideally no more than the leakage current of the source powering them. Most of the devices manage deep-sleep currents of a few microamperes, whilesome of the best low-power radio chips can survive on a few hundred nanoamperes.
The second parameter is how quickly the device can wake up, assemble and transmit the data packet, wait for an acknowledgement (assuming that the protocol requires one) and then return to the sleep mode (Figure 2.4). The length of this time is principally determined by the standard, which specifies the number and size of packets that need to be sent over the wireless link. Standards that aim for this ultra-low power market minimize the need for

unnecessary transactions, and can give total 'ON' times less than 5 milliseconds. Moreover, for many radios the current consumption in receive mode is greater than it is in transmit mode. So if a transmission needs to be acknowledged,it is important that it is done as quickly as possible, so that the receiver does not need to be awake and consume energy during the wait for acknowledgement process.

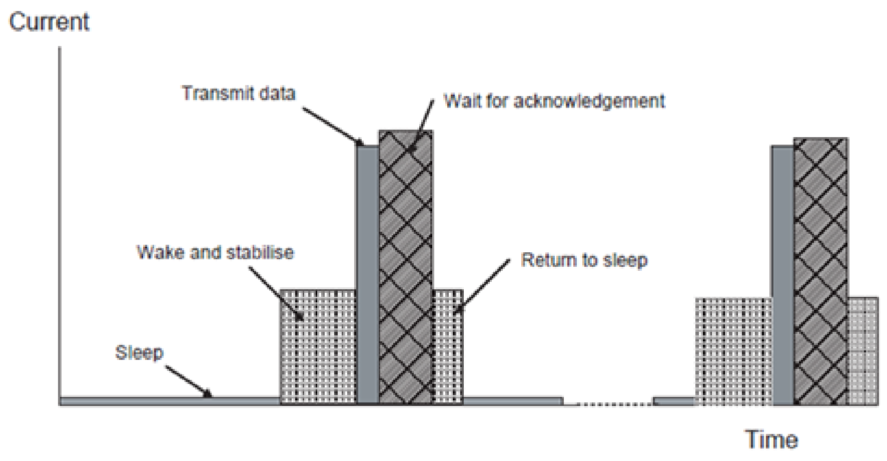


Figure 2.4: Sensor Life Cycle - Power Consumption example

2.4 IEEE 802.15.4 Standard

IEEE 802.15.4 standard defines the protocol and compatible interconnection for data communication devices using low-data-rate, low-power, and low-complexity short-range radiofrequency (RF) transmissions in a wireless personal area network (WPAN). The definition of the network layers is based on the OSI model, although only the physical layer (PHY) and medium access control (MAC) layers are defined (as shown in Figure 2.5).

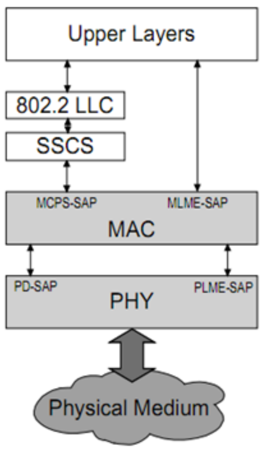


Figure 2.5: IEEE 802.15.4 Protocol Stack

The 2006 revision upgrade included two optional physical layers(PHYs) yielding higher data rates in the lower frequency bands and therefore the following four PHYs are specified:

PHY (MHz)	**Frequency Band**	**Channel Number**	**Modulation**	**Bit-Rate** (Kb/s)	**Symbols**
868/915	868-868.6	0	BPSK	20	Binary
	902-928	1-10	BPSK	40	Binary
868/915	868-868.6	0	ASK	250	20-bit PSSS
(optional)	902-928	1-10	ASK	250	5-bit PSSS
868/915	868-868.6	0	O-QPSK	100	16-ary Orth.
(optional)	902-928	1-10	ASK	250	16-ary Orth.
2450	2400-2483.5	11-26	O-QPSK	250	16-ary Orth.

2.4.1 IEEE 802.15.4 Protocol Standard

The standard uses carrier sense multiple access with collision avoidance (CSMA-CA) medium access mechanism, while the media access is contention based. However, using the optional superframe structure, time slots can be allocated by the PAN coordinator to devices with time critical data. Connectivity to higher performance networks is provided through a PAN coordinator. Two different device types can participate in an IEEE 802.15.4 network: a full-function device (FFD) and a reduced-function device (RFD). The FFD can operate in three modes serving as a personal area network (PAN) coordinator, a coordinator or a device and is able to talk to RFDs or other FFDs, while an RFD can talk only to an FFD. RFD are intended for extremely simple application and therefore they can be implemented using minimal resources and memory capacity.
Two types of topologies are supported by this standard: star topology and peer-to-peer topology. All devices operating on a network of either topology shall have unique 64-bit addresses, which may be used for direct communication within the PAN, or a short address may be allocated by the PAN coordinator when the device associates and used instead. Peer-to-peertopology allows more complex network formations to be implemented, such as mesh networking topology. A peer-to-peer network can be ad hoc, self-organizing, self-healing and may also allow multiple hops to route messages from any device to any other device on the network. Such functions can be added at the higher layer, but are not part of the 802.15.4 standard.
The MAC sublayer provides the MAC data service and the MAC management service interfacing to the MAC sublayer management entity service access point (known as MLME-SAP). The transmission can use beaconless mode or beacon mode. In the first one a CSMA-CA channel access mode is performed similarly to the IEEE 802.11 protocol standard. Each time a device wishes to transmit data frames or MAC commands, it waits for a random period. If the channel is found to be idle, following the random backoff, the device transmits its data. If the channel is found to be busy following the random backoff, the device waits for another random period before trying to access the channel again. Acknowledgment frames are sent without using a CSMA-CA.
In the case of beacon mode, the standard allows the optional use of a superframe structure which is bounded by network beacons sent by the coordinator and is divided into 16 equally sized slots. The beacon frame is transmitted in the first slot of each superframe and is used to synchronize the attached devices, to identify the PAN, and to describe the structure of the superframes. Any device wishing to communicate during the contention

access period (CAP) between two beacons competes with other devices using a slotted CSMA-CA mechanism. Beacon mode uses an hybrid Time Division Multiple Access (TDMA) which allows to reserve time-slots for critical data transmission.

2.4.1.1 Frame Format

The IEEE 802.15.4 standard uses four types of frames:

- Data frame, for data transmission.
- Acknowledgement frame, transmitted from the receiver nodeafter 12 symbol period (192μs), after the reception of a frame, if explicitly requested by an ack bit flag in the MAC header data.
- MAC Layer command frames, used in the beacon-enabled mode to use some services at the MAC layer like coordinator association-disassociation and the transmission synchronization management.
- Beacon frame, used by the coordinator in the beacon-enabled mode.

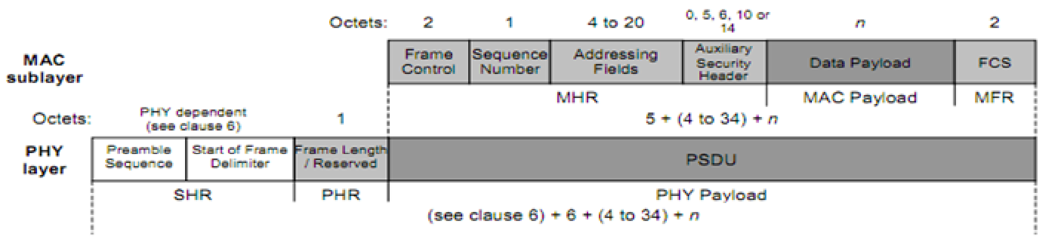


Figure 2.6: IEEE 802.15.4 Data Frame

The structure of a frame, which originates from the upper layers, is shown in Figure 2.6. The data payload is passed to the MAC sublayer and is referred to as the MAC service data unit (MSDU). The MAC payload is prefixed with an MHR, which contains the FrameControl field, data sequence number (DSN), addressing fields, and optionally the auxiliary security header, and appended with an MFR, which is composed of a 16-bit FCS.
The Frame Control field is 2 octets in length and contains information defining the frame type (beacon, data, acknowledgement or MAC command) , addressing field, and other control flags. The Sequence Number field is 1 octet in length and specifies the sequence identifier for the frame. The addressing field includes the Destination PAN Identifier, the Destination Address,

the Source PAN Identifier and the Source address. The Auxiliary Security Header field has a variable length and specifies information required for security processing, including how the frame is actually protected (security level) and which keying material from the MAC security PIB is used. The FCS field is 2 octets in length and contains a 16-bit ITU-T CRC (cyclic redundancy check) in order to detect bit errors in every frame. The FCS is calculated over the MHRand MAC payload parts of the frame using the following standard generator of polynomial of degree 16:

$G_{16} = x^{16} + x^{12} + x^{5} + 1$

2.4.1.2 Security

The actual frame protection provided can be adapted on a frame-by-frame basis and allows for varying levels of data authenticity (to minimize security overhead in transmitted frames where required) and for optional data confidentiality. When nontrivial protection is required, replay protection is always provided.
Cryptographic frame protection may use a key shared between two peer devices (link key) or a key shared among a group of devices (group key), thus allowing some flexibility and application-specific tradeoffs between key storage and key maintenance costs versus the cryptographic protection provided. If a group key is used for peer-to-peer communication, protection is provided only against outsider devices and not against potential malicious devices in the key-sharing group. There are three fields in the IEEE 802.15.4 MAC frame which are related to security issues:

- Frame Control (located in the MAC Header)
- Auxiliary Security Control (in the MAC Header)
- Data Payload (in the MAC Payload field)

As shown in Figure 2.7 the Auxiliary Security Frame is only enabled if the Security Enabled subfield of the Frame Control Frame is turned on. This special header has 3 fields:

- Security Control (1B) specifies which kind of protection is used.
- Frame Counter (4B) is a counter given by the source of the current frame in order to protect the message from replaying protection. For this reason each message has a unique sequence ID represented by this field.

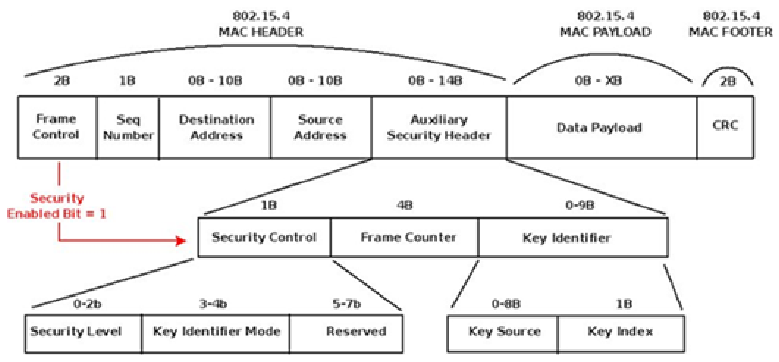


Figure 2.7: IEEE 802.15.4 Security

- Key Identifier (0-9B) specifies the information needed to know what key we are using with the node we are communicating with.

Each 802.15.4 transceiver has to manage a list to control its "trusted brothers" along with the security policy . For this reason each node has to control its own Access Control List (ACL) which stores the following fields:

- Address: of the node we want to communicate with
- Security Suite: the security police which is being used (AEC-CTR, AES-CCM-64, AES-CCM-128,...)
- Key: the 128b key used in the AES algorithm
- Last Initial Vector (IV) and Replay Counter: both are the same field. The Last IV is used by the source and the Replay Counter by the destination as a message ID in order to avoid reply attacks.

When a node wants to send a message to a specific node or receives a packet, it looks at the ACL to see if it is a trusted brother or not. In the case it is, the node uses de data inside the specific row to apply the security measures. In the case the node is not in the list or its message is rejected, an authentication process starts.

2.4.2 ZigBee

The IEEE 802.15.4 protocol does not standardize the higher communication protocol layers, including the network and application layers. To assure interoperability between devices operating with the IEEE 802.15.4 standard, the behavior of these layers must be specified. The creation of such a specification has been taken up by the ZigBee Alliance. The IEEE 802.15.4 MAC and PHY layers define the RF and communication components of neighboring devices, while ZigBee stack layers, on the other hand, include a network layer, an application layer and a security service provider(SSP), as shown in Figure 2.8.

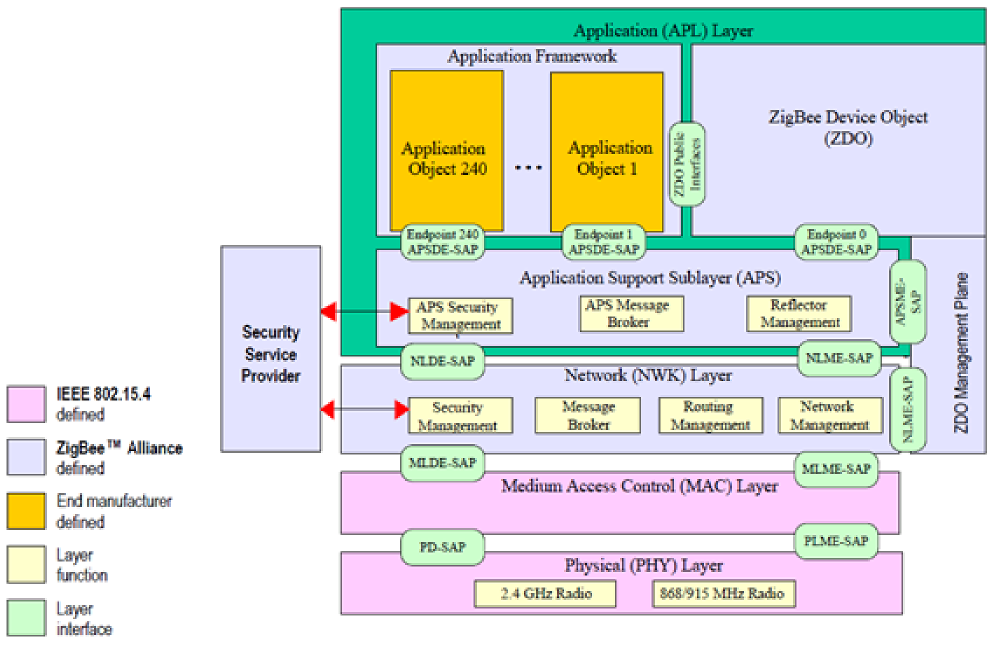


Figure 2.8: ZigBee protocol stack

ZigBee uses Direct-Sequence Spread Spectrum (DSSS) in the 2.4GHz band, with Offset-Quadrature Phase-Shift Keying (O-QPSK) modulation. Each channel has a width of 2MHz, with 5MHz channel spacing. The 868 and 900MHz bands also use direct-sequence spread spectrum but with binary-phase-shift keying modulation (BPSK). Raw data throughput rates of 250Kbps can be achieved at 2.4GHz (16 channels), 40Kbps at 915MHz (10 channels), and 20Kbps at 868MHz (1 channel). Transmission range is between 10 and 75m, although it is heavily dependent on the particular environment and on the device output power, which is generally 0 dBm (1 mW).

In the ZigBee specification, three different types of ZigBee devices are defined:

- ZigBee coordinator (ZC): The most capable device, the coordinator forms the root of the network tree and might bridge to other networks. There is exactly one ZigBee coordinator in each network since it is the device that started the network originally. It is able to store information about the network, including acting as the Trust Center & repository for security keys.

- ZigBee Router (ZR): These devices extend network area coverage, dynamically route around obstacles, and provide backup routes in case of network congestion or device failure. They can connect to the coordinator and other routers, and also support child devices.

- ZigBee End Device (ZED): Contains just enough functionality to talk to the parent node (either the coordinator or a router); it cannot relay data from other devices. This relationship allows the node to be asleep a significant amount of the time thereby giving long battery life. A ZED requires the least amount of memory, and therefore can be less expensive to manufacture than a ZR or ZC.

The protocol builds an algorithmic research (Ad-hoc On-demand Distance Vector) to automatically construct an ad-hoc network of nodes which can form a star network, a mesh network or a cluster tree, as shown in Figure 2.9.

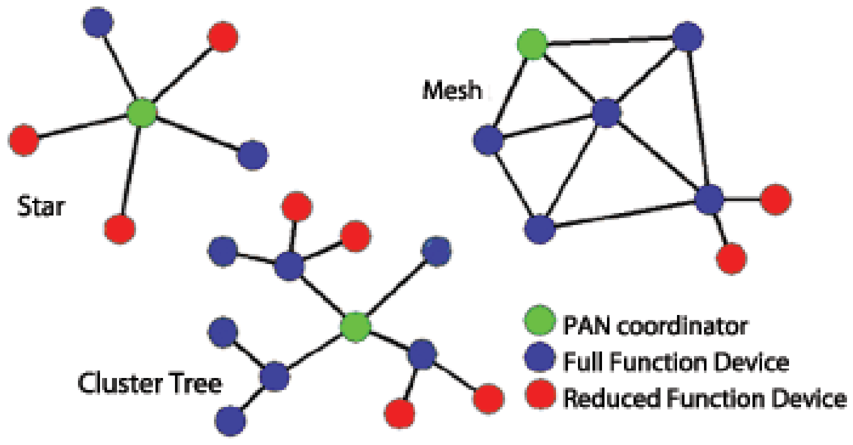


Figure 2.9: ZigBee Network Topologies

ZigBee operates in two main modes: non-beacon mode and beacon mode. Beacon mode is a fully coordinated mode where all the devices know when to coordinate with one another. In this mode, the network coordinator will periodically "wake-up" and send out a beacon to the devices within its network. This beacon subsequently wakes up each device, who must determine if it has any message to receive. If not, the device returns to sleep, as will the network coordinator, once its job is complete. Non-beacon mode, on the other hand, is less coordinated, as any device can communicate with the coordinator at will. However, this operation can cause different devices within the network to interfere with one another, and the coordinator must always be awake to listen for signals, thus requiring more power.

2.4.2.1 Network Layer (NWK)

The ZigBee stack architecture includes a number of layered components including the IEEE 802.15.4-2003 Medium Access Control (MAC) layer, Physical (PHY) layer, and the ZigBee Network (NWK) layer. Each component provides an application with its own set of services and capabilities. , its primary purpose is to describe the component labeled Application (APL) Layer. In order to interface with the application layer, the network layer conceptually includes two service entities that provide the necessary functionality. These service entities are the data service, provides the data transmission service via its associated SAP, and the management service, which provides the management service via its associated SAP.

The Network Layer Data Entity provides a data service to allow an application to transport application protocol data units between two or more devices located on the same network. The Network Layer Data Entity will provide the following services:

- Generation of the Network level PDU (NPDU): The NLDE shall be capable of generating an NPDU from an application support sub-layer PDU through the addition of an appropriate protocol header.

- Topology-specific routing: The NLDE shall be able to transmit an NPDU to an appropriate device that is either the final destination of the communication or the next step toward the final destination in the communication chain.

- Security: The ability to ensure both the authenticity and confidentiality of a transmission.

The Network Layer Management Entity provides a management service to allow an application to interact with the stack and provides the following services:

- Configuring a new device: this is the ability to sufficiently configure the stack for operation as required. Configuration options include beginning an operation as a ZigBee coordinator or joining an existing network.
- Starting a network: this is the ability to establish a new network.
- Joining, rejoining and leaving a network: this is the ability to join, rejoin or leave a network as well as the ability of a ZigBee coordinator or ZigBee router to request that a device leave the network.
- Addressing: this is the ability of ZigBee coordinators and routers to assign addresses to devices joining the network.
- Neighbor discovery: this is the ability to discover, record, and report information pertaining to the one-hop neighbors of a device.
- Route discovery: this is the ability to discover and record paths through the network, whereby messages may be efficiently routed.
- Reception control: this is the ability for a device to control when the receiver is activated and for how long, enabling MAC sub-layer synchronization or direct reception.
- Routing: this is the ability to use different routing mechanisms such as unicast, broadcast, multicast or many to one to efficiently exchange data in the network.

2.4.2.2 Application Layer (APL)

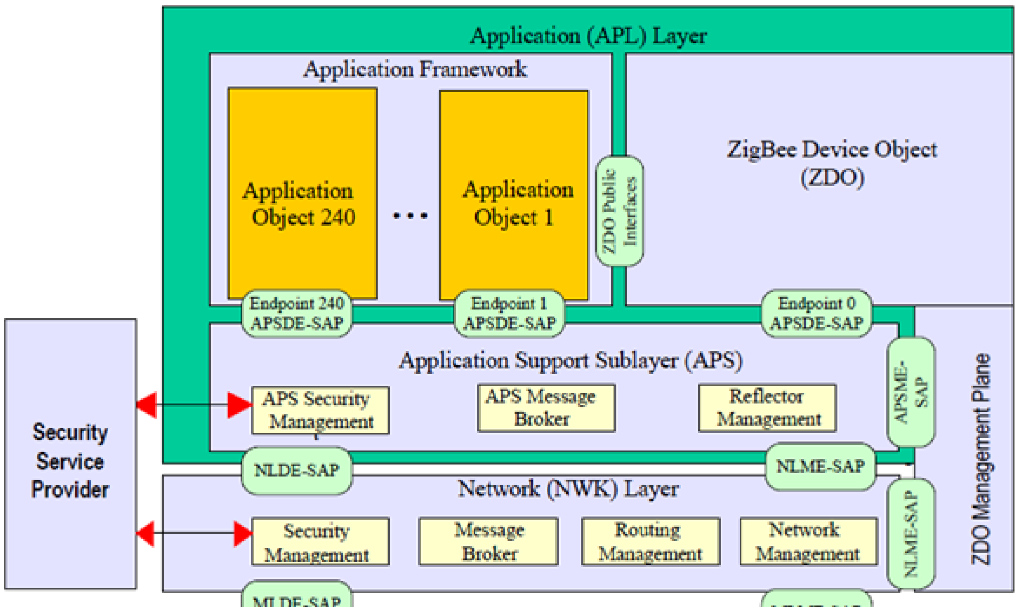


Figure 2.10: Zigbee Application Layer

The ZigBee application layer consists of the APS sublayer, the ZigBeeDevice Object (containing the ZDO management plane), and the manufacturer defined application objects, as shown in Figure 2.10. The application support sub-layer (APS) provides an interface between the network layer (NWK) and the application layer (APL) through a general set of services that are used by both the ZDO and the manufacturer-defined application objects. The services are provided by two entities:

- The APS data entity (APSDE) through the APSDE service access point (APSDE-SAP).
- The APS management entity (APSME) through the APSME service access point (APSME-SAP).

The APSDE provides the data transmission service between two or more application entities located in the same network.
The application framework in ZigBee is the environment in which application objects are hosted on ZigBee devices. Up to 240 distinct application objects can be defined, each identified by an endpoint address from 1 to 240. Two additional endpoints are defined for APSDESAP usage: endpoint 0 is

reserved for the data interface to the ZDO, and endpoint 255 is reserved for the data interface function to broadcast data to all application objects. Endpoints 241-254 are reserved for future use.

2.4.2.3 Energy consumption

The energy consumption of a ZigBee wireless sensor network is strictly connected to the duty cycle. In beacon enabled mode, the duty cycle is given by the expression $\frac{2^{SO}}{2^{BO}}$, where SO and BO are two MAC layer parameters that define the time interval between beacons (Beacon Order BO) and the duration of the active part (Superframe Order SO), as shown in Figure 2.11.

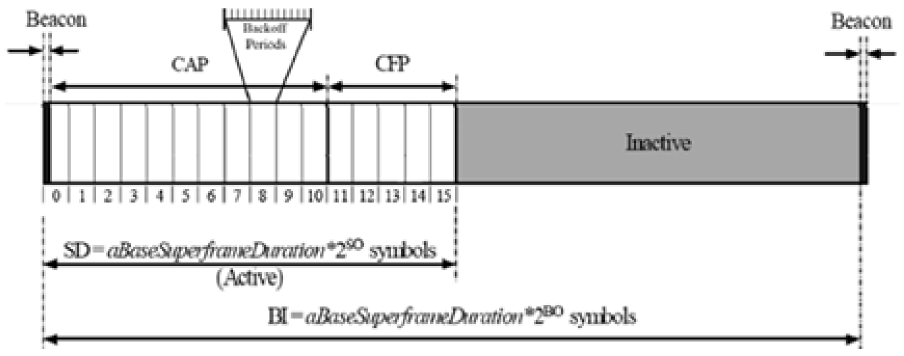


Figure 2.11: IEEE 802.15.4 - Zigbee Superframe structure

Low duty cycle conserves energy by putting devices to sleep. However, a low duty cycle reduces the bandwidth and increases latency. With the same duty cycle, different combination of SO and BO are possible, which means the superframe structure has different active period and inactive period.

2.4.2.4 Security in ZigBee

ZigBee implements two extra security layers on top of the 802.15.4 one: the Network and Application security layers. All the security policies rely on the AES 128b encryption algorithm so the hardware architecture previously deployed for the link level (MAC layer) is still valid. There are three kind of Keys: master, link and network keys.

- Master Keys: They are pre-installed in each node. Their function is to keep confidencial the Link Keys exchange between two nodes in the Key Establishment Procedure (SKKE).

- Link Keys: They are unique between each pair of nodes. These keys are managed by the Application level. They are used to encrypt all the information between each two devices, for this reason more memory resources are needed in each device

- Network key: It is a unique 128b key shared among all the devices in the network. It is generated by the Trust Center and regenerated at different intervals. Each node has to get the Network Key in order to join the network. Once the trust center decides to change the Network Key, the new one is spread through the network using the old Network Key (see image above about ”ZigBee Residential Mode”). Once this new key is updated in a device, its Frame Counter (see in the previous sections) is initialized to zero. This Trust Center is normally the Coordinador, however, it can be a dedicated device. It has to authenticate and validate each device which attempts to join the network.

Each pair of devices can have set both Network and Link Keys. In this case the Link key is always used (more security although more memory is needed). There are two kinds of security policies which the Trust Center can follow:

- Commercial mode: the Trust Center share Master and Link Keys with any of the devices in the network. This mode requires high memory resources. This mode offers a complete centralized model for the Key Security control.

- Residential mode: the Trust Center shares just the Network Key (it is the ideal mode when embedded devices have to cope with this task due to the low resources they have). This is the mode normally chosen for the Wireless Sensor Network model

2.4.3 6LoWPAN

The Internet Engineering Task Force (IETF), which creates and maintains all core Internet standards and architecture work, defined the 6LoWPAN standard, which enables the efficient use of IPv6 over low-power and low rate wireless networks on simple embedded devices. This was made through an adaptation layer and the optimization of the related protocol, which was possible with a simplification of the IPv6 functionality, defining very compact header formats and taking the nature of wireless networks into account.
The 6LoWPAN concept originated from the idea that the Internet Protocol could and should be applied even to the smallest devices, and that low-power devices with limited processing capabilities should be able to participate in the Internet of Things. The vision behind the Internet of Things is that smart objects are becoming IP enabled, which means they will become integral part of the Internet. The scale of these IP enabled objects is already estimated to be larger than trillions of devices, while there are several applications which could benefit from the use of the internet protocols because IP based devices can be easily connected to other IP networks without the use of any gateway or proxy, while the IP technology is very well known, have existed for decades, is specified to an open and free way with standard processes and documents available to anyone.

2.4.3.1 6LoWPAN Network Architecture

The Internet of Things is created by connecting clouds of wireless embedded devices, where each cloud is an independent network on the Internet, which means that IP packets are sent within the network, but doesn't act as a transit to other networks. A LoWPAN can be defined as the collection of 6LoWPAN nodes which share a common IPv6 address prefix, the first 64 bits of an IPv6 address.
Three different kind of LoWPAN can be defined, as shown in Figure 2.12:

- Simple LoWPAN: connected through a LoWPAN Edge Router to another IP network
- Extended LoWPAN: includes the LoWPAN of multiple edge routers along with a backbone link interconnecting them
- Ad Hoc LoWPAN: not connected to the Internet, it operates without an infrastructure

The main difference between simple and extended LoWPAN is the presence of multiple edge routers sharing the same IPv6 prefix and common

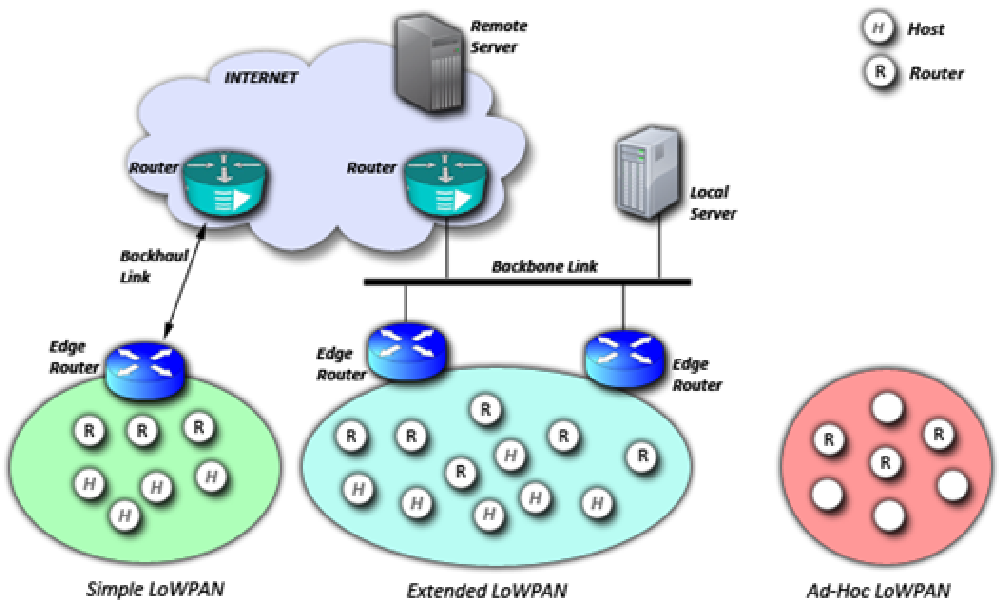


Figure 2.12: 6LoWPAN Network Types

backbone link(i.e. Ethernet), which leads to a remarkable offload of the Neighbor Discovery messaging in the backbone link and simplifies LoWPAN node operation because IPv6 addresses are stable through the network.
The edge router plays an important role in the 6LoWPAN network not only because it routes the traffic in and out of the LoWPAN, but also because it handles the 6LoWPAN compression and Neighbor Discovery, which defines how hosts and routers interact with each other on the same link. For example the network interfaces of the nodes in a LoWPAN share the same IPv6 prefix and, in order to facilitate efficient network operation, nodes register with an edge router during the Neighbor Discovery. Moreover, if the LoWPAN is connected to an IPv4 network, the edge router should also handle the IPv4 interconnectivity.
The interaction between LoWPAN nodes and IP nodes in other networks happens in an end-to-end way: every single LoWPAN node is identified by a unique IPv6 address and is capable of sending and receiving IPv6 packets.

2.4.3.2 6LoWPAN Protocol Stack

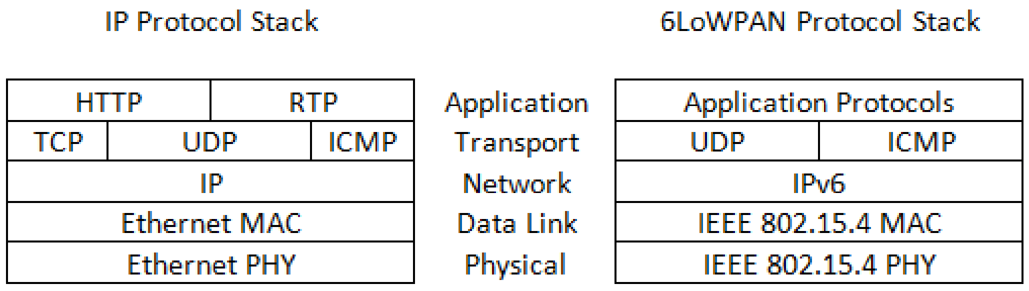


Figure 2.13: 6LoWPAN Protocol Stack

A simple IPv6 protocol stack with 6LoWPAN is almost the same as a normal IP stack, as shown in Figure 2.13. There are, anyway, some relevant differences:

- 6LoWPAN supports only IPv6, for which a small adaptation layer was defined to optimize IPv6 over IEEE 802.15.4
- The UDP (User Datagram Protocol) is the most common transport protocol used by 6LoWPAN, while the TCP protocol is not commonly used for performance, complexity and efficiency reasons.

The adaptation between the full IPv6 and the 6LoWPAN format is performed by the edge routers.

2.4.3.3 6LoWPAN packet format

An adaptation layer has to map IP datagrams to the services provided by the subnetwork, which is usually considered to be at Layer 2. In particular, the Layer 2 has to solve the following problems:

- In case a packet is overheard by multiple receivers, not all of which may need to act on it, the Layer 2 address provides an efficient way to make this decision
- The IP packet needs to be encapsulated in a subnetwork in a way that the receiver Layer 2 can extract the IP packet again. This leads to three sub-problems:
  - IP packets may not fit into the data units that Layer 2 can transport: the maximum packet size that can be sent using an IP

network Interface (MTU) for IPv6 is at least 1280 bytes. IEEE 802.15.4 can only transport L2 packets of up to 127 bytes. In order to transport larger IPv6 packets segmentation and reassembly (also called fragmentation) is implemented.

- There may be need to distinguish different kinds of encapsulation
- In a LoWPAN the typical IP/UDP header size of 48 bytes already consumes a significant part of the payload space. For this reason header compression is a key feature and 6LoWPAN comes with its own header compression.

6LoWPAN attempts to have modest link layer requirement: the basic service required by the link layer is for one node to be able to send unicast packets to another node within radio reach (one hop neighbor). Regarding the four types of 802.15.4 MAC Layer frame (data, acknowledgement, command and beacon frame), the 6LoWPAN specification only concerns data frames, which carry both source and destination address. IEEE 802.15.4 nodes are permanently identified by EUI-64 identifiers (8 bytes), but the standard defines also a 16 bit short address format.
The basic IEEE 802.15.4 data packet format does not contain any fields that further identify the payload carried by a packet, which means that there is no multiplexing information that allows the receiver to distinguish 6LoWPAN packets from any other data packets that might be sent or to distinguish the different kind of 6LoWPAN packets. One of the duties of the 6LoWPAN encapsulation format is to provide a packet type identifier, which is not provided by the IEEE 802.15.4 itself. The first byte of the payload is therefore used as a dispatch byte, providing both a type identifier and, when possible, some additional information within the subtype.

00	Not a LoWPAN packet
01	Normal Dispatch
10	Mesh Header
11	Fragmentation Header

Some of the formats defined by 6LoWPAN are designed to carry further 6LoWPAN PDUs as their payload. When multiple headers need to be present, the question is which header should be transported as the payload of which other header (i.e. which order the header should be nested). The various headers should be used in the following order:

- Addressing: the mesh header, carrying the Layer 2 original source and final destination address and hop count, followed by a 6LoWPAN PDU

- Hop-by-hop processing: headers that are Layer 2 hop-by-hop options (such as the broadcast header) followed by a 6LoWPAN PDU
- Destination processing: the fragmentation header, carrying fragments that, after possibly having been carried through multiple Layer 2 hops, need to be reassembled to a 6LoWPAN PDU on the destination node
- Payload: headers carrying layer 3 packets such as IPv6 (figure below), LOWPAN_HC1 or LOWPAN_IPHC

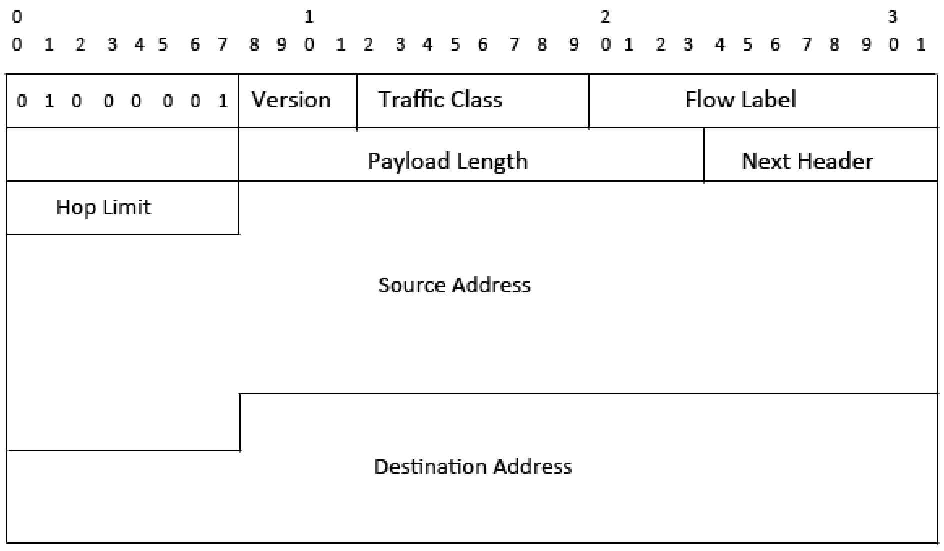


Figure 2.14: uncompressed IPv6 packet with 6LoWPAN header

2.4.3.4 Addressing

An IP adaptation layer involves usually at least two kind of addresses: link layer addresses (Layer 2) and IP addresses (layer 3). In addition, 6LoWPAN supports two address formats at the link layer (the 64-bit EUI-64 address and the dynamically assigned 16-bit short address). To ensure global uniqueness of the EUI-64s, the manufacturer first has to buy a 24-bit OUI(organization unique identifier) from the IEEE for a one-time fee, and then builds the EUI-64 from the OUI and 40 bits of extension identifier chosen by the manufacture with a procedure similar to the one for creating the Ethernet's 48 bit MAC address, as shown in Figure 2.15.

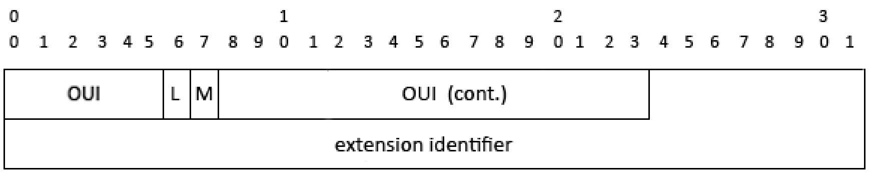


Figure 2.15: Composition of a EUI-64

It is important to note that the least significant bit (M) is used to distinguish multicast address from unicast ones, while the second least significant bit (L) is used to distinguish locally assigned addresses from universal addresses assigned globally with the OUI scheme.

One easy way of forming an IPv6 address is to combine a 64-bit prefix with a 64-bit EUI-64 as an interface ID to yield a 128-bit value. The designers of IPv6 only applied one twist: the L bit was inverted, turning into a U bit for universal address. In the hexadecimal the IPv6 address representation the universal addresses have the first 16 bit subfield in the interface ID which are XOR-ed with 0200 (hexadecimal), as shown in Figure 2.16. This allow the easy to remember address like 2001:db8::1 for locally assigned addresses.

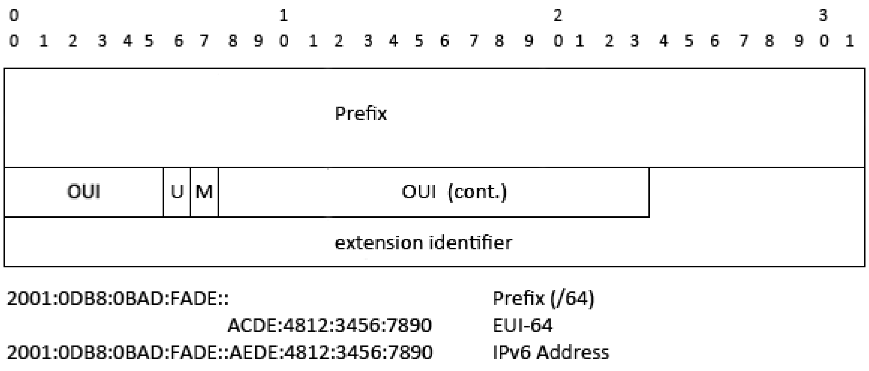


Figure 2.16: Composition of an IPv6 address from an EUI-64: U is the inverted L bit

In 6LoWPAN this derivation of IPv6 addresses from link layer is mandatory because the Neighbor Discovery protocol rely on this mapping. While in the IEEE 802.15.4 the 16-bit short address is assigned by the PAN coordinator during the association procedure, in 6LoWPAN this is done by the edge router according to the Neighbor Discovery procedure. To form an IPv6 address out of a 16-bit short address, the short address (assigned by the PAN coordinator) is combined with a PAN identifier, as shown in Figure 2.17. The rule for the universal/local bit requires bit6 of the PAN identifier to be zero, as this is not an IEEE assigned universal EUI-64.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

PAN Identifier	0	PAN Identifier	0	0	F	F
F	E	0	0	16-bit Short Address		

Figure 2.17: Interface identifier for 16-bit short address

Frequently the packets have to perform multiple radio hops through the network, which involves two related processes: forwarding and routing. In each node the routing protocol fills in a routing information base (RIB) which contains all the information needed to perform the routing protocol. Usually the RIB can be simplified to a forwarding information base (FIB) which is consulted when a packet arrives and needs to be forwarded. There are two different way to fill the FIB: proactively and reactively. In the first one the FIB should always contain an entry for each packet that can actually be forwarded, while in the second one the FIB fills the gaps only when a packet arrives.

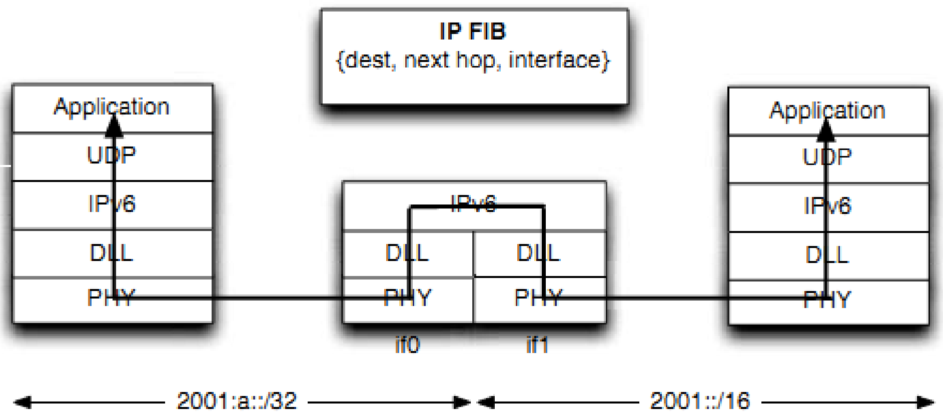


Figure 2.18: Forwarding Information Identifier

When packets arrives at a router on an interface if0, it looks up the destination address in its FIB, selects an interface to forward it out and sends the packet encapsulated with the new link layer address. In 6LoWPAN forwarding is motivated by the fact that the first node may not have the radio

range to reach the third node.
When route and forwarding happens in the layer 3 (also called route-over forwarding), no special support from the adaptation layer format is required. Before the layer 3 forwarding engine sees the packet, the adaptation layer has already decapsulated the packet. This means that fragmentation and reassembly are performed at each hop in route-forwarding, as the layer 3 addresses are part of the initial bytes of the IPv6 header (present only in the first fragment of a larger packet).

2.4.3.5 Header compression

One of the main characteristics of 6LoWPAN is the limited payload size of the packets provided by IEEE 802.15.4, half of which is then consumed by the size of an IPv6 header. 6LoWPAN is more efficient when all the IPv6 packets can be made to fit into a single IEEE 802.15.4 packet, avoiding therefore fragmentation and reassembly. Much of the information that is repeatedly sent in sequence of network headers is redundant and could be therefore compressed. Data compression techniques (such as gzip and DEFLATE algorithm) are optimized in order to suppress redundancies in a given data item, but these techniques don't work well on small data items such as single packets. Data compression algorithms are best applied to the application layer content of the packet, while a separate set of techniques has focused on compressing the headers in sequences of packets.
Header compression is mostly performed hop-by-hop as part of the adaptation layer. This allows compressing the full header stack including the IP header just before sending the packet on a link, and the decompressing and reconstructing the header stack in full before the packet is routed and sent on a different link. The advantage of this approach is that deploying header compression becomes a local decision between two neighbors.
There are plenty of header compression standards and techniques, one of those gained popularity its own formal notation: robust header compression (ROHC). This standard focuses on compressing flows of packets (like a sequence of packets from a single TCP connection) expending a considerable complexity for compressing the full stack of headers (IP/TCP or IP/UDP/RTP) by setting up flow state for each new connection or stream and stepping through a set of compression states during the exchange of the initial packets of the flow until a high level of compression is reached. The main disadvantage of this technique (and of all the flow-based header compression ones) is that the per-flow state is likely to get out of sync between sender and receiver when packets are lost. Because of this disadvantage and the high complexity of the algorithm behind, this protocol was judged inappropriate

for the resource-constrained LoWPAN systems by the IETF team.
Instead of the ROHC protocol, the IETF team preferred to employ in the original 6LoWPAN format exclusively the stateless compression (without state there is no synchronization problem). The 6LoWPAN format specification defines two header compression schemes designed to work together: HC1 to compress IPv6 headers and HC2 to compress UDP headers. Figure 2.19 shows the initial bytes of an HC1 or HC1/HC2 compressed payload.

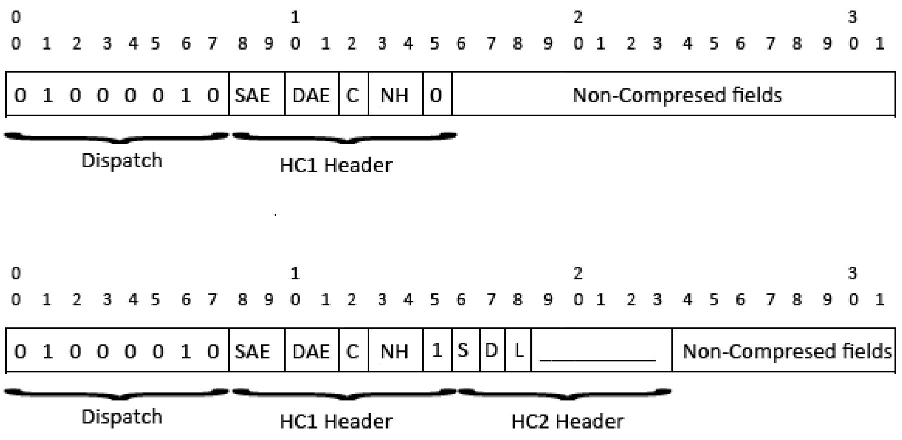


Figure 2.19: HC1 and HC2 Header

The goal of HC1/HC2 is to enable header compression in an entirely stateless way, with no requirement for previous agreement between nodes exchanging the compressed packets. HC1/HC2 exploits internal redundancies in the packet , which is caused by the way IP addresses are formed from Layer 2 addresses. When a 6LoWPAN host sends a packet, the Layer 2 source address of the packet will reflect the MAC address of the host. The IID half of the IP address is created from the MAC address as well, so it is redundant and can be elided. Similar considerations apply to the two destination addresses present in packets sent to hosts. HC1 only optimizes the case where the 64 bits of an IP address indicate a local link address, prefix FE80::/64. HC1 performs source address encoding (SAE) and destination address encoding (DAE), while the rest of the HC1 header is concerned with the compression of the non-address components of an IPv6 header.
The largest part of the IPv6 headers are the two IPv6 addresses in the header (together they can consume up to 40% of the usable space of a packet). Unfortunately it is impossible to compress the large globally routable IPv6 address without at least some state, which is the reason why the 6LoWPAN working group standardized a second header compression method context based. The most important correctness aspect about this method is to make sure

that the context stays synchronized between compressor and de-compressor. The context is divided into multiple slots that may be changed independently: a sending node should only start using a context slot when there is a sufficient reason to believe that the updated value for this slot has percolated to the receiving node. To further reduce the probability of damage, LoWPAN nodes should use context-based header compression only when a higher-layer protocol is used to protect the IPv6 addresses with some form of pseudo-header-based checksum. Like in the stateless header compression, the context based header compression is divided into a compression scheme for the IP header and an optional compression scheme for the next header.

2.4.3.6 Fragmentation and reassembly

While the smallest IPv6 packet would contain only the 40 byte IP header (and 0 bytes of payload), the largest IPv6 packet would contain a payload of up to 65535 (216-1) bytes. Most of the subnetwork define a maximum transmission unit in order to define the maximum packet size that can be transported efficiently. In a host with a single interface it is usually easy to find the MTU which allows an application to adjust the size of the packet that it sends. On the other hand, in a multiple interface multi-hop network the MTU is likely to change from hop to hop on the way to the packet's destination.
When a source of a datagram wants to use fragmentation it needs to insert a separate fragmentation header as an extension header. All the IPv6 subnetworks have to provide a minimum MTU of 1280 bytes, which was chosen to leave space for tunneling headers that may have to be wrapped around the packet before transmission an Ethernet and to leave some space for the IP header, while the recommended MTU size is 1500 bytes. Higher layers that want to send datagrams larger than the path MTU can fragment at the source using the IPv6 fragmentation header shown in Figure 2.20.

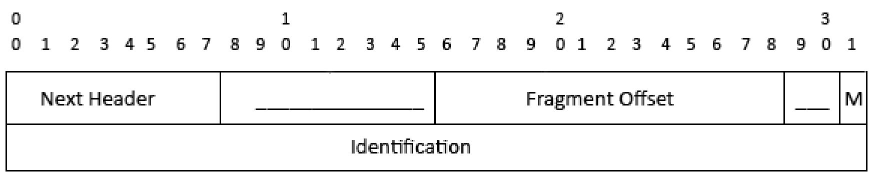


Figure 2.20: IPv6 fragmented header (M=MF)

6LoWPAN has different requirements on its fragmentation and reassembly mechanism. On one hand its link layer already provides error checking

and variable length frames, on the other hand 6LoWPAN links do not provide the in-sequence virtual circuit semantics like ATM (asynchronous transfer mode) so more information than a single bit is required to splice together the right frames into one IPv6 packet. Instead of providing more fragments flag like in IPv4, 6LoWPAN copies the size of the packet to be reassembled (IPv6 header + IPv6 payload) into every fragment. This enables the receiving end to allocate a buffer for the whole reassembly unit upon reception of the first fragment, independent of which of the fragments actually arrives first. In this way the receiver to allocate a buffer for the whole reassembly unit upon reception of the first fragment, independent of which of the fragments actually arrives first. A 16-bit datagram_tag, combined with the sender's link layer address, the destination's link layer address and the datagram_size is used to distinguish the different packets to be reassembled. An 8-bit datagram_offset indicates the position of the fragment in the reassembled IPv6 packet.
Fragmentation is undesirable for many reason, but the main one regards the decoupling between unit of loss (the fragment) and unit of retransmission (the entire packet) with the related unefficiencies. Consecutive layer-2 hops might have different values for max_frame; in certain cases the first hop in a LoWPAN might split up a packet into 80-bytes fragments, only to have the next hop split each of these into one 72-byte and one 8-byte fragment. The question is what are the packet sizes that an application can choose to make it likely not to cause layer-2 fragmentation. A UDP-based application will have payloads of 50-60 bytes or less to have a reasonable expectation that no fragmentation occurs in the LoWPAN.

2.4.3.7 Bootstrapping, Neighbor Discovery and Security

A 6LoWPAN device which is connected for the first time has to perform the following actions:

- Find the LoWPAN it is going to be part of
- Establish networking parameters such as the IP address prefix and its own IPv6
- Establish security associations with the relevant entities in the network
- Build paths out of the node to the relevant entities, maintain them and possibly start forwarding for the other nodes in the network
- Some of these establishments have to be repeated dynamically over small timescales (for example the selection of the router or the routing

paths for forwarding). Other parameters are less dynamic and their setup can be structured in two phases:

- Commissioning: some of the establishment of state require human intervention; security relationships are initialized in order to protect the network

- Bootstrapping : the node is ready to operate without further human intervention but there may still be state that needs to be acquired both when a device initializes (power-up fresh batteries) or when it enters a LoWPAN. A protocol that is related to bootstrapping is the Neighbor Discovery.

A node uses Neighbor Discovery to discover other nodes on the same link, to determine their link-layer addresses, to find routers and to maintain reachability information about the paths to neighbors that the node is actively communicating with. The protocol divides nodes into the traditional roles of host and router, where only the router forwards IP packets that are not addressed to itself. As many nodes in LoWPAN will be limited in their capabilities, 6LoWPAN-Neighbor Discovery introduces a third role, the edge router, specialized in performing some of the more complex functions of 6LoWPAN-Neighbor Discovery and therefore reducing the complexity of the tasks performed by the other routers and in particular by the hosts. The main concept added by the Neighbor Discovery protocol is the whiteboard maintained by the edge routers, centralizing some of the protocol state.
As mentioned before, 6LoWPAN simplifies the IPv6 addressing model by requiring that the node addresses are formed from an interface ID built from a MAC layer address and a prefix. Two such addresses are needed for each 6LoWPAN interface: a link local address (constructed from the prefix FE80::/10) and a globally routable address, constructed from the globally routable prefix of the LoWPAN. In the standard Neighbor Discovery protocol, routers periodically send router announcements (RA) multicast messages and the nodes, in case they don't want to wait for the periodical RA, can solicit a router with a Router Solicitation (RS) multicast message. Additionally, there is a space where a 4-bit Context ID number (CID) can be given: this makes the prefix supplied also available for context-based header compression, which works correctly only if all nodes in the 6LoWPAN share a common view of the context. Neighbor Discovery therefore specifies that the complete set of context is to be disseminated in the entire LoWPAN starting from the edge router, which include the context information in their RA in order to make it available to all first-hop routers.

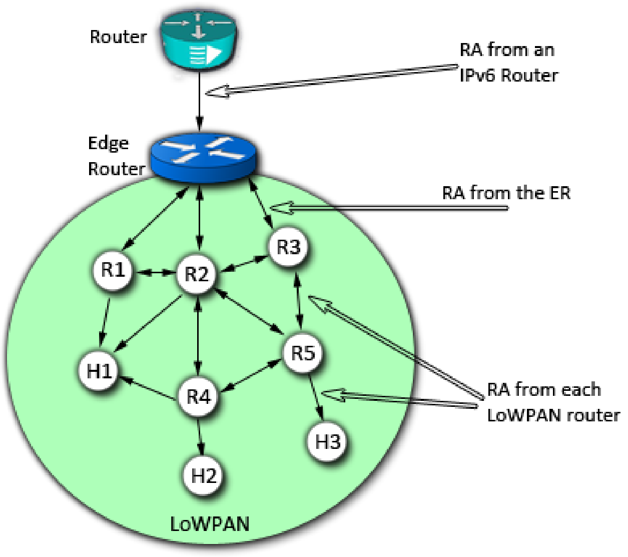


Figure 2.21: Router Advertisement dissemination

The next step after forming an address would be to perform Duplicate Address Detection (DAD) on it, which is done by sending a Neighbor Solicitation (NS) to a solicited-node address, a multicast address formed as a function of the address to be validated. To do this, 6LoWPAN-Neighbor Discovery uses the edge routers as the focal point for DAD: every edge router maintains a whiteboard on which nodes can scribble their address and which other nodes can later read. This is done using two new ICMP6 messages: Node registration (NR) and node confirmation (NC), and the entire process is called registration (Figure 2.22).

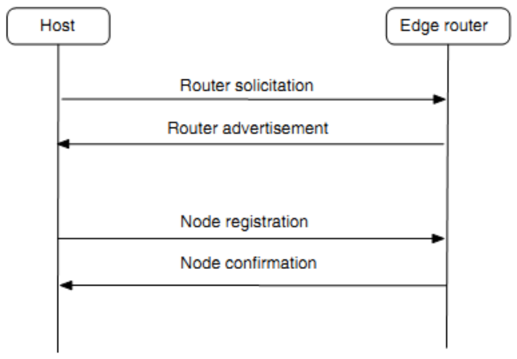


Figure 2.22: Basic router discovery/registration process with an edge router

The whiteboard entries, called bindings, are soft state, which means they need to be refreshed periodically to remain in place. The whiteboard of an edge router serves as a shared database for all nodes that have registered to that edge router and, via the mechanism that hold together an extended LoWPAN for the whole LoWPAN. As the LoWPAN is a distributed system, there is always a possibility of multiple nodes trying to create entries in that database that are in conflict with each other. There are two levels of collision detection in 6LoWPAN-Neighbor Discovery:

- Address collision detection and resolution: multiple nodes try to register the same IPv6 address, while only one of those should succeed. Each registration is identified by the pair of OII (Owner Interface Identifier)and the IPv6; a node registration that tries to register an IPv6 address which is already registered with a different OII is denied.

- OII collision detection: the address collision detection and resolution mechanism are based on the assumption that the OII is globally unique from the way the EUI-64 are allocated. However if an error is allocating or storing EUI-64 causes two nodes to share an OOI, address collision detection breaks down, leading to malfunctioning of the LoWPAN.

The standard Neighbor Discovery protocol makes the responsibility of each node to find other nodes in conflict with it, while 6LoWPAN-Neighbor Discovery supports conflict detection with boot-time owner nonces: random numbers generated each time a node starts up. The owner nonce is used to set up a registration and maintain it; in addition to the nonce, an 8-bit sequence number called Transaction ID (TID) serves to correlate consecutive messages from one node. The TID is sent in each Node Registration message and echoed back by the Edge Router in the Node Confirmation message. The registration process becomes slightly more complicated when the node registering is not adjiacent to an edge router. Nodes can register to a LoWPAN router in the one-hop neighborhood as long as that router indicates its ability to handle the registration; the router then relays the NR to the edge router and the NC back to the node, as shown in the Figure 2.23.

Given that there may be multiple routers in the one-hop neighborhood available for registration, the host is supposed to choose the "best" one: edge routers that don't have a good reason to go incognito identify themselves in their RA message by setting the default router preference field to high (01), while it is set to medium (00) for RAs from other LoWPAN routers.
On the other hand, if no routers are in the one-hop neighborhood the ER metric in the 6LoWPAN prefix summary option, a 16-bit unsigned integer indicating how good this router is at the next hop towards the outside world,

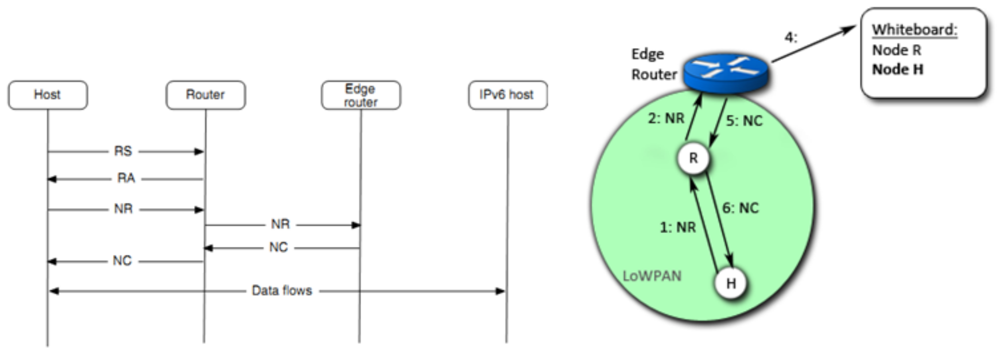


Figure 2.23: Nodes registration with an Edge Router

comes in handy. This field is usually set to 0 as they are by definition the best way out of the LoWPAN.

2.4.3.8 Node, router and edge router operations

LoWPAN nodes start operation by auto-configuring the link local address derived from the globally unique EUI-64 of the LoWPAN interface. From the point of view address and requires confirmation via an NR/NC message exchange with an edge router before becoming fully operational (preferred). When the Node Confirmation message arrives and indicates success, the node takes note of its new 16-bit short address, adds the corresponding IPv6 address to its LoWPAN interface and changes the status of the addresses from optimistic to preferred. As a result of the way LoWPAN nodes generate their IPv6 addresses, there is a one-to-one mapping between link layer addresses and corresponding link-local addresses. A node knows whether another node is in the one-hop neighborhood or needs it be addressed via a router because the node may have cached information about its one-hop neighborhood from the previous packet received. The usual behavior is to send the packet to the default router selected with the ER metric criterion in the absence of any cache entry. In addition to unicast addresses, nodes need to support the all-nodes multicast address FF02::1 is used for receiving RAs from routers. Multicast addresses are always considered to be on-link and are resolved as specified in the 6LoWPAN format specification.
A LoWPAN router begins operations like any other LoWPAN node: it sets up its interfaces and their addresses, performs the required Duplicate Address Detection with the whiteboard and then start running the routing protocol; then it can advertise its services to other nodes using periodic router Advertisement and by listening to Router Solicitations. Another duty of the Router is to relay Node Registration messages from adjacent nodes to an

edge router and relay back Node Confirmation messages to the originating node.
The edge router is characterized by the presence of multiple interfaces used to run the communication. One of these is connected to a larger IPv6 network via a simple backhaul link, making the LoWPAN a simple LoWPAN, or it can be a backbone link connecting to other edge routers in the same extended LoWPAN. In the case of a single edge router, it has to run the whiteboard and the two conflict detection algorithms, is the source of the network parameters disseminated in the Router Advertisement and performs the routing from other IPv6 networks into a back out of the LoWPAN. In an extended LoWPAN, multiple edge routers are interconnected with a backbone link. The backbone link has the same prefix as the LoWPAN, requiring the edge routers to bi-directionally translate between 6LoWPAN-ND on the LoWPAN interfaces and standard Neighbor Discovery on the backbone link. The collision detection algorithms are extended over the backbone link and the neighbor solicitations on the backbone link are answered by the edge routers.

2.4.3.9 Security in 6LoWPAN

The security objectives of a wireless system can be grouped into three categories: confidentiality , which means that data can't be overheard by unintended listeners, achieved by cryptographic encryption; integrity, which means that data cannot be altered by unauthorized parties and which can be achieved by cryptographic integrity checks to messages; availability, which means that the system is not subject to denial of service attacks.
Layer-2 security mechanism goal is to increase robustness against attacks on confidentiality and integrity. The encryption mechanism chosen is based on the AES (advanced encryption standard) algorithm. IEEE 802.15.4 uses AES in the counter with CBC-MAC mode, which provides not only encryption but also integrity check mechanism. AES/CCM encrypts a message m and authenticates that together with additional authenticated data a, using a secret key K and a nonce N. A parameter L controls the number of bytes used for counting the AES blocks in the message and m must be shorter than 28L bytes. For a IEEE 802.15.4 packet, the smallest value of L=2 is plenty. The nonce N is of length 15-L, i.e. 13 bytes for IEEE 802.15.4. The result of AES/CCM is an encrypted message of the same length as m as well as an authentication value of length M bytes, where M is a parameter that can be any even value between 4 and 16. The authentication value can only be created correctly if K is known.
Layer-3 mechanism has the goal to bring end-to-end security mechanism and

the way to achieve this was ported from an IPv4 feature known as IPsec. This system has two main components: the first one regards packet formats and related specifications that define the confidentiality and integrity mechanism for the actual data, while the second one regards a key management scheme called IKE (internet key exchange). IPsec defines two packet formats for cryptographically protected data: the IP authentication header, which provides integrity protection and authentication only, and the IP encapsulating security payload (ESP), which combines this function with confidentiality protection through encryption and which is more popular in the 6LoWPAN systems.
In summary, ESP with AES/CCM is not too heavyweight for end-to-end encryption and integrity checking between LoWPAN and its correspondent nodes. The per-packet overhead could be a bit less with 1-4 bytes of overhead for padding and eight bytes for the explicitly transmitted initialization vector.

2.4.3.10 Mobility

Mobility in IP networks is the act of a node changing its topological point of attachment. There are several causes of mobility in low power wireless networks:

- Physical movement: nodes in the network move in relation to each other
- Radio channel: changes in the environment which cause changes in radio propagation, called fading
- Network performance: packet loss and delay on wireless networks may be caused by poor signal strength, collisions, overloaded channel capacity or node congestion
- Sleep schedules: if a node finds itself attached to a sleeping router without a suitable duty cycle for the application, this may cause the node to move to a better point of attachment
- Node failure: due to battery depletion, nodes tend to be prone to failure. Therefore the failure of a router causes a topology change for nodes using it as default router

There is another class of mobility in which an entire network moves its point of attachment, named network mobility, which occurs when an edge router changes its point of attachment while the nodes in the LoWPAN remain

attached to it. When the ipv6 address of the edge router changes, this affects the addressing of all nodes in the LoWPAN.
Two kinds of mobility can be defined:

- Roaming: a mobile node moves from one network to another, typically with no existing packet stream
- Handover: a process in which a mobile node disconnects from its existing point of attachment and attaches itself to a new point of attachment.

Mobility can also be described in the terms of micro and macro mobility, where the first one refers to mobility that occurs within the network domain and where the IPv6 prefix does not change. On the other hand, macro mobility refers to mobility between LoWPANs in which the IPv6 prefix changes. The macro-mobility process requires both roaming and handover, while micro-mobility requires only handover. When micro-mobility occurs between attachment points that are part of the same link, the link layer may be able to deal with the mobility without any noticeable changes to the network layer.

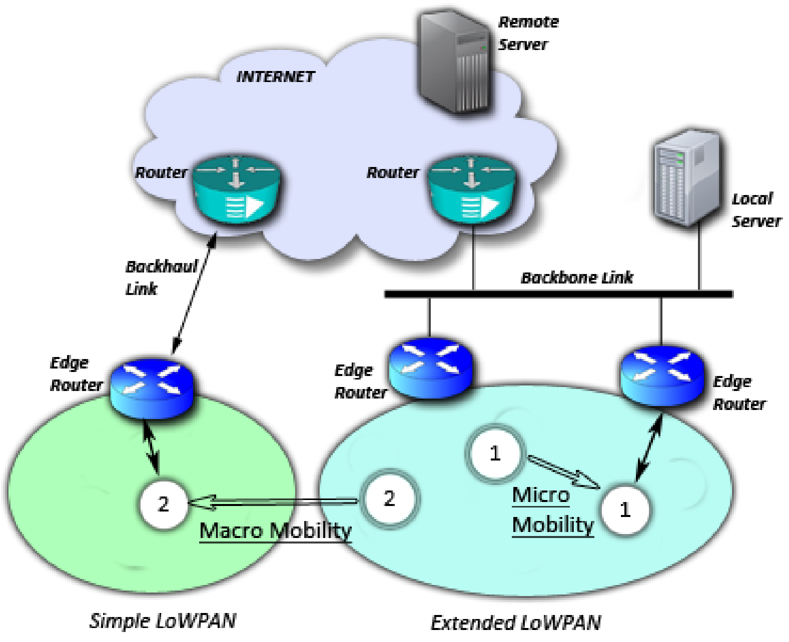


Figure 2.24: 6LoWPAN Micro and Macro Mobility

Low power wireless technologies like IEEE 802.15.4 tend to leave mobility to be dealt with by the network layer. Topology changes are node controlled (such as in wifi) rather than network controlled (such as in cellular system). Neighbor discovery for 6LoWPAN includes a built-in feature for dealing with micro-mobility in extended LoWPAN topologies . This is achieved using a Neighbor Discovery proxy technique and whiteboard synchronization between edge routers, allowing a node to keep the same IPv6 address regardless of its point of attachment within the extended LoWPAN. Macro-mobility always involves a change of IPv6 address for a node. The simplest way to deal with this is to simply restart when detecting an IP change. This is used when the node acts as a client, but if it acts as a server and must therefore be reachable by any time, macro-mobility is a real challenge. One way of dealing with this is at the application layer, using for example the session initiation protocol (SIP), uniform resource identifiers (URI), or a domain name server (DNS).
Mobile IPv6 provides one way of dealing with this at the network layer by maintaining a home address on behalf of mobile nodes which does not change, thus requiring DNS change.
When handover occurs one thing that must be considered is how the transport layer protocol reacts to a change in IP address for one of the end-to-end points. 6LoWPAN applications often make use of UDP as a transport, which is resilient to changes in IP address as each datagram is independent. An application using UDP still needs to deal with the change in IP address, correlating the new address to the same end-point. In order for an application server communicating with a mobile node with changing IPv6 address to function, it needs to use some sort of unique and stable identifier for each node: for example the EUI-64 of the node interface, or the domain name resolved using the DNS. Care must be taken when using the DNS for such purposes as updates may not propagate immediately, although by using a client to send dynamic updates and with careful Time to Live (TTL) settings.
The mobility of nodes on the internet can be dealt with at the network layer using a protocol called Mobile IP and updated to Mobile IPv6. The goal of this protocol is to allow a host to be contacted using a well-known IP address, regardless of its location on the internet, using the concept of a home address, which is associated with a host's home network. When a host is away from his home network and attaches to another network domain(visited network), the new IP address is called its care-of address. A node communicating with a mobile node roaming in a visited network is called the correspondent node. Mobile IP works using a special kind of routing functionality (which is called binding), which is host controlled and is implemented by an entity called the

Home Agent that must be present in the home network domain of a mobile node in order to use Mobile IP. The home agent is responsible for maintaining a binding between a node's permanent home address and the temporary care of address used while roaming in a visited network. The home agent therefore acts as a forwarder of traffic.

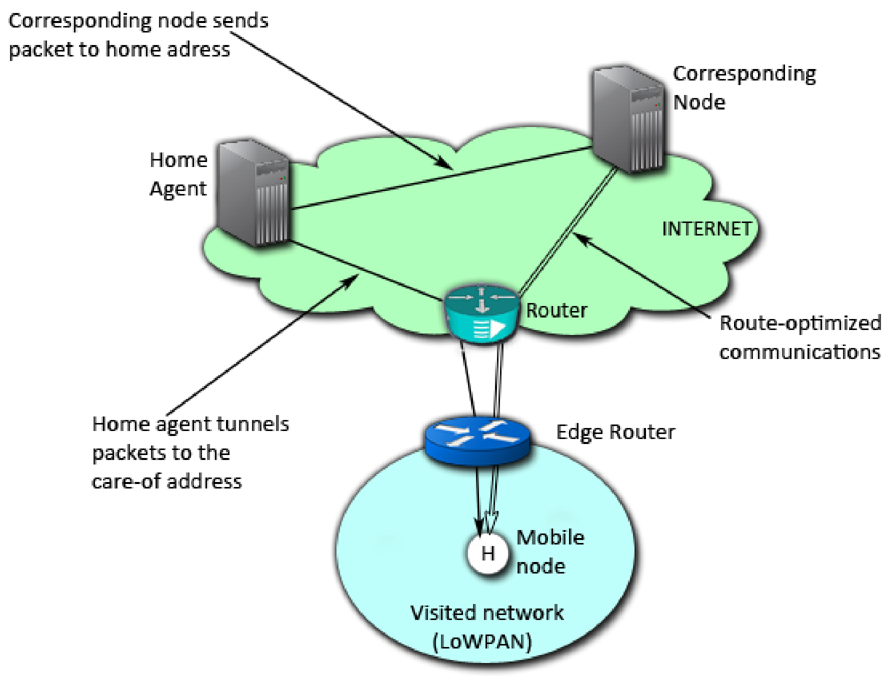

Corresponding node sends packet to home adress
Corresponding Node
Home Agent
INTERNET
Router
Route-optimized communications
Home agent tunnels packets to the care-of address
Edge Router
H
Mobile node
Visited network (LoWPAN)

Figure 2.25: Mobile IPv6

The use of Mobile IPv6 with 6LoWPAN has the following problems:

- IPv6-in-IPv6 tunneling between the Home Agent and the LoWPAN node would need a large header overhead
- The requirement of IPsec security association between MIPv6 entities may be unreasonable for LoWPAN nodes
- The added complexity of implementing MIPv6 in term of code size and RAM is unjustifiable for LoWPAN nodes
- In domains with large LoWPAN and frequent mobile nodes, the traffic caused by MIPv6 could be high for a low-bandwidth link

A better solution for network mobility in LoWPAN is available using the Network Mobility Protocol (NEMO). The philosophy behind NEMO is to extend Mobile IP so that each node does not need to run Mobile IP, but only the router they are attached to runs Mobile IP. This approach fits perfectly with the 6LoWPAN model because the nodes are not capable of dealing with Mobile IPv6, while edge routers or other router entities run full IPv6 stacks and have the capability to deal with Mobile IPv6. The NEMO protocol works by introducing a new logical entity called the mobile router, which is responsible for handling Mobile IPv6 functions for the entire mobile network. Mobile IPv6 normally only handles forwarding for the home address bound by mobile nodes, while NEMO extends the functionality of the Home Agent to be able to deal with prefixes in addition to home address of mobile nodes. A mobile router works like a normal Mobile IPv6 host, setting up a bidirectional tunnel with its Home Agent, but in addition it negotiates prefixes to be forwarded to it by the Home Agent, which forwards then all packets matching the bound prefix to the mobile router.

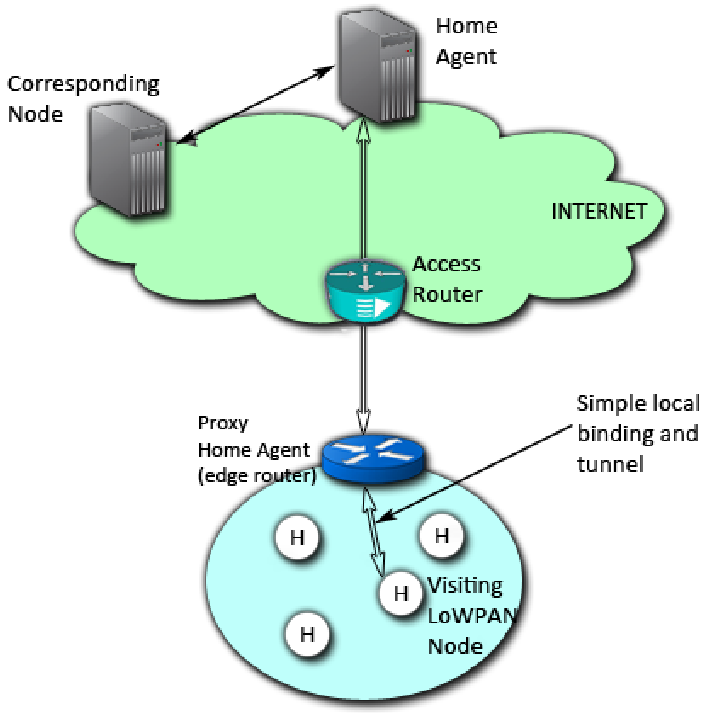


Figure 2.26: Network Mobility Protocol

2.4.3.11 Routing

While IP mobility considers technique for preserving the IP address as it moves from one point to another, IP routing deals with maintaining routing tables on IP routers which indicate which next-hop forwarding decision should be made for the destination of an IP packet. As IP networks are packet switched, forwarding decisions are made hop-by-hop, based on the destination address in a packer. In 6LoWPAN IP addresses are structured and this structure is used to group addresses together under a single route entry. In IPv6 an address prefix is used for this purpose, which is why it's called prefix-based routing. Two types of routing can be considered in a 6LoWPAN network: routing inside the LoWPAN and routing between a LoWPAN and another IP network.

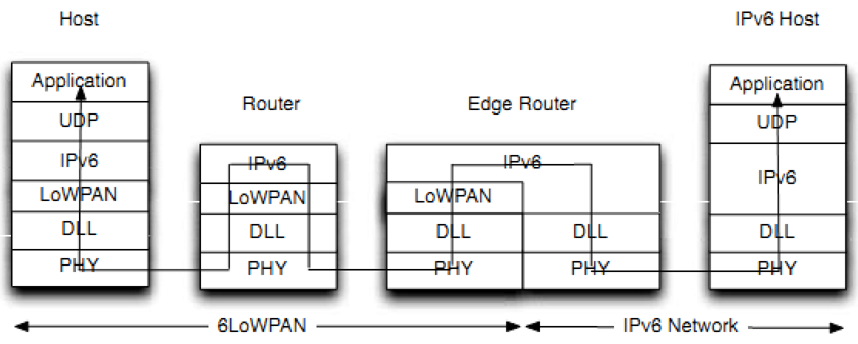


Figure 2.27: Routing in a 6LoWPAN network

There are then two main classes of routing protocols in 6LoWPAN: distance-vector routing and link-state routing.
In distance-vector routing, to each link is assigned a cost using appropriate routing metrics and when a packet has to be sent from node A to node B, the path with the lowest cost will be chosen. The routing table of each router keeps soft-state route entries for the destination, with the associated path cost. Depending on the routing algorithm, routing information is updated either proactively (a priori) or reactively (on demand). Algorithms using a proactive approach build up routing information on each node before the routes are needed. Thus they proactively prepare for the data traffic by learning routes to all possible or likely destinations. Most of the protocols that are intra-domain IP use this approach as the topologies are stable. The advantage of this approach is that routes are immediately available when needed, but this comes at the cost of increased signaling overhead, especially with frequent topology changes and increased state for routers. Reactive routing protocols, on the other hand, store little information after auto-configuration

of the routing protocol. Routes are discovered dynamically only at the time they are needed and the advantage of this approach is that signaling and route state grows only as needed, so it's well suited to ad hoc networks with frequent topology change. The main advantages of the distance-vector routing technique is the simplicity, the local adapted nature and the low signaling overhead.
In link state routing each node acquires complete information about the entire network, called graph. In order to do this each node floods the network with information about its link information to nearby destinations. After receiving link-state reports from sufficient networks, each node calculates a tree with the shortest-path from itself to each destination using algorithms like Dijkstra. This tree is used either to maintain the routing table in each node or to include a source-route in the header of the IP packet. This kind of algorithms incur to a large amount of overhead, especially in networks with frequent topology change and they require substantial memory resources. Thus they are not suitable for LoWPAN nodes, but may be usefully applied offline to edge routers which have sufficient memory capacity for collecting the link-state information.
In the path selection process, route metrics are used to choose the best route. Typical metrics for IP routing include hop count, bandwidth, delay, MTU and reliability.
The metrics considered can be classified as follows:

- Link versus node metrics
- Qualitative versus quantitative metrics
- Dynamic versus static metrics

Example of link metrics include throughput, latency and link reliability. Node metrics may include memory, processing load and residual energy. The metrics identified are summarized in the table below.
Legend: QT=Quantitative, QL=Qualitative, ST=Static, DY=Dynamic

Metric	**Type**	**Description**
Node memory	QT, ST	The memory available for routing information on a node
Node CPU	QT, ST	Computational power
Node energy	QT, DY	The residual energy left for battery-powered nodes, important for optimizing network lifetime
Node overload	QT, DY	A simple indication of the network load (e.g. queue size) of a node
Link Throughput	QT, DY	The total and currently available throughput of a link
Link Latency	QT, DY	The range of latency and current latency of a link
Link reliability	QT, DY	The link reliability specified as e.g. average packet error rate, which is a critical routing metric
Link coloring	QL, ST	This static attribute is used to prefer or avoid specific links for specific traffic types

Most of these metrics are used for building and maintaining the routing topologies, while others are used for making forwarding decisions. The ROLL/RPL routing protocol uses a very granular depth metric for building the basic topology. The use of dynamic metrics is particularly challenging, as it may lead to routing instability; moreover a set of metrics will be used for path calculation and it is important that this calculation is consistent throughout the same routing domain.

2.4.3.12 Border Routing

Border routing between two IP routing domains is a common issue on the Internet, where intra-domain and inter-domain routing protocols intersect.

With the advent of IP routing in wireless stub networks using mesh routing protocols border routing becomes an issue as well. In 6LoWPAN there are three border routing cases to consider: simple and extended LoWPAN and router redistribution. In simple LoWPAN there is only an edge router and the subnet of its LoWPAN interface is different from that of its IPv6 interface. As the simple LoWPAN and the IPv6 link are on different subnets, prefix-based route entries between the two prefixes are used. Moreover, it is mandatory that edge routes filter out any outgoing traffic from or incoming traffic to addresses not in its whiteboard. Therefore it is not necessary to run a routing protocol on the edge router's IPv6 interface.

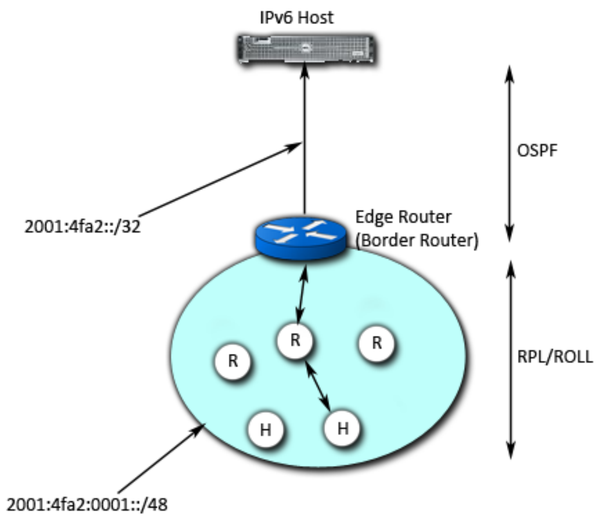


Figure 2.28: 6LoWPAN Border Routing Example

In the Figure 2.28 an example of border routing in the Simple LoWPAN case is shown, with ROLL routing in the LoWPAN and OSPF (open shortest path first) routing on the backhaul link. The route to 2001:4fa2:0001::/48 would be redistributed to OSPF.
Border routing between extended LoWPAN and an IP network can be performed in two different places as the LoWPAN and the ipv6 interfaces of edge routers are in the same subnet, but it must be done using destination route entries. The simplest way to achieve this is to use edge router whiteboard entries to maintain these route table entries and a routing protocol could be used on the backbone link to enable routing over the backbone between parts of the extended LoWPAN. Border routing could also be performed on the router between the backbone link and another IP network, while routing here would be achieved using prefix-based route entries, like in the simple LoWPAN case.

Route redistribution has to be performed if border routing involves routing algorithms on both its interfaces. In this case, a router advertises some routes maintained by an algorithm on one interface into the algorithm of another interface. Because LoWPANs are stub networks, this redistribution always happens from the LoWPAN routing protocol to the protocol on an IP interface.

2.4.3.13 Application protocols

Application protocols can be defined as all the messages and methods having to do with inter-process communication via the Internet Protocol. The application layer depends on the transport layer to provide host-to-host communication and port multiplexing allowing multiple processes to communicate between end-points simultaneously. The limitations of 6LoWPAN, such as small frame sizes, limited data rates, limited memory, sleeping node cycles, along with the mobility of devices make the design of new application protocols and the adaptation of existing ones difficult.
Web services enable the communication between processes using well-defined message sequences with the simple object access protocol (SOAP) or stateless resources with representational state transfer (REST) style design. What makes 6LoWPAN very different from other solutions is that the same network can be used by a large variety of devices running different applications thanks to the internet model. Although the internet protocol provides basic packet networking over heterogeneous links, it is UDP and TCP that allow for the large range of application protocols by providing best-effort (UDP) and reliable connection oriented (TCP) multiplexed communications between application processes. IP protocols use a socket-based approach, where process end-points are identified by 16-bit source and destination port identifiers. These are commonly called Internet sockets or network socket and the communication between two end-points is uniquely identified for each transport by a four-tuple consisting on the local and remote socket addresses:
$\{sourceIPAddress; sourcePort; destinationIPaddress; destinationPort\}$
6LoWPAN supports the compression of UDP ports down to a range of 16, which is useful because LoWPAN usually has a limited number of applications. Application protocols used over 6LoWPAN need to take a number of requirements into account which are typically not an issue over general IP networks, which include:

- Link layer: issues include lossy asymmetrical links, typical payload sizes of 70-100 bytes, limited bandwidth and no native multicast support. The most limiting feature of ISM band radios is their small frame size:

IEEE 802.15.4 has a physical layer payload size of 127 bytes in length, resulting in 72-116 bytes of available UDP payload depending on the MAC and 6LoWPAN features in use.

- Networking: issues include the use of UDP, limited compressed UDP port space and performance issues regarding the use of fragmentation. If the UDP source or destination ports are compressed then the port space can be limited down to 16 ports. The fragmentation of large payloads increases delay, packet loss probability and congestion.

- Host issues: 6LoWPAN hosts and networks are often mobile, therefore the use of an IPv6 address for identifying 6LoWPAN devices is not recommended. The IP address changes each time the LoWPAN node or the whole LoWPAN changes its point of attachment. A unique serial number such as the EUI-64 of the device is a reliable identifier. The most application-friendly method is to use a domain name to identify a device, which is updated with the current IPv6 address of the device each time it moves, using appropriate DNS technique. Furthermore battery powered nodes use sleep periods with duty cycles often between 1 and 5%. The intermittent node availability due to mobility and sleep schedules needs to be taken into account during application design. For example the synchronous polling of LoWPAN nodes from a server should be avoided. Instead communication should be node initiated and asynchronous when possible.

- Compression: issues include header and payload compression and whether it is performed end-to-end or by an intermediate proxy.

2.4.4 IEEE 802.15.4a - Ultra Wide band

The IEEE 802.15.4a Standard provides an alternative physical layer for low rate WPAN and is an amendment of the IEEE 802.15.4 Standard. The Direct Sequence Ultra Wide Band approach was chosen for the standard due to its spectral efficiency, robustness at low transmit powers and support for high-precision ranging. UWB PHY waveforms employ an impulse radio scheme.
The standard specifies three independent frequency bands and a total of 16 channels (or 32 complex channels). A single mandatory channel is specified for each band, one of which a compliant device must implement to adhere to the standard. Inside each channel there is support for two complex channels. A complex channel has a unique 31 bit preamble code used to construct the synchronization header part of a UWB PHY frame. The channel number together with the preamble code makes up a complex channel.
Ultra Wide Band is a spread spectrum technology, a Radio Frequency communication technology in which the bandwidth of the baseband signal is intentionally spread over a larger bandwidth by injecting a higher-frequency signal. Consequently, the transmit energy is spread over a wider bandwidth and the signal appears as noise. UWB differs from conventional spread spectrum technologies because in a UWB system the information is transmitted through a series of short pulses or a "chirped" signal while with traditional spread spectrum systems information is transmitted by modulating a continuous carrier signal.
The energy contained in a single pulse is very low, therefore in order to extract information out of UWB transmitted data and to fight noise and interference, a technique called spreading is used. A symbol to be transmitted is represented by a number of consecutive pulses instead of a single pulse. The pseudorandom bit sequence determining the polarity of these pulses is referred to as spreading code. In order to restore the original bit, i.e. to despread the sequence of pulses, a cross-correlation operation $\sum_{i=1}^{N}(x_i c_i)$ of N pulses x with the spreading code c, also of length N, is calculated. The value of the decision variable y determines the value of the original bit.

2.4.4.1 IEEE 802.15.4a Frame Structure

In IEEE 802.15.4a UWB IR systems data is being transmitted in frames consisting of three major sub-parts:

- Synchronization header (SHR) / Preamble
- Data header (PHR)
- Data unit, i.e. the actual payload data (PSDU)

The synchronization header, also referred to as preamble, is being transmitted in order to aid receiver algorithms intiming acquisition and frame synchronization. It also serves the purpose of channel estimation and gain control setting optimization. This preamble is composed of SHR symbols that contain a number of isolated pulses.
The data unit and its header are constructed out of PSDU symbols. Each symbol is able to carry two bits of information: one bit is coded in the symbol half in which a burst, i.e. a concatenation of pulses, occurs (burst position modulation, BPM). Another bit is used to determine the polarity (phase) of the burst itself (binary phase shift keying, BPSK). The state of a linear feedback shift register (LFSR) varies the spreading code for each transmitted PSDU symbol. This spreading code determines the burst sequence as well as the exact burst position within one of the symbol halves ("hopping code"). A PSDU symbol is depicted in the Figure 2.29

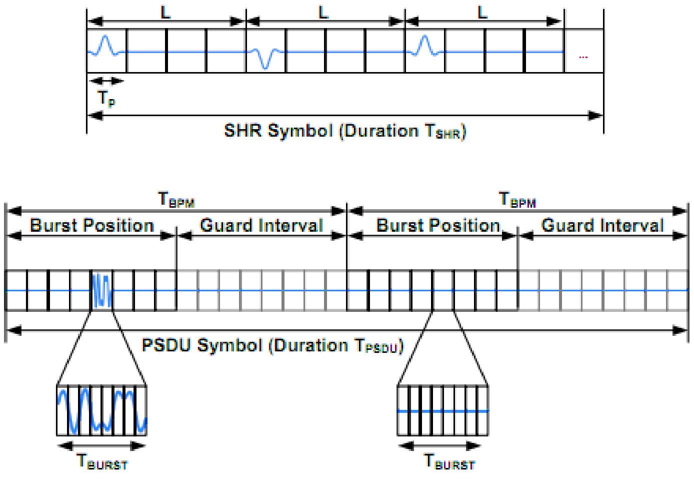


Figure 2.29: Ultra Wide Band PSDU Symbol

The 802.15.4a compliant Ultra Wide Band Integrated Circuits available in the market are still very expensive. Two of those ICs can be mentioned: *IMEC Digital UWB Transmitter IC* and the *TES IEEE 802.15.4a transceiver with ranging capability.* The first one, in order to reduce the startup time and reduce energy, uses a phase-aligned Frequency Locked Loop (FLL) instead of the traditional Phased Locked Loop (PLL). The second one implements the complete LR-UWB functionalities specified in the IEEE 802.15.4a standard, including localization and tracking algorithms, AES security engine, Embedded Flash/RAM and signal conditioning and directional UWB antennas.

2.4.5 Wireless Hart

Wireless Highway Addressable Remote Transducer (Wireless HART) is a proprietary wireless communications standard suitable for industrial applications such as process control, measurement and management applications. The main characteristics of this standard are its reliability, security, compatibility with existing devices and its energy efficiency. The reliability is traduced by its capability of coexisting with other wireless networks in the vicinity (immunity against interference) thanks to adopted modulation scheme (channel hopping) and the usage of time-synchronized messaging.
The security is ensured through the usage of encryption, authentication, key management, and other open industry-standard security practices. And the energy efficiency is guaranteed by the Smart Data Publishing and other techniques that make batteries, solar and other low-power options practical for wireless devices. A WirelessHART network consists of three basic components:

- Wireless field devices connected to process or plant equipment. This device could be a device with WirelessHART built in or an existing installed HART-enabled device with a WirelessHART adapter attached to it.
- Gateways enable communication between these devices and host applications connected to a high-speed backbone or other existing plant communications network.
- A Network Manager is responsible for configuring the network, scheduling communications between devices, managing message routes, and monitoring network health. The Network Manager can be integrated into the gateway, host application, or process automation controller.

The network uses IEEE 802.15.4 compatible radios operating in the 2.4GHz ISM band, with the Time Division Multiple Access (TDMA) for access to communication medium. The complete time of communications executes inside predetermined time slots of 10ms. Each device in the mesh network can serve as a router for messages from other devices. In other words, a device doesn't have to communicate directly to a gateway, but just forward its message to the next closest device. This extends the range of the network and provides redundant communication routes to increase reliability.
The Network Manager determines the redundant routes based on latency, efficiency and reliability. To ensure the redundant routes to remain open and unobstructed, messages continuously alternate between the redundant paths.

Consequently, like the Internet, if a message is unable to reach its destination by one path, it is automatically re-routed to follow a known-good, redundant path with no loss of data.
The WirelessHART standard supports multiple messaging modes including one-way publishing of process and control values, spontaneous notification by exception, ad-hoc request/response, and auto-segmented block transfers of large data sets. These capabilities allow communications to be tailored to application requirements thereby reducing power usage and overhead.
To increase reliability it is used technique of hopping between transfer channels which makes possible the work on different frequencies i.e. that different appliances transfer messages in the framework of same time slot using different transfer channels. In this way appearance of interferences is avoided and the multi-path fading effect decreases.
Data transfer from all communication appliances in WirelessHART network passes across the gateway which must direct packages towards before hand quoted destination. Gateway uses standard HART commands for communication with network devices and host applications. The network manager creates start superframe and configures the HART Gateway.

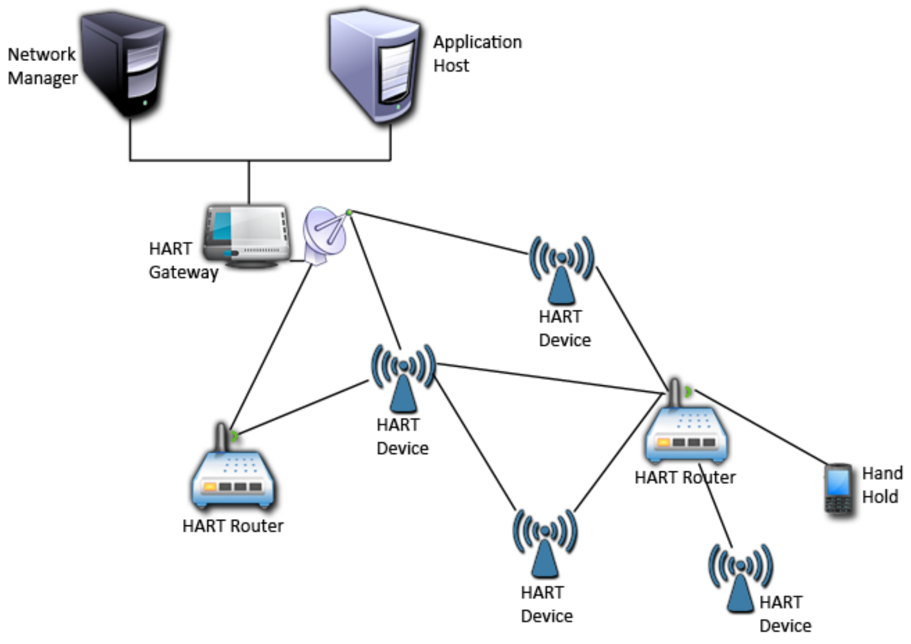


Figure 2.30: A WirelessHART Network

2.5 Z-Wave

Developed by a Danish company, Zensys, Z-Wave protocol is a low bandwidth half duplex proprietary protocol designed for reliable wireless communication in a low cost control network. The protocols main purpose is to communicate short control messages in a reliable manner from a control unit to one or more nodes in the network. Z-Wave is not designed to transfer large amounts of data or to transfer any kind of streaming or timing critical data.
The protocol consists of 4 layers, the MAC layer that controls the RF media, the Transfer Layer, that handles frame integrity checks, acknowledgements, and retransmissions, the Routing Layer that controls the routing of frames in the network and the application interface; and finally the Application Layer controls the payload in the transmitted and received frames (Figure 2.31).

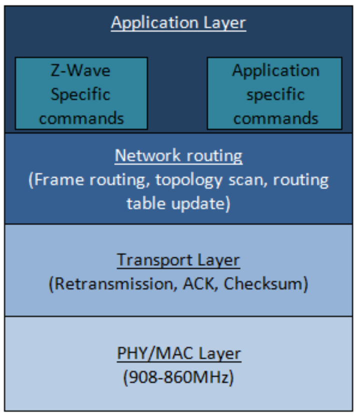

Application Layer
Z-Wave Specific commands
Application specific commands
Network routing
(Frame routing, topology scan, routing table update)
Transport Layer
(Retransmission, ACK, Checksum)
PHY/MAC Layer
(908-860MHz)

Figure 2.31: Z-Wave Protocol Stack

The modulation used by Z-Wave for communication is the Gaussian Frequency Shift Keying (GFSK) and the maximum bandwidth, which depends on the ISM band chosen, can be 9,600 bit/s or 40 kbit/s.
The frequency bands used by Z-Wave are the 900 MHz ISM band: 908.42 MHz (United States); 868.42 MHz (Europe); 919.82 MHz (Hong Kong); 921.42 MHz (Australia/New Zealand). In Europe, the 868 MHz band has a 1% duty cycle limitation, meaning that a Z-Wave unit can only transmit 1% of the time. This limitation is not present in the U.S. 908 MHz band, but U.S. legislation imposes a 1 mW transmission power limit. Z-Wave units can be in power-save mode and only be active 0.1% of the time, thus reducing power consumption dramatically. The maximum distance range is estimated to be around 30 meters in open air conditions,with reduced range indoors depending on building materials etc.

2.5.1 Topology and Routing

Z-Wave uses a source-routed mesh network topology and has one or more master controllers that control routing and security. A Device can communicate to another by using intermediate nodes to actively route around obstacles or radio dead spots that might occur. Therefore a Z-Wave network can span much farther than the radio range of a single unit; however with several of these hops a delay is introduced between the control command and the desired result. In order for Z-Wave units to be able to route unsolicited messages, they cannot be in sleep mode. Therefore, most battery-operated devices are not designed as repeater units. A Z-Wave network can consist of up to 232 devices with the option of bridging networks if more devices are required.

2.5.2 Security

The original version of Z-Wave used a security algorithm called Triple Data Encryption Algorithm (DES) with a 56-bit key, but in the revised version it was not implemented anymore. Instead of the DES sexurity, an optional security method named "rolling code" was implemented, which works in the following way: when the transmitter button is pressed, it sends a 40-bit access code along with the function instruction (to open the car door or garage door, etc). The receiver holds the current 40-bit access code in memory, and if it receives the same code then it accepts the instruction. Both the transmitter and the receiver use the same pseudo-random number generator to pick identical new codes each time and access code is received. So the transmitter and the receiver are synchronized. In case the transmitter is pressed and for some reason the receiver does not pick up the signal, the synchronization of the two security codes will be lost. The system gets around this by accepting any of the next several codes in the sequence. Suppose the system is set to accept the next 256 codes generated by the pseudo-random number generator. If someone accidentally sets off the transmitter 257 times without a successfully reception by the receiver, then the transmitter will be ignored. By dropping hardware security from its series 200 Z-Wave chips, Zensys achieved a 35% saving in the amount of code that has to be saved on the chip.

2.6 Bluetooth and Bluetooth Low Energy

Bluetooth, also known as the IEEE 802.15.1 standard, is based on a wireless radio system designed for short-range and cheap devices to replace cables for computer peripherals, such as mouse, keyboards, joysticks, and printers. This range of applications is known as wireless personal area network (WPAN).
Two connectivity topologies are defined in Bluetooth: the piconet and scatternet. A piconet is a WPAN formed by a Bluetooth device serving as a master in the piconet and one or more Bluetooth devices serving as slaves (Figure 2.32). A frequency-hopping channel based on the address of the master defines each piconet and all the devices participating in a communication in a given piconet are synchronized using the clock of the master. Slaves communicate only with their master in a point-to-point fashion under the control of the master. The master's transmissions may be either point-to-point or point-to multipoint. Also, besides in an active mode, a slave device can be in the parked or standby modes so as to reduce power consumptions.

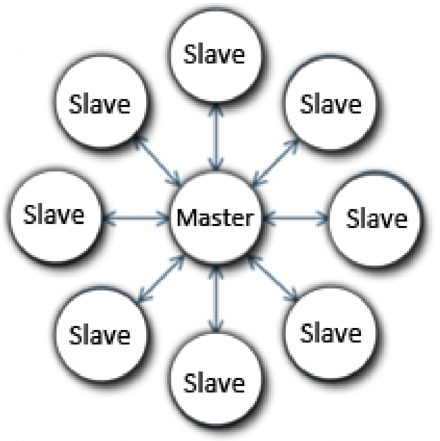


Figure 2.32: Bluetooth Piconet

A scatternet is a collection of operational Bluetooth piconets overlapping in time and space (Figure 2.33). Two piconets can be connected to form a scatternet. A Bluetooth device may participate in several piconets at the same time, thus allowing the possibility that the information could flow beyond the coverage area of the single piconet. A device in a scatternet could be a slave in several piconets, but master in only one of them.

Much of the way in which the topology works and devices talk to each other stems from the nature of a frequency-hopping network. Adaptive frequency hopping (AFH) works by scanning all of the channels and looking

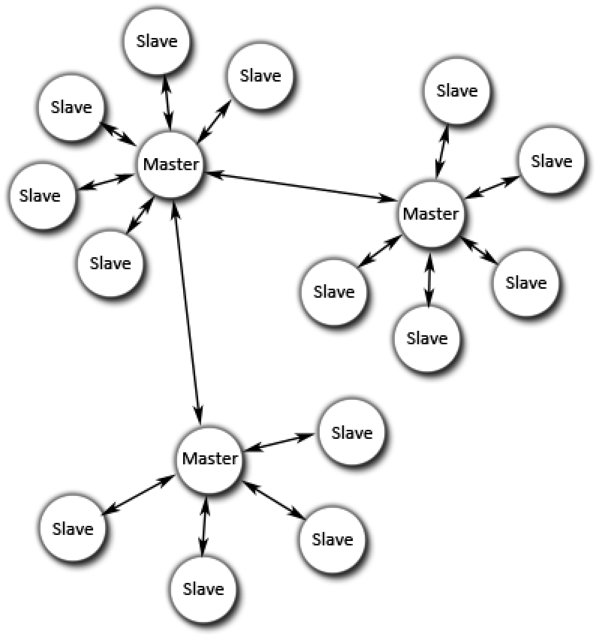


Figure 2.33: Bluetooth Scatternet

for activity on each of them. This is used to build up a picture of channels in use, which is then used to modify the hopping sequence of a Bluetooth piconet, so that these channels are avoided.
Frequency hopping naturally divides transmission up into time-slots, each of which lasts for the duration of one hop. For Bluetooth, the connection scheme starts off with 1600 equally spaced hops every second, resulting in a base time slot of 625 μs.

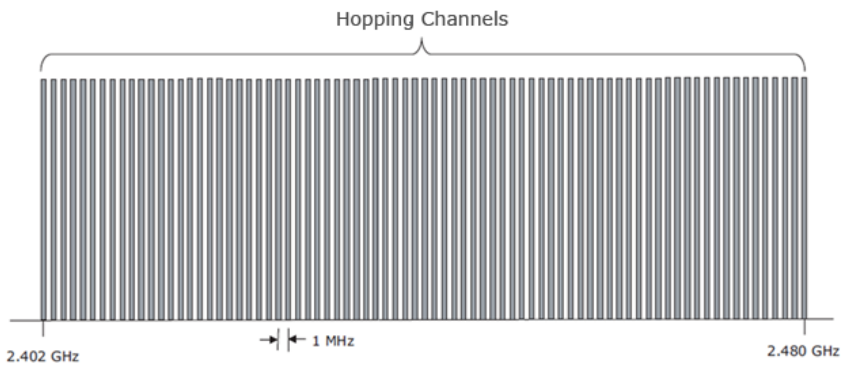


Figure 2.34: Spectrum usage of Bluetooth

The odd-numbered time slots are reserved for the master device, while the even-numbered ones are reserved to slave devices. Slave devices can only transmit in response to a query from the master device. Further, even devices have data to transmit, they must wait until the master device authorize them. Two types of connections are possible: Synchronous Connection Orientated (SCO), or Asynchronous Connection Less (ACL).
The SCO (Synchronous Connection Oriented) channels are used for data or voice streaming by reserving slots which are symmetric between the master and a slave. A Bluetooth master can support three simultaneous SCO channels, which can be split between up to three slaves. Each channel provides a bandwidth of 64 kbps. Packets are neither acknowledgednor retransmitted.
An ACL (Asynchronous Connection Less) channel supplies an asynchronous access between master and slave (a single channel per couple), with the slot as base. The data packets only are used. In addition, a slave can only transmit after having received a packet from the master (the following slot). For this purpose, the master can send polling packets to the slaves (when there is nothing more asynchronous to transmit). Most ACL formats incorporate FEC and header error correction (HEC) to detect and correct errors. Data rates of up to 723 kbps are achievable using ACL links with basic-rate Bluetooth and 2.1 Mbps using enhanced-data rate.

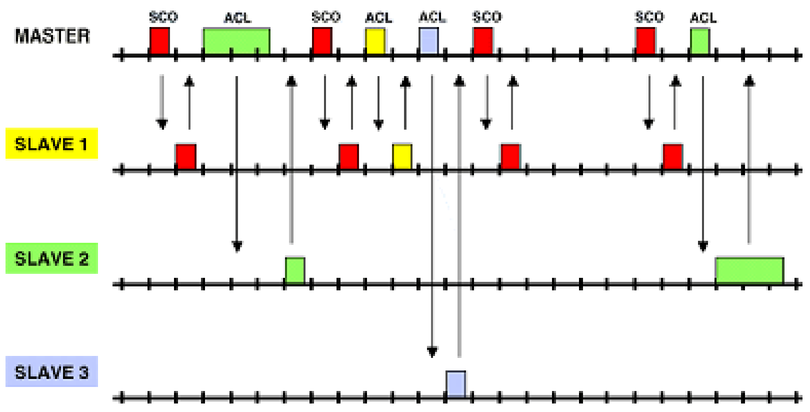


Figure 2.35: Bluetooth Synchronous Connection Oriented and Asynchronous Connection Less Channels

2.6.1 Bluetooth Low Energy

Bluetooth Low Energy, also known as ULP (Ultra Low Power) Bluetooth, operates in the same spectrum range (2402-2480 MHz) as classic Bluetooth, but uses a different set of channels. Instead of Bluetooth's 1 MHz wide 79 channels, Bluetooth Low Energy (BLE) has 40 2 MHz wide channels.
Bluetooth Low Energy is designed with two equally important implementation alternatives: single-mode and dual-mode. Small devices like tokens, watches and sports sensors based on a single-mode Bluetooth Low Energy implementation will enjoy the low-power consumption advantages enabled for highly integrated and compact devices. In dual-mode implementations Bluetooth Low Energy functionality is integrated into Classic Bluetooth circuitry. The architecture will share Classic Bluetooth technology radio and antenna, enhancing currently chips with the new low energy stack and enhancing the development of Classic Bluetooth devices with new capabilities.

Technical Specification	**Classic Bluetooth**	**Bluetooth LE**
Distance/Range	100 m (330 ft)	200 m (660 ft)
Over the air data rate	1-3 Mb/s	1 Mb/s
Application throughput	700Kb/s-2.1 Mb/s	260 Kb/s
Active slaves	7	Not defined; implementation dependent
Security	64/128-bit and application layer user defined	128-bit AES with Counter Mode CBC-MAC and application layer user defined
Robustness	Adaptive fast frequency hopping, FEC, fast ACK	Adaptive frequency hopping, Lazy Acknowledgement, 24-bit CRC, 32-bit Message Integrity Check
Latency (from a non connected state)	Typically 100 ms	6 ms

Total time to send data (det.battery life)	100 ms	6 ms
Voice capable	Yes	No
Network topology	Scatternet	Star-bus
Peak current consumption	<30 mA	<20 mA (max 15 mA to run on coin cell battery)
Service discovery	Yes	Yes
Profile concept	Yes	Yes
Primary use cases	Mobile phones, gaming, headsets, stereo audio streaming, automotive, PCs, security, proximity, healthcare, sports & fitness, etc.	Mobile phones, gaming, PCs, watches, sports and fitness, healthcare, security & proximity, automotive, home electronics, automation, Industrial, etc.

The Ultra Low Power chip's radio wakes up when it has data to transmit and the transmission itself is short. This leads to some coexistence issues with standard Bluetooth: since both modes use the same radio, some accommodation must be made for sharing it. Two elements in the dual-mode protocol stack, Admission Control and the DuMo Manager, have been added to avoid coexistence issue.

2.7 Wavenis

Wavenis is a proprietary ultra-low-power and long-range wireless technology developed by Coronis and currently used in diverse markets where communication ability and device autonomy present conflicting requirements. These markets include telemetry, industrial automation, remote utility meter monitoring, home comfort, alarms for protecting people and property, home healthcare, centralized building management, access control, cold-chain monitoring, as well as long-range UHF RFID applications for the identification, tracking, and locating of people and objects.

Specification	
General	Ultra-low-power and long-range wireless technology Multi-year battery life, 2-way communications Repeater function in every device (up to 4 hops) Complete API (application programming interface) Asynchronous or synchronous (with common realtime clock)
Frequency bands	868 MHz, 915 MHz, 433 MHz Certified ETS300-220, FCC15-247, 15-249
Distance Range	LOS range up to 1km (25mW, +15dBm) LOS range up to 4km (500mW) - not in Europe
Data rates	4.8 kbps to 100 kbps Typical values: 9.6 kbps @ 433 MHz and 868 MHz; 19.2 kbps @ 915 MHz
Link budget	-110dBm @ 19.2 kbps - 50kHz BW channel -113dBm @ 4.8 kbps - 25kHz BW channel

Network Access time	Programmable from 12.8 ms to 12.8s (typically 1.28s)
Power management	Programmable output power from -10dBm to +14dBm Automatic Frequency Control Automatic Sensitivity Control Adaptive Frequency Hopping Receiver Sampling, which reduces average power consumption by a factor of 20-30
Ultra-low-power operation	Smart cyclic standby-receive mode 10μA average operating current with 1s access time 5μA average operating current with 2s access time 17mA RX current in full run mode 45mA TX current (25mW output power) 2μA STANDBY current
Security and robustness	FHSS (Fast Frequency Hopping Spread Spectrum) Single channel operation for narrowband applications GFSK modulation FEC (Forward Error Correction) - BCH(31,21) Data scrambling via LFSR feed Data interleaving (16x16 matrix) 3DES, AES-128, RSA, EMV security algorithms upon request Combined RTS/CTS and TDMA (CSMA/TDMA for broadcast/ multicast) Carrier Sense (CS) optional CSMA/CA(2)
Network	Point-to-point, point-to-multipoint (repeater, polling, broadcast/multicast) Quality of Service management Fast RSSI detection

2.7.1 Wavenis Operation

Wavenis data rates are programmable, from a few to a hundred kilobytes per second. Most Wavenis applications use low data rates (typically, 19.2-38.4 kbps), with higher rates when larger amounts of data are required. The low data rate enables the use of narrowband (and highly sensitive) receivers, with a high link budget, resulting in long-range radio coverage and satisfactory operating ranges. Wavenis maximizes the radio link budget between devices in order to compensate for bad propagation conditions and signal attenuation indoors, and to compensate for poor antenna gain resulting from tiny footprint design and mandatory low-cost solutions. Among other things, this makes it possible to enable wireless communication in hard-to-reach devices without repeater nodes. Wavenis networks are not limited in size and they can range from a few devices to hundreds. However, time-critical applications by their very nature limit cluster size (due to TDMA management when feedback is required after a broadcast command). Nevertheless, a high number of devices can be achieved using time-shifted cluster-tree topology within remotely-monitored fixed networks. With Wavenis, wireless monitoring can be performed from fixed access points and/or from portable or mobile devices. As a result, Wavenis networks can be operated in many scenarios: over LANs (local area networks), via modem, through Compact Flash or SD cards in handheld terminals or PDAs, or by WAN (wide area network) GSM fixed network monitoring.

2.8 Dash7

The goal of DASH7 is to expand the market for low power wireless technologies by leveraging ISO 18000-7. DASH7 is an acronym that stands for "Developers' Alliance for Standards Harmonization of ISO 18000-7", It operates at the 433 MHz ISM band (which is available for use worldwide) and provides a multi-year battery life, a communication range of up to 2 km (potentially farther) with an E.R.P. of 10mW, low latency for tracking moving objects, small protocol stack, sensor and security support, and data transfer of up to 200 kbit/s. Networks based on DASH7 differ from typical wire-line and wireless networks that operate with a "session". DASH7 networks serves applications in which low power usage is essential, and data transmission is typically much slower and/or sporadic. Instead of replicating a wire-line "session", DASH7 was designed with the concept of BLAST:

- *B*ursty: Data transfer is abrupt and does not include content such as video, audio, or other isochronous forms of data.
- *L*ight: For most applications, packet sizes are limited to 256 bytes. Transmission of multiple, consecutive packets may occur but is generally avoided if possible.
- *As*ynchronous: DASH7's main method of communication is by command-response, which by design requires no periodic network "hand-shaking" or synchronization between devices. On the other hand, this communication method has the requirement that every device is always in listening mode, which effects on the energy efficiency.
- *T*ransitive: A DASH7 system of devices is inherently mobile or transitional. Unlike other wireless technologies DASH7 is upload-centric, not download-centric, so devices do not have to be managed extensively by fixed infrastructure (i.e. base stations).

Technical Specifications	
Range	Dynamically adjustable from 10 meters to 10 kilometers
Power	<1 mw power draw
Data Rate	Dynamically adjustable from 28kbps to 200kbps
Frequency	433.92 MHz (available worldwide)
Signal Propagation	Penetrates Walls, Concrete, Water
Real-Time Locating Precision	within 4 meters
Latency	Configurable, but worst case is less than two seconds
P2P Messaging	Yes
IPv6 Support	Yes
Security	128-bit AES, public key
Application Profiles	None
Standard	ISO/IEC 18000-7

2.8.1 Relation between frequency and distance range

The Friis equation provides a theoretical distance value for free space communication when receiver sensitivity (P_r), transmission power (P_t), receiver antenna gain (G_r), transmitter gain (G_t), and wavelength (λ) are known. The range value derived here is highly optimistic for real world scenarios, at least because it doesn't account for bandwidth or modulation.
$Range = \frac{1}{\frac{4\pi}{\lambda}\sqrt{\frac{P_r}{P_t G_t G_r}}}$

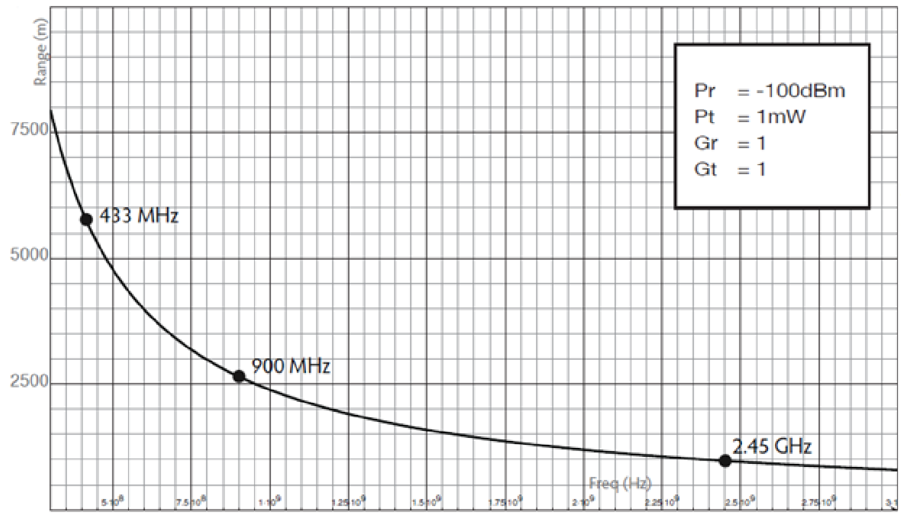


Figure 2.36: Distance range as function of the frequency

As shown in the Figure 2.36, the distance achieved by the use of a 433MHz transmission is much higher (more than 5Km theoretically) than the one achieved by other transmission techniques that uses different ISM Bands, for example Bluetooth Low Energy or ZigBee.

2.8.2 Drawbacks of the 433MHz ISM Band

The main drawback of the choice of the 433MHz ISM band is based on the fact that this frequency band is getting more and more crowded in the last years. Wireless headphones and microphones, garage door openers, wireless alarm systems, baby monitor intercoms and indoor/outdoor temperature indicators with a remote transmitter are just some of the devices that use this frequency band. They operate in the UHF band from 433.075 MHz to 434.775 MHz with 25 kHz channel spacing, for a total of 69 channels. These devices are frequency modulated (FM) with a maximum legal power output of 10 mW. The only solution for the setup of a reliable and efficient wireless sensor network using this frequency band is the use of powerful channel coding techniques. Moreover if the remote device is supposed to transmit for more than very brief periods of time the use of the 433MHz band is not possible. This band is reserved for brief periodic data transfer and the law imposes a limit on the percentage of time that the unit can be transmitting. Last drawback is that the 433MHz wavelength is approximately 69 cm, while for 2.4GHz is 10 cm, which means that the antenna design becomes to be not as simple as for higher frequency devices. The half-wave dipole is simply not an option for 433 MHz because it is 35 cm long. Even when using slightly less efficient folded-dipoles, the antenna preferred by most 2.45 GHz radios, the size is too big for most use below 1 GHz. There are quite a few designs for small antennas workable at lower frequencies anyway and some of them even come close to matching the folded dipole in performance. A common design is the small loop antenna, which is easy to design, cheap, and easy to implement within a printed-circuit board, but another solutions can be the use of an helical loop antenna, with a further performance improvements possible by inserting a ferromagnetic core into the helix, particularly with smaller loop helixes.

Chapter 3

Contiki Operative System 2.5

Contiki is an open source, multi-tasking and portable operating system for memory-constrained networked embedded systems written by Adam Dunkels at the Networked Embedded Systems group at the Swedish Institute of Computer Science. Contiki is designed for embedded systems with small amounts of memory: a typical Contiki configuration is 2 kilobytes of RAM and 40 kilobytes of ROM. Contiki, thanks to its uIPv6 stack, is IPv6 Ready Phase 1 certified, which combined with power-efficient radio mechanisms such as ContikiMAC, allows battery-operated devices to participate in IPv6 networking (also routers can run on batteries).
The new Contiki version number 2.5 supports not only the 6LoWPAN header compression, but also the new IETF RPL IPv6 routing protocol and the IETF CoAP application layer protocol, among many other protocols and mechanisms. Contiki consists of an event-driven kernel, on top of which application programs are dynamically loaded and unloaded at runtime. Contiki processes use light-weight protothreads that provide a linear, thread-like programming style on top of the event-driven kernel. Contiki is implemented in the C language, is available as open source under a BSD-style license and has been ported to a number of microcontroller architectures, including the Texas Instruments MSP430 and the Atmel AVR.
Contiki contains two communication stacks: uIP and Rime. uIP is a small RFC-compliant TCP/IP stack that makes it possible for Contiki to communicate over the Internet, while Rime is a lightweight communication stack designed for low-power radios. Rime provides a wide range of communication primitives, from best-effort local area broadcast to reliable multi-hop bulk data flooding.

3.1 System overview

A Contiki system consists of the kernel, libraries, the program loader, and a set of processes. A process could be either an application program or a service, where a service implements functionality used by more than one application process. All processes (both application programs and services) can be dynamically replaced at run-time and the communication between processes always goes through the kernel, which does not provide a hardware abstraction layer, but let device drivers and applications to communicate directly with the hardware.
An event handler function and an optional poll handler function defines a process, which state is held in the process' private memory and the kernel only keeps a pointer to the process state. All processes share the same address space and do not run in different protection domains. Interprocess communication is done by posting events.

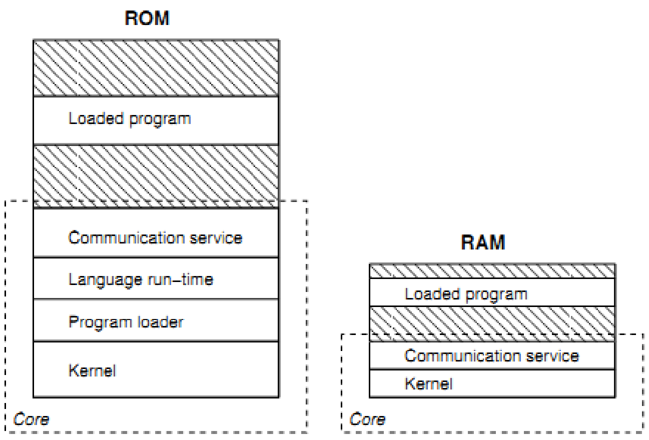


Figure 3.1: Partitioning into core and loaded

A Contiki system is partitioned into two parts: the core and the loaded programs, as shown in Figure 3.1. When the application is compiled, the partitioning is made and is specific to the deployment in which Contiki is used. The core consists typically of the Contiki kernel, the program loader, the most commonly used parts of the language runtime and support libraries, and a communication stack with device drivers forthe communication hardware. The core is compiled into a single binary image that is stored in the devices prior to deployment and is generally not modified after deployment, but it is important to note that a special boot loader can be used to overwrite or patch the core. Programs are loaded into the system by the program loader, which may obtain the program binaries either by using the communication stack or by using directly attached storage such as EEPROM. Programs to be loaded into the system are typically first stored in EEPROM before they are programmed into the code memory.

3.2 Kernel Architecture

The kernel in Contiki consists of a lightweight event scheduler that dispatches events to running processes and periodically calls processes' polling handlers. The execution of the programs is triggered either by events dispatched by the kernel or through the polling mechanism. The kernel does not preempt an event handler once it has been scheduled, therefore event handlers must run to completion and may use internal mechanisms to achieve preemption. The kernel supports two kind of events:

- asynchronous events: they are a form of deferred procedure call and they are enqueued by the kernel and dispatched to the target process some time later.
- synchronous event: they are similar to asynchronous but immediately causes the target process to be scheduled.

The control returns to the posting process only after the target has finished processing the event, which can be seen as an inter-process procedure call and similarly to the door abstraction used in the ”ring” operating system. The Contiki kernel uses a single shared stack for all process execution and the use of asynchronous events reduces stack space requirements because the stack is rewound between each invocation of event handlers.
The kernel provides a polling mechanismin addition to the events. Polling can be seen as high priority events that are scheduled in-between each asynchronous event and which is used by processes that operate near the hardware to check for status updates of hardware devices. All the processes that implement a poll handler are called, in order of their priority, when a poll is scheduled.

3.3 Events

The Contiki kernel is event-driven. The idea of such a system is that every execution of a part of the application is a reaction to an event. The entire application (kernel + libraries + user code) may contain several processes that will execute concurrently.
The different processes usually execute for some time, then wait for events

to happen. While waiting, a process is said to be blocked. When a event happens, the kernel executes the process passing the information about the event. Events can be classified in three kinds:

- timer events, where a process may set a timer to generate an event after a given time, it will block until the timer expires and then continue its execution. This is useful for periodic actions, or for networking protocols e.g. involving synchronization;
- external events, where peripheral devices connected to IO pins of the microcontroller with interrupt capabilities may generate events when triggering interruptions. A push-button, a radio chip or a shock detector accelerometer are a few examples of devices that could generate interruptions, thus events. Processes may wait for such events to react accordingly.
- internal events, where any process has the possibility to address events to any other process, or itself. This is useful for inter-process communication as informing a process that data is ready for computation.

An interrupt service routine will post an event to a process when it is executed. Events have the following information:

- process: the process addressed by the event. It can be either one specific process or all the registered processes;
- event type: the type of event. The user can define some event types for the processes to differentiate them, such as one when a packet is received, one when a packet is sent;
- data: additionally, some data may be provided along with the event for the process.

3.4 Processes

Processes are the task-equivalent of Contiki. A process is a C function, most likely containing an infinite loop and some blocking macro calls. Since the Contiki event-driven kernel is not preemptive, each process when executed will run until it blocks for an event. Several macros are defined for the different blocking possibilities, which allows programming state-machines as a sequential flow of control.
The process mechanism uses the underlying protothread library which in turn

uses the local continuation library. Protothreads are extremely lightweight stackless threads designed for severely memory constrained systems, such as small embedded systems or wireless sensor network nodes. Protothreads provide linear code execution for event-driven systems implemented in C and can be used with or without an underlying operating system to provide blocking event-handlers. They provide sequential flow of control without complex state machines or full multi-threading.

3.5 Two level scheduling hierarchy

All the event scheduling is done at a single level in Contiki and events can only be preempted by interrupts, but can't preempt each other. Interrupts are normally implemented using hardware interrupts but they can also be implemented using an underlying real-time executive, technique used by the Linux kernel to provide real-time guarantees.
Contiki never disables interrupts in order to be able to support an underlying real-time executive and, because of this, it does not allow events to be posted by interrupt handlers as that would lead to race-conditions in the event handler. On the other hand, the kernel provides a polling flag to request a poll event which provides interrupt handlers with a way to request immediate polling.

3.6 Services

A service is a process that implements functionality that can be used by other processes. A service can be seen as a form of a shared library, can be dynamically replaced at runtime and must therefore be dynamically linked. Examples of services are communication protocol stacks, higher level functionality such as sensor data handling algorithms, sensor device drivers etc.
A service consists of a service interface,which is characterized by a version number and a function table with pointers tothe functions that implement the interface, and a process that implements the interface. A service layer, conceptually situated directly next to the kernel services, keeps track of running services and provides a way to find installed services. A service is identified by a textual string that describes the service and the service layer uses ordinary string matching to querying installed services.
A stub library is used to communicate with a service by application programs that are using the service. The stub library is linked with the application and uses the service layer to find the service process. Once a service has been

located, the service stub caches the process ID of the service process anduses this ID for all future requests. Through the service interface stub, programs call services and need not be aware of the fact that a particular function is implemented as a service (Figure 3.2). The first time the service is called, the service interface stub performs a service lookup in the service layer and if the specified service exists in the system, the lookup returns a pointer to the service interface.
The version number in the service interface is checked with the version of the interface stub. The service interface contains, in addition to the version number, pointers to the implementation of all service functions which are contained in the service process. The interface stub calls the implementation of the requested function if the version number of the service stub matches the number in the service interface.

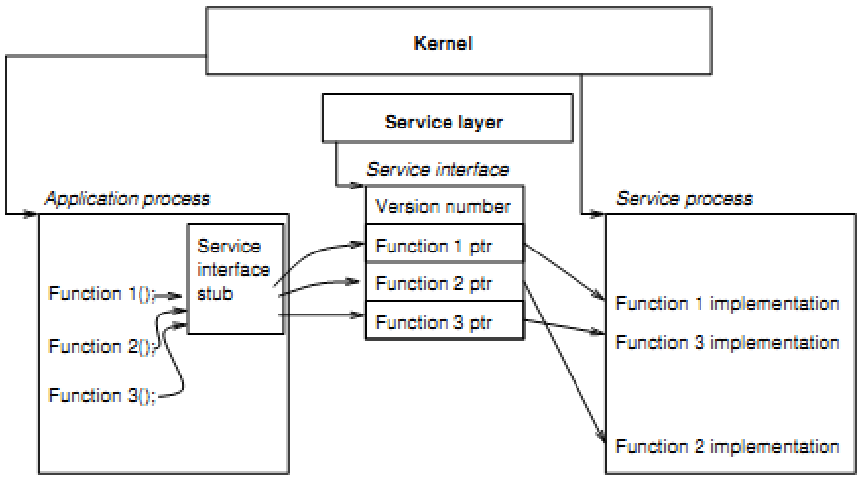


Figure 3.2: An application function calling a service

3.7 Libraries

The Contiki kernel only provides the most basic CP multiplexing and event handling features, while the rest of the system is implemented as system libraries that are optionally linked with programs. Programs can be linked with libraries in three different ways:

- Programs can be statically linked with libraries that are part of the core.

- Programs can be statically linked with libraries that are part of the loadable program.

- Programs can call services implementing a specific library. Libraries that are implemented as services can be dynamically replaced at runtime.

Runtime libraries such as often-used parts of the language runtime libraries are typically best placed in the Contiki core. Rarely used or application specific libraries are more appropriately linked with loadable programs. Libraries that are part of the core are always present in the system and don't have to be included in loadable program binaries.
For example it is possible to consider a program that uses thememcpy(), a frequently used C library function, and atoi(), a less common C library function, to copy memory and to convert strings to integers, respectively. Therefore, in this particular example, memcpy() is included in the system core but not atoi(). When the program is linked to produce a binary, the memcpy() function will be linked against its static address in the core, while the object code for the part of the C library that implements theatoi() function must, however, be included in the program binary.

3.8 Communication support

In Contiki, communication is implemented as a service in order to enable runtime replacement and to provide multiple communication stacks to be loaded simultaneously. In experimental research, this can be used to evaluate and compare different communication protocols. Moreover, the communication stack may be split into different services, as shown in Figure 3.3, which enables runtime replacement of individual parts of the communication stack.

The service mechanism is used by the communication services to call each other and synchronous events to communicate with application programs. Because synchronous event handlers are required to be run to completion, it is possible to use a single buffer for all communication processing, avoiding therefore data copy. A device driver reads an incoming packet into the communication buffer and then calls the upper layer communication service using the service mechanisms. The communication stack processes the headers of the packet and posts a synchronous event to the application program for which the packet was destined. The application program acts on the packet contents and optionally puts a reply in the buffer before it returns control to the communication stack, which prepends its headers to the out-

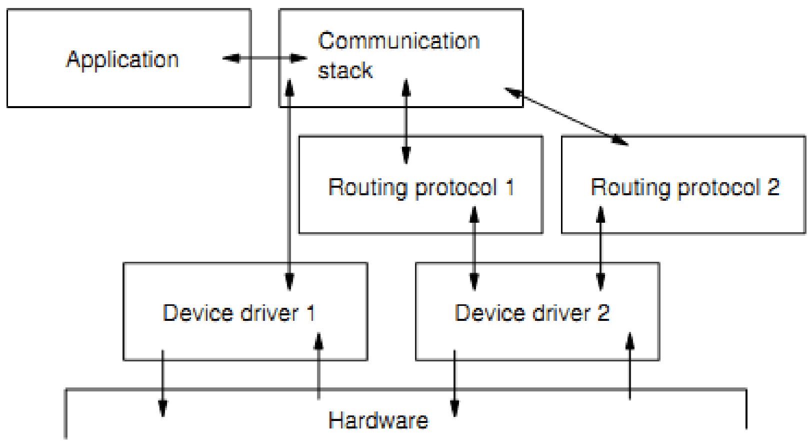


Figure 3.3: Contiki loosely coupled communication

going packet and returns control to the device driver so that the packet can be transmitted.

3.9 The uIP TCP/IP stack

The uIP TCP/IP stack is intended to make it possible to communicate using the TCP/IP protocol suite even on small 8-bit micro-controllers. Despite being small and simple, uIP do not require their peers to have complex, full-size stacks, but can communicate with peers running a similarly light-weight stack. The code size is on the order of a few kilobytes and RAM usage can be configured to be as low as a few hundred bytes. The uIP implementation is designed to have only the absolute minimal set of features needed for a full TCP/IP stack. It can only handle a single network interface and contains the IP, ICMP, UDP and TCP protocols. The uIPv6 stack is the first one developed for very constrained devices to satisfy all the IPv6 Ready Phase-1 requirements. It works independently from any particular MAC or link layer and the interface between uIPv6 and the lower layers consists of two wrappers for the link-layer input/output functions, the link-layer address, and a couple of constants. This creates a level of abstraction which enables the easy integration of many different MAC and Link Layer protocols (see Figure 3.4).

The uIPv6 stack runs as a Contiki protothread: at system startup, uIPv6 initializes its network interface and the node creates its link-local IPv6 address by combining the fe80 :: 0/64 prefix and its 802.15.4 MAC address. It

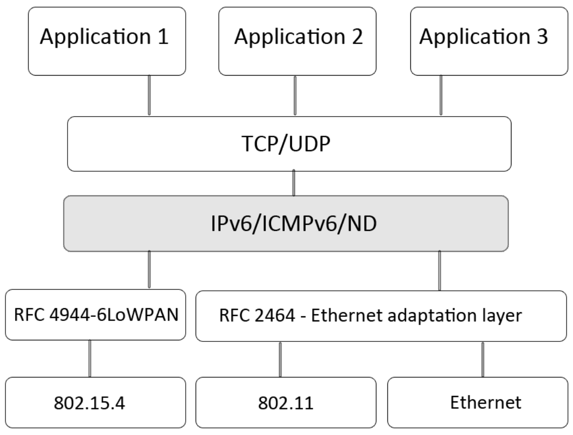


Figure 3.4: The uIPv6 stack runs over any MAC and link layer

performs then Duplicate Address Detection (DAD) to make sure the address is not already used by another node and issues router solicitation messages to trigger advertisement from routers on the network. The information received is used by the node in order to configure its global addresses and update its network parameters. The uIPv6 stack uses different timers for the messages sent during the DAD and the router discovery processes. In further operations, events generated by lower-layers trigger the processing of incoming IPv6 packets by the stack, which includes generating the appropriate response and updating the different neighbor discovery and interface data structures.
Next-hop determination is performed when a node needs to send a packet in order to find the neighbor to which the packet should be sent. If the MAC address of this neighbor is not contained in its cache, the node performs address resolution to obtain it.
A single global buffer is used by the uIPv6 stack for incoming and outgoing packets. The length of the buffer is the length of the MAC header plus 1280 bytes (the minimum link MTU), but additional buffers are available to support fragment reassembly and per neighbor packet buffering. The main data structures are the interface address list and the neighbor cache, prefix list, and default router list which are required by Neighbor Discovery. Two periodic timers are used to manage and remove outdated information from these structures. The total amount of memory usage for uIPv6 depends heavily on the applications of the particular device in which the implementations are to be run. The memory configuration determines both the amount of traffic the system should be able to handle and the maximum amount of simultaneous connections. It is possible to run the uIP implementation with as little

as 200 bytes of RAM, but such a configuration will provide extremely low throughput and will only allow a small number of simultaneous connections. The uIP stack requires a lower layer (according to the OSI model) in order to communicate with peers. Two different types of peers can be distinguished:

- Nodes: communication between nodes is achieved with a wireless link. The uIP stack needs to be able to send and receive packets. When it comes to IPv6, Contiki follows a route-over configuration. Therefore, uIP6 uses a simple MAC layer called sicslowMAC. Beside header compression provided by the 6LoWPAN module, it just forwards the packet to/from the radio. In the case of IPv4, Contiki chose a mesh-under configuration. This is done with the Rime communication stack, which provides mesh routing and route discovery and therefore uIP uses it to forward packets on the network. From the IP point of view, all the nodes of the sensor network form a local sub-network, even though multiple radio hops may be required.

- Gateways: to reach a network entity outside the wireless sensor network, a gateway is required. It's a system that will make the link between the wireless sensors network and another network. It will typically be a PC in most experiments, although it could be many embedded system. The connection between a PC and a mote is a serial link. IP packets are sent between these two using SLIP, which stands for Serial Line IP. On the computer side, a program must run to do the interface between the serial line and a network interface. Depending on the uIP stack version, the functionality is not the same.

  - With uIPv6, a node will be loaded with a very simple program that forwards every packet from the radio to the serial link and vice versa. It doesn't do any address comparison, there is no IP stack on it, besides the header compression/decompression mechanism (6LoWPAN). This node is seen from the PC point of view as an ethernet network interface.
  - With uIPv4 the node connected to the PC will act as a gateway, with all the IP stack in it. Every time it has a packet to send, it will check its IP address: if it belongs to the wireless sensor network subnet range, then it will send it using its radio, otherwise it will send it to the computer using the serial link. The computer runs a program that create a IP network interface.

3.10 RIME

The Rime communication stack provides a set of lightweight communication primitives ranging from best-effort anonymous local area broadcast to reliable network flooding.
Rime draws heavily from communication abstractions for distributed programming where layers of simple abstractions are combined to form powerful high-level abstractions. The purpose of Rime is to simplify implementation of sensor network protocols and facilitate code reuse. The code footprint of Rime is less than two kilobytes and the data memory requirements on the order of tens of bytes. Rime does not allow for a fully modular structure where any module can be replaced, but enforces a strict layering structure where only the lowest layer and the application layer can be replaced.

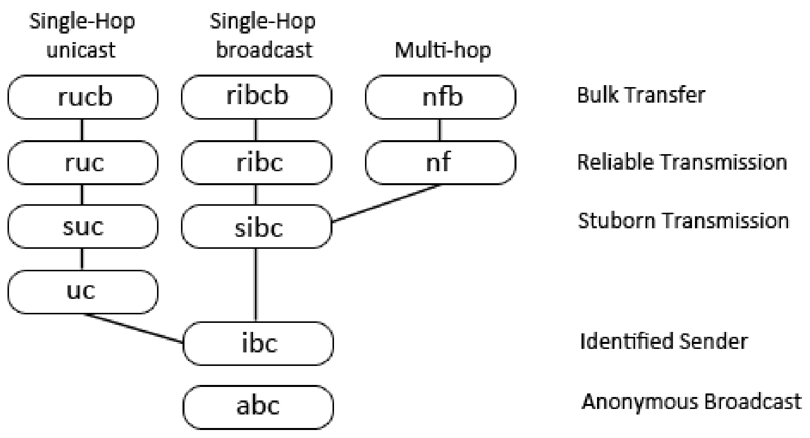


Figure 3.5: The RIME communication stack organization

Rime is organized in layers as shown in Figure 3.5. The layers are designed to be extremely simple, both in terms of interface and implementation. Each layer adds its own header to outgoing messages. The thin layers in Rime enable code reuse within the stack. For example, reliable communication is not implemented in a single layer but divided into two layers, one that implements acknowledgments and sequencing, and one that resends messages until the upper layer tells it to stop. This is called “layer stubborn”. A stubborn layer is not only used by reliable protocols but also by protocols that send periodic messages such as neighbor maintenance for routing protocols and repeated transmission of messages in Rime’s network flooding layer (NF). The lowest level primitive in Rime is anonymous best-effort broadcast, ABC. The ABC layer provides a 16-bit channel abstraction but no node addressing; it is added by upper layers. The identified sender best-effort broadcast, IBC,

adds a sender identity header field and the unicast abstraction, UC, adds a receiver header field.
One of the basic ideas of Rime is to shift the burden, in terms of memory footprint, from protocol implementations to Rime. To reduce memory footprint, Rime uses a single buffer for both incoming and outgoing packets similar to uIP. Layers that need to queue data, e.g. a stubborn protocol or a MAClayer, copy the data to dynamically allocated queue buffers.

3.11 6LoWPAN Implementation

6LoWPAN Contiki Implementation is based on RFC4944 *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*, *draft-hui-6LoWPAN-interop-00 Interoperability Test for 6LoWPAN*, and *draft-hui-6LoWPAN-hc-01 Compression format for IPv6 datagrams in 6LoWPAN Networks*.
RFC4944 defines address configuration mechanisms based on 802.15.4 16-bit and 64-bit addresses, fragmentation of IPv6 packets below IP layer, IPv6 and UDP header compression, a mesh header to enable link-layer forwarding in a mesh under topology, and a broadcast header to enable broadcast in a mesh under topology. *Draft-hui-6LoWPAN-hc-01* defines a stateful header compression mechanism which should soon deprecate the stateless header compression mechanism defined in RFC4944. It is much more powerful and flexible, in particular it allows compression of some multicast addresses and of all global unicast addresses.
6LoWPAN does not run as a separate process, but it's called by the MAC process when a 6LoWPAN packet is received, and by the TCp/IP process when an IPv6 packet needs to be sent. It is initialized from the MAC process, which calls sicsLoWPAN_init (giving as argument a pointer to the mac_driver structure), while the main 6LoWPAN functions are implemented in the sicsLoWPAN.h and sicsLoWPAN.c files which are used to format packets between the 802.15.4 and the IPv6 layers. The format of a 802.15.4 address is defined in the file uip.h, while the format of ND link-layer address options depends on the length of the link-layer addresses. 802.15.4 specificities regarding link-layer address options are implemented in uip-nd6.h. The address autoconfiguration mechanism also depends on the format of the link-layer address. The dependency is reflected in the uip_netif_addr_autoconf_set function in uip-netif.c. At initialization, the input function in sicsLoWPAN.c is set as the function to be called by the MAC upon packet reception. The output function is set as the tcpip_output function. At packet reception, the link-layer copies the 802.15.4 payload in the rime buffer, sets its length and stores the source and destination link-layer addresses as two rime addresses.

Fragmentation support is enabled by setting the SICSLOWPAN_CONF_FRAG compilation option. As complex buffer allocation mechanism is not supported, for now a new 1280 bytes buffer (sicsLoWPAN_buf) to reassemble packets is defined. At reception, once all the fragments are received, the packet is copied to uip_buf, set uip_len, and call tcpip_input.
MAC_MAX_PAYLOAD defines the maximum payload length in a 802.15.4 frame, which is constant and equal to 102 bytes (the 802.15.4 frame can be maximum 127 bytes long, and the header 25 bytes long).
The Header Compression scheme supported is defined in the SICSLOWPAN_CONF_COMPRESSION compilation option. HC1, HC01, and IPv6 compression are supported. HC1 and IPv6 compression are defined in RFC4944, HC01 in *draft-hui-6LoWPAN-hc*. IPv6 compression means sending packets with no compression, and adding the IPv6 dispatch before the IPv6 header. If at compile time either HC1 or HC01 are chosen, all fields at sending will be compressed, and packets compressed with the chosen scheme will be accept as well as uncompressed packets. Note that HC1 and HC01 support is mutually exclusive. HC01 should soon deprecate HC1.
When a packet is received, the input function is called. Fragmentation issues are handled, then the dispatch byte is checked: if it is IPv6, the packet is treated inline. If it is HC1 or HC01, the corresponding decompression function (uncompress_hdr_hc1 or uncompress_hdr_hc01) is called.

3.12 Preemptive multi-threading

Preemptive multi-threading is implemented in Contiki as a library on top of the event-based kernel. The library is optionally linked with applications that explicitly require a multi-threaded model of operation and is divided into two parts: a platform independent part that interfaces to the event kernel and a platform specific part implementing the stack switching and preemption primitives. Usually, the preemption is implemented using a timer interrupt that saves the processor registers onto the stack and switches back to the kernel stack. In practice very little code (25 lines of C code in the implementation for the MSP430) needs to be rewritten when porting the platform specific part of the library.
Unlike normal Contiki processes, each thread requires a separate stack and the library provides the necessary stack management functions. Threads execute on their own stack until they either explicitly yield or are preempted. The API of the multi-threading library consists of four functions that can be called from arunning thread (mt yield(), mt post(), mt wait(),and mt exit()) and two functions called to setup and run a thread (mt start() and mt

exec()). The mt exec() function performs the actual scheduling of a thread and is called from an event handler.

3.13 Code size

One of the most important characteristics of operating system for constrained devices is the compactness in terms of both code size and RAM usage in order to leave room for applications running on top of the system.

Module	Code size (AVR)	Code size (MSP430)	RAM usage
Kernel	1044	810	10+4e+2p
Service layer	128	110	0
Program loader	-	658	8
Multi-threading	678	582	8+s
Timer library	90	60	0
Replicator stub	182	98	4
Replicator	1752	1558	200
Total	3874	3876	230 + 4e + 2p + s

In the Table above the compiled code size (in bytes) and the RAM usage of the Contiki 2.1 system compiled for two architectures (the Texas Instruments MSP430 and the Atmel AVR) is shown. The numbers report the size of both core components and an example application: a sensor data replicator service which consists of the service interface stub for the service as well as the implementation of the service itself. The program loader is only implemented on the MSP430 platform.
The RAM requirement depends on the maximum number of processes (p) that the system is configured to have,the maximum size of the asynchronous event queue (e) and, in the case of multi-threaded operation, the size of the thread stacks (s).

3.14 Contiki RPL

ContikiRPL was implemented in the version 2.5 of Contiki by Joakim Eriksson and Nicolas Tsiftes from the Contiki team. It was written in C programming language, using the APIs of the Contiki Operating System.

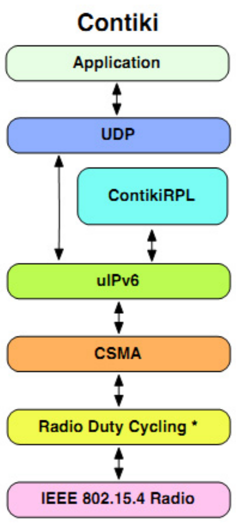


Figure 3.6: Contiki Stack with ContikiRPL

A network that manages its routing topologies using RPL may run one or more RPL instances, where each instance defines a topology that is built using a unique metric or constraint within the network. In each RPL instance, multiple Directed Acyclic Graphs (DAGs) may exist, each having a different DAGroot. What is important to note is that a node can join multiple RPL instances, but must belong only to one DAG within each instance. When forming the topology, each sink constructs a packet called a DAG Information Object (DIO), and sends it to all children: any child that decides to join the DAG may pass the DIO further to its own children. The DIO contains a rank, a number increased monotonically when a child joins the DAG, used to prevent routing loops and help the nodes to distinguish between parents and siblings.
DAGs may need to change if the network restructures because of mobility or link quality variance: RPL ensures that DAGs are adjusted occasionally by having the root send out a new DAG iteration. A Trickle timer is used in order to regulate when nodes should forward such information to their children, which efficiently suppresses many redundant updates in dense networks. It is important to note that, in order to generate a fast repair, nodes that detect routing inconsistencies (for example the loss of a parent) reset their Trickle timers to their minimum values.
The routing protocol uses Contiki's modular IPv6 routing interface, which has three functions: activate, deactivate and lookup. The activate function initializes the Directed Acyclic Graph(DAG) construction by sending a DAG Information Solicitation (DIS); neighbors that belong to a DAG will

reset their Trickle timers, and shortly thereafter the node will receive at least one DIO. The deactivate function deallocates internal structures and sends a no-DAO to the node's neighbors. After deactivation, the module stops responding to route-lookup requests, but may be reactivated later. If uIPv6 detects that the destination for a packet is not an immediate neighbor, it asks RPL for the route using the lookup function.

ContikiRPL implements the RPL protocol and two objective functions: OF0 and the Minimum Rank Objective Function with Hysteresis(MRHOF). It separates protocol logic, message construction and parsing, and objective functions into different modules. The protocol logic module manages DODAG information, maintains a set of candidate parents and their associated information, communicates with objective function modules, and validates RPL messages at a logical level according to the RPL specification. The message construction and parsing module translates between RPL's ICMPv6 message format and Contiki RPL's own abstract data structures. Finally, the objective function modules implement an objective function API. To facilitate simple replacement of and experimentation with objective functions, their internal operation is opaque to ContikiRPL.
ContikiRPL sets up forwarding tables for uIPv6 and leaves the actual packet forwarding to uIPv6 instead of making the forwarding decisions per packet. Outgoing IPv6 packets flow from the uIPv6layer to the 6LoWPAN layer for header compression, fragmentation. The outgoing packet is then sent by the 6LoWPAN layer to the MAClayer. The default Contiki MAC layer is a CSMA/CA mechanism that places outgoing packets on a queue, which are then transmitted in order through the radio duty cycling (RDC) layer. The RDC layer in turn transmits the packets through the radio link layer. The MAC layer will retransmit packets until it sees a link-layer acknowledgment from the receiver, but outgoing packets have a configurable threshold for the maximum number of transmissions.
Link cost estimation updates are provided by an external neighbor information module through a callback function. ContikiRPL recomputes the path cost to the sink via the updated link, and checks with the selected objective function whether to switch the preferred parent. The link cost reflects the per-neighbor ETX metric, which is calculated using an exponentially-weighted moving average function over the number of link-layer transmissions with $\alpha = 0.2$. The ETX is used to compute the rank for the MRHOF objective function and in parent selection for the OF0 objective function. New neighbor table entries have an initial ETX estimate of 5. The ContikiRPL neighbor eviction policy is to keep neighbors that have good ETX estimates and low ranks.

The table below shows the implementation complexity for ContikiRPL on a Tmote SKY module. The total ROM size is 3224 bytes, which is approximately 6.5% of the Tmote Sky's ROM. The RAM size is dependent on the amount of candidate neighbors that should be stored; ContikiRPL reserves space for 10 neighbors. This setting results in a RAM footprint of 800 bytes, which is approximately 8% of the available space.

Module	RAM (bytes)	ROM (bytes)
Generic IPv6 routing	420	484
RPL packet generation and parsing	2	1316
RPL protocol logic	378	1074
RPL timer handling	0	350
ContikiRPL Total	800	3224

Chapter 4

RPL Routing Protocol

The Routing Protocol for Low Power Lossy Networks (RPL) provides a mechanism whereby point-to-point, multipoint-to-point and point-to-multipoint traffic from the central control point to the devices inside the LoWPAN is supported. This is possible through a mechanism which disseminate information over the dynamically-formed network topology, which enables minimal configuration in the nodes, allowing them to operate mostly autonomously. RPL may in particular disseminate IPv6 Network Discovery information such as Prefix Information Option (PIO) and Route Information Option (RIO). Neighbor discovery information that is disseminated by RPL conserves all its original semantics for router to host, with limited extensions for router to router. RPL also introduces the capability to bind a subnet together with common prefix and to route within that subnet.
In order to be useful in a wide range of LoWPAN application domains, RPL separates packet processing and forwarding from the routing optimization objective. RPL operations require bidirectional links, while in some LoWPAN scenarios those links may exhibit asymmetric proprieties. It is required that the reachability of a router is verified before the router can be used as a parent.

4.1 Protocol Overview

RPL organizes a topology as a Directed Acyclic Graph (DAG) that is partitioned into one or more Destination Oriented DAGs (DODAGs). RPL routes are optimized for traffic to or from one or more roots that act as sinks for the topology, which means there is one DODAG per sink. RPL uses four values to identify and maintain a topology:

- RPLInstanceID: it identifies a set of one or more Destination Oriented DAGs; the set of DODAGs intentified by a RPLInstanceID is called a RPL Instance. All the DODAGs in the same RPL Instance use the same Objective Function (OF), which defines how routing metrics, optimization objectives and related functions are used to compute a Rank (a number that identifies the node's individual position relative to other nodes with respect to a DODAG root) and which dictates how parents in the DODAG are selected

- DODAGID: the scope of a DODAGID is a RPL Instance and the combination of RPLInstanceID and DODAGID uniquely identifies a single DODAG in the network. It is important to note that a RPL Instance may have multiple DODAGs, each of which has an unique DODAGID.

- DODAGVersionNumber: the scope of a DODAGVersionNumber is a DODAG and the combination with a RPLInstanceID and a DODAGID uniquely identifies a DODAG Version

- Rank: the scope of Rank is a DODAG Version. It establishes a partial order over a DODAG version and defines an individual node position with respect to the DODAG root.

A RPL Instance contains one or more DODAG roots and may provide routes to certain destination prefixes, reachable via the DODAG roots or alternate paths within the DODAG. It may comprise a single DODAG with a single root, multiple uncoordinated DODAGs with independent roots, a single DODAG with a virtual root that coordinates the LoWPAN sinks over a backbone network or a combination of the above.

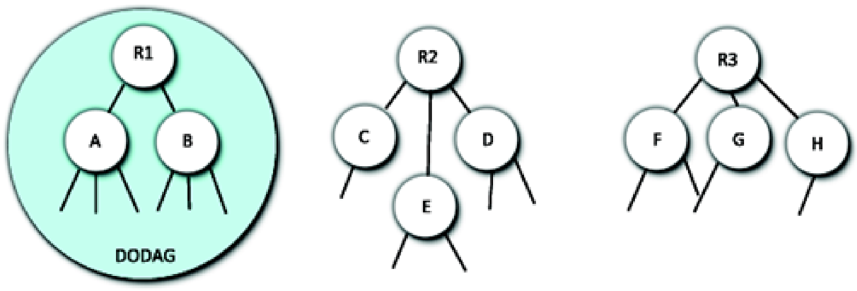


Figure 4.1: RPL Instance

The Figure 4.1 shows an example of an RPL Instance comprising three DODAGs. The DODAG roots are R1, R2 and R3 and each of these advertise the same RPLInstanceID.

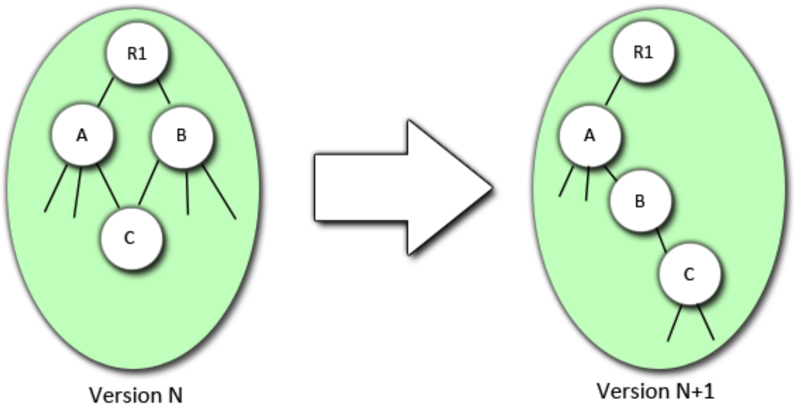


Figure 4.2: RPL DODAG Version number increment

The Figure 4.2 shows how a DODAG Version number increment leads to a new DODAG Version and, in this case, a different DODAG topology.
The Objective Function defines how RPL nodes select and optimize routes within a RPL Instance. The Objective Function is identified by an Objective Code Point within the DIO configuration option and defines how nodes translate one or more metrics and constraints into a value, the Rank, which approximates the node's distance from a DODAG root. Another role of the Objective Function is to define how the nodes select their parents.
DODAGs can be grounded or floating, where the grounded offers connectivity to hosts that are required for satisfying the application-defined goal and a floating only provides routes to nodes within the DODAG and may be used for example to preserve inner connectivity during repair. The root has the task of advertising which is the case. An high level overview of the distributed algorithm which constructs the DODAG is as follows:

- Some nodes are configured to be DODAG roots, with associated DODAG configurations
- Nodes advertise their presence, affiliation with a DODAG, routing cost and related metrics by sending link-local multicast DODAG information Object (DIO) messages to all RPL nodes
- Nodes listen for DIOs and use their information to join a new DODAG

or to maintain an existing DODAG according to the specified Objective Function and Rank of their neighbors

- Nodes provision routing table entries for the destinations specified in the DIO message via their DODAG parents in the DODAG Version. Nodes that decide to join a DODAG can provision one or more DODAG parents as the next-hop for the default route and a number of other external routes for the associated instance.

For applications that require P2MP or P2P traffic, RPL uses Destination Advertisement Object (DAO) messages to establish downward routes. RPL supports two modes of downward traffic: storing or non-storing. In both cases, P2P packets travel Up toward a DODAG Root then Down to the final destination, but in the non-storing case the packet will travel all the way to the root before travelling down, while in the storing case the packet may be directed down towards the destination by a common ancestor of the source and the destination prior to reaching a DODAG root. The rank of a node is a scalar representation of the location of that node within a DODAG Version and is used to avoid and detect loops. The exact calculation of the rank is left to the Objective Function and it may depend on parents, link metrics, node metrics and node configuration and policies. It is incremented in a strictly monotonic fashion and can be used to validate a progression from or towards the root. The Rank can be thought as a fixed point number, where the position of the radix point between the integer part of the fractional part is determined by MinHopRankIncrease, the minimum increase between a node and any of its DODAG parents. MinHopRankIncrease creates a tradeoff between hop cost precision and the maximum number of hops a network can support. When an objective function computes a rank, the objective function operates on the entire rank quantity and when the rank is compared the integer portion of the rank is to be used: DAGRank(rank) = floor (rank/MinHopRankIncrease).
Given two nodes A and B, if DAGRank(A) is less than DAGRank(B) then the position of A is closer to the DODAG Root than the position of B. Node A may safely be a DODAG parent for Node B without risk of creating a loop. If DAGRank(A) equals DAGRank(B) the position of A and B with respect to the DODAG root are identical and routing through a node with equal Rank may cause a routing loop. RPL tries to avoid creating loops when undergoing topology changes and includes rank-based datapath validation mechanism for detecting loops when they occur. RPL guarantees neither loop free path selection nor tight delay convergence times, but can detect and repair a loop as soon as it is used. RPL uses this loop detection to ensure that packets make forward progress within the DODAG Version

and trigger repairs when necessary.
A DODAG loop may occur when a node detaches from the DODAG and reattaches to a device in its prior sub-DODAG, which may happen in particular when DIO messages are missed. Strict use of the DODAG Version Number can avoid this kind of loop. A DAO loop can occur when the parent has a route installed upon receiving and processing a DAO Message from a child, but the child has subsequently cleaned up the related DAO state. This kind of loop happens when a NO-Path message was missed (a kind of DAO message that invalidates a previously announced prefix). RPL includes an optional mechanism to acknowledge DAO messages, which can reduce the impact of a single DAO message being missed.

4.2 RPL Instances

There are two types of RPL Instances, local and global, and the RPLInstanceID space is divided between Global and Local instances to allow for both coordinated and unilateral allocation of RPLInstanceIDs. Global RPL Instances are coordinated, have one or more DODAGs and are typically long-lived, while local RPL Instances are always a single DODAG whose singular root owns the corresponding DODAGID, allocates the Local RPLInstanceID in a unilateral manner and can be used, for example, for constructing DODAGs in support of a future on-demand routing solution.
Data packets and control packets within RPL network are tagged to unambiguously identify what RPL Instance they are part of: every RPL control message has a RPLInstanceID field and the data packets that flow within the network expose the RPLInstanceID as part of the RPL Packet Information that RPL requires.
A global RPLInstanceID must be unique in the whole LoWPAN and there can be up to 128 global instance in the whole network. A global RPLInstanceID is encoded in a RPLInstanceID field as follows: Local instances are always used in conjunction with a DODAGID, are allocated and managed by the node that owns the DODAGID and up to 64 local instances can be supported.

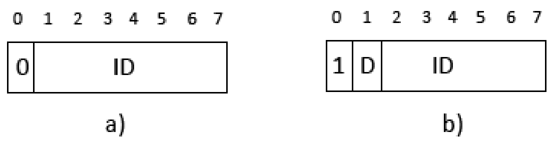


Figure 4.3: RPL field format for a) Global Instance ID b) Local Instance ID

In Figure 4.3 the D flag is set to 0 in RPL control messages, while in data packets it indicates whether the DODAGID is the source or the destination of the packet.

4.3 ICMPv6 RPL Control messages

A RPL control message is identified by a code and composed of a base that depends on the code, and some options. The source address is a link-local address and the destination address is either the all-RPL-nodes multicast address (FF02::1A) or a link-local unicast address of the destination. The RPL control message consists of an ICMPv6 header followed by a message body which is comprised of a message base and some options.

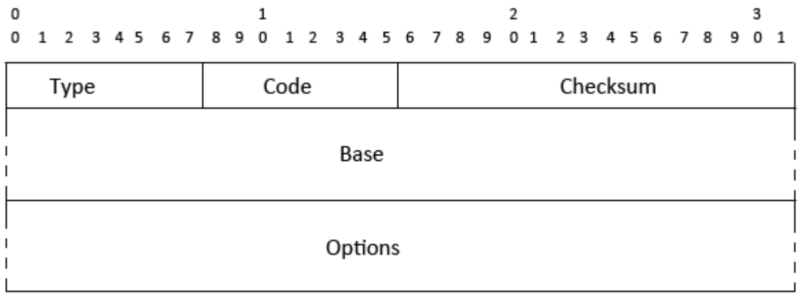


Figure 4.4: RPL Control Message

The code field identifies the type of RPL Control message:

0x00	DODAG Information Solicitation (DIS)
0x01	DODAG Information Object (DIO)
0x02	Destination Advertisement Object (DAO)
0x03	Destination Advertisement Object Acknowledgment (DAO-ACK)
0x80	Secure DODAG Information Solicitation
0x81	Secure DODAG Information Object
0x82	Secure Destination Advertisement Object
0x83	Secure Destination Advertisement Object Acknowledgment
0x8A	Consistency Check

Each RPL message has a secure variant which provides integrity, confidentiality and replay protection. Because the security covers the base message and the options, the security information are placed between the checksum and the base (Figure 4.5).

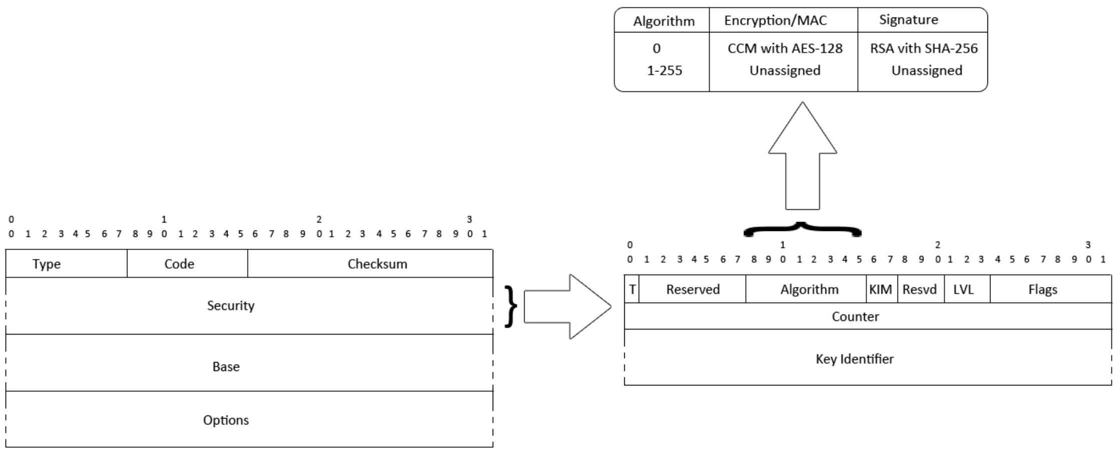


Figure 4.5: Secure RPL control message and security algorithm

The security algorithm field specifies the encryption, Message Authentication Code (MAC) and signature scheme the network uses. The DODAG Information Solicitation (DIS) message can be used to solicit a DODAG Information Object from an RPL node, similarly to that of a Router Solicitation specified in IPv6 Neighbor Discovery. A node may use DIS to probe its neighborhood for nearby DODAGs.
The DODAG Information Object (DIO) carries information that allows a node to discover a RPL Instance, learn its configuration parameters, select a DODAG parent set and maintain the DODAG (Figure 4.6).

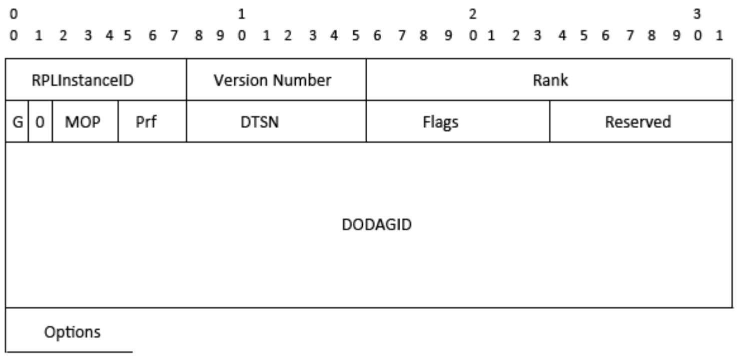


Figure 4.6: The DIO base object

MOP	Meaning
0	No downward routes maintained by RPL
1	No storing mode
2	Storing without multicast support
3	Storing with multicast support
-	All other values are unassigned

In the DIO the DODAGID is a 128-bit IPv6 address set by the DODAG root which uniquely identifies a DODAG and must be a routable IPv6 address belonging to the DODAG root. Destination Advertisement Object (DAO) is used to propagate destination information upwards along the DODAG (Figure 4.7). In storing mode the DAO message is unicast by the child to the selected parent, while in the non-storing mode the DAO message is unicast to the DODAG root. In both modes the DAO message can optionally be acknowledged by its destination with a DAO-ACK.

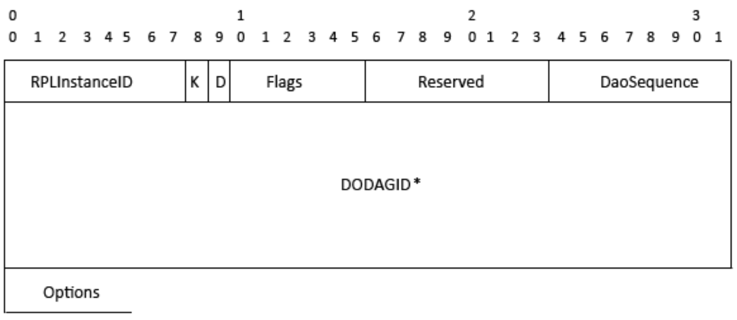


Figure 4.7: The DAO base object

It is important to note that the '*' denotes that the DODAGID is not always present: if the 'D' flag is set then a local RPLInstanceID is used and the DODAGID field is present. The 'K' flag indicates if a DAO-ACK is expected; the DAOSequence is incremented at each unique DAO message from a node and echoed in the DAO-ACK message.

The consistency check message (CC) is used to check secure message counters and issue challenge/response and it must be sent as a secured RPL message (Figure 4.8).

The CC nonce is a 16-bit unsigned integer set by a CC request and the corresponding CC response includes the same CC nonce value as the request. The Destination Counter is a 32-bit unsigned integer value indicating the

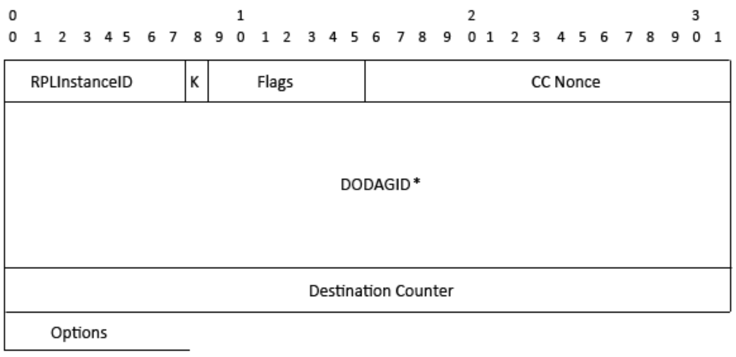


Figure 4.8: Consistency Check Nonce

sender's estimate of the destination's current security Counter value and allows new or recovered nodes to resynchronize through CC message exchanges.

The RPL control message options use the format shown in Figure 4.9:

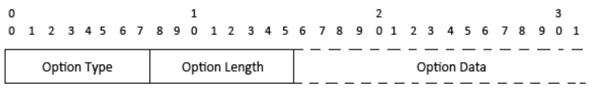


Figure 4.9: RPL control message options format

Pad1 option may be present in DIS, DIO, DAO and CC messages and is used to insert a single octet of padding into the message to enable options alignment.
PadN option may be present in the same messages as in Pad1, but is used to insert two or more octets of padding into the message to enable options alignment.
Metric container option may be present in DIO or DAO messages and is used to report metrics along the DODAG.
Route Information option may be present in DIO messages and carries the same information as the IPv6 Neighbor Discovery Route Information option. It is used to indicate that connectivity to the specified destination prefix is available from the DODAG root.
DODAG Configuration option may be present in DIO messages and is used to distribute configuration information for DODAG operation through the DODAG.
RPL Target option may be present in DAO messages and is used to indicate a target IPv6 address prefix or multicast group that is reachable or queried along the DODAG. In a DAO the RPL target option indicates reachability.

Transit Information option may be present in DAO messages and is used for a node to indicate attributes for a path to one or more destinations, where the destinations are indicated by one or more Target options that immediately precede the Transit Information option. It can be used for a node to indicate its DODAG parents to an ancestor that is collecting DODAG routing information, typically for the purpose of constructing source routes.
Solicit Information option may be present in DIS messages and is used for a node to request DIO messages from a subset of neighboring nodes. It contains flags that indicate which predicates a node should check when deciding whether to reset its Trickle timer (a node resets it when all predicates are true). There are three possible predicates: V (Version predicate), which is true if the receiver's DODAGVersionNumber matched the requested version number; I (InstanceID predicate), which is true when the RPL node's current RPLInstanceID matches the requested RPLInstanceID; D (DODAGID predicate), which is true if the RPL node's parent set has the same DODAGID.
Prefix Information option may be present in DIO messages and is used to distribute the prefix in use inside the DODAG. An RPL node may use this option for the purpose of Stateless Address Auto Configuration from a prefix advertised by a parent.
RPL Target Descriptor option may be immediately followed by one opaque descriptor that qualifies that specific target.

4.4 Sequence counter

In order to validate the freshness and the synchronization of protocol information, three different sequence numbers are used:

- DODAGVersionNumber is present in the DIO base to indicate the version of the DODAG being formed and is monotonically incremented by the root each time the root decides to form a new version of the DODAG in order to revalidate the integrity and allow a global repairs to occur. This number is propagated unchanged down the DODAG as routers join the new DODAG version. An older value (lesser) indicates that the originating router has not migrated to the new DODAG version and can't be used as a parent once the receiving node has migrated to the newer DODAG version.

- DAOSequence is present in the DAO base to correlate a DAO message and a DAO ACK message. This number is locally significant to the node that issues a DAO message for its own consumption to detect the loss of a DAO message and enable retries

- Path Sequence is present in the Transit Information option in a DAO message and is used to differentiate a movement where a newer route supersedes a stale one from a route redundancy scenario where multiple routes exist in parallel for a same target. This number is globally significant in a DODAG and indicates the freshness of the route to the associated target. An older value (lesser) received from an originating router indicates that the originating router holds stale routing states and should not be considered anymore as a potential next-hop for the target. This number is computed by the node that advertises the target and is unchanged as the DAO content is propagated towards the root by parent routers.

4.5 Upward routes

Nodes that decide to join a DODAG must provision at least one DODAG parent as a default route for the associated instance, which enables a packet to be forwarded upwards until it eventually hits a common ancestor from which it will be routed downwards to the destination.
A DIO message can also transport explicit routing information, like the DODAGID, a global or unique local IPv6 address of the root, or a RIO Prefix, used to advertise an external route that is reachable via the root and is interpreted as a capability of the root as opposed to a routing advertisement and must not be redistributed in another routing protocol. It should be used by an ingress RPL router to select a DODAG when a packet is injected in a RPL domain from a node attached to that RPL router.
A node that is not a DODAG root has to advertise the same values as its preferred DODAG parent for the following base fields: Grounded (G), Mode of operation (MOP), DAGPreference (Prf), Version, RPLInstanceID and DODAGID. In this way these values will propagate down the DODAG unchanged and advertise by every node that has a route to that DODAG root. A node may, on the other hand, update the Rank field and the DTSN field at each hop. Moreover, the DODAGID field must be unique within the RPL Instance and must be a routable IPv6 address belonging to the root.
Upward route discovery allows a node to join a DODAG by discovering neighbors that are members of the DODAG of interest and identifying a set of parents. RPL's upward route discovery algorithms and processing are in terms or three logical sets of link-local nodes: first the candidate neighbor set is a subset of the nodes that can be reached via link-local multicast; second the parent set is a restricted subset of the candidate neighbor set; third the preferred parent is a member of the parent set that is the preferred next hop in

upward routes. In this way there is a consistent partial order on nodes within the DODAG and as long as node's rank don't change every node's route to a DODAG root is loop-free. The above rules govern a single DODAG version, while the following define how RPL operates when there are multiple DODAG versions:

1. The tuple (RPLInstanceID, DODAGID, DODAGVersionNumber) uniquely defines a DODAG Version and every element of a node's DODAG parent set must belong to the same DODAG Version.

2. A node is a member of a DODAG version if every element of its DODAG parent set belongs to that DODAG version or if that node is the root of the corresponding DODAG.

3. A node must not send DIOs for DODAG versions of which is not a member

4. DODAG roots may increment the DODAGVersionNumber that they advertise and thus move to a new DODAG Version.

5. Within a given DODAG a node that is not a root must not advertise a DODAGVersionNumber higher than the highest DODAGVersionNumber he has heard

6. Once a node has advertised a DODAGVersion by sending a DIO, it must not be a member of a previous DODAGVersion of the same DODAG.

The objective function and the set of advertised routing metrics and constraints of a DAG determines how a node selects its neighbor set, parent set and preferred parents. This selection implicitly also determines the DODAG within a DAG and can include administrative preference (Prf) as well as metrics.
Regarding the Rank and the movement within a DODAG version, a node must not advertise a Rank less than or equal to any member of its parent set within the DODAG version and may advertise a Rank lower than its prior advertisement within the DODAG version. Let L be the lowest rank within a DODAG version that a given node as advertise: that node must not advertise an effective rank higher than L+DAGMaxRankIncrease. Moreover, a node may choose to join a different DODAG within a RPL Instance and such a join has no rank restriction unless that different DODAG is a DODAG version of which this node has previously been a member and may choose to migrate to the next DODAG Version within the DODAG at any time after

hearing the next DODAGVersionNumber advertised from suitable DODAG Parent. In this case the DODAG parent set needs to be rebuild for the new version.
When a DIO message is received, the receiving node must first determine whether or not the DIO message should be accepted for further processing and subsequently present the DIO message for further processing if eligible. If the DIO message is malformed it must be discarded by the node, but if the sender of the DIO message is a member of the candidate neighbor set and the DIO message is not malformed, the node must process the DIO.
The DODAG selection is implementation and Objective Function dependent: nodes should provide a mean to filter out a parent whose availability is detected as fluctuating and should verify that bidirectional connectivity and adeguate link quality is available with a candidate neighbor before it considers that candidate as a DODAG parent.
In some cases a RPL node may attach to a DODAG as a leaf node only, which does not extend DODAG connectivity but may still need to transmit DIOs on occasion. A node operating as a leaf node must not transmit DIOs containing the DAG metric container and must advertise a DAGRank of INFINITE_RANK. It may suppress DIO transmission unless it has been triggered due to detection of inconsistency when a packet is being forwarded or in response to a unicast DIS message and it may transmit unicast DAOs and multicast DAOs to the 1-hop neighborhood.

4.6 Downward routes

RPL constructs and maintains downward routes with Destination Advertisement Object (DAO) messages. Downward routes support P2MP flows from the DODAG roots toward the leaves and P2P flows toward a DODAG Root through an upward route, then away from the DODAG root to a destination through a downward route.
A RPL Instance may choose between two different modes for maintaining downward routes: storing and non-storing modes. In the first one, nodes store downward routing tables for their sub-DODAG and each hop on a downward route examinates its routing table to decide on the next hop, while in the second one (non-storing), nodes do not store downward routing tables and downward packets are routed with source routes populated by a DODAG root.
To establish downward routes, RPL nodes send DAO messages upwards and the next hop destination of these DAO messages are called DAO parents. The collection of a node's DAO parents is called DAO Parent Set. A node

may send DAO messages using the all-RPL-nodes multicast address (optimization to provision one-hop routing). In storing-mode operation a node must not address unicast DAO messages to nodes that are not DAO parents and the IPv6 source and destination addresses of a DAO message must be link-local address. In non-storing-mode a node must not address unicast DAO messages to nodes that are not DODAG roots and the IPv6 source and destination addresses of a DAO message must be a unique-local or a global addresses. The selection of DAO parents is implementation and objective function specific.
Destination Advertisement can be configured to be entirely disabled, as reported in the Mode of Operation (MOP) in the DIO message. All nodes who join a DODAG have to abide by the MOP setting from the root and, if the MOP is 0 (no downward routing), nodes must not transmit DAO messages. In non-storing mode the DODAG root should store source routing table entries for destination learned from DAOs and must be able to generate source routes for those destinations learned from DAOs which were stored. On the other hand in storing mode all non-root and non-leaf nodes must store routing table entries for destinations learned from DAOs.
For each target associated with a node, that node is responsible to transmit DAO message in order to provision downward routes. The node must increment the Path Sequence counter and generate a new DAO message when the Path Lifetime has to be updated or the Parent Address list has to be changed. The Target+Transit information contained in the DAO messages subsequently propagates up the DODAG. In storing mode the node generates such DAO to each of its DAO parents in order to enable multipath.
A node might send DAO messages when it receives DAO messages, as a result of changes in its DAO parent set, or in response to another event such as the expiry of a related prefix lifetime. In non-storing mode every DAO message a node receives is "new", while in storing mode a DAO message is "new" if it has a newer Path Sequence number or if it has additional Path Control bits, or if it is a No-Path message that removes the last downward route to a prefix.
Because DAOs flow upwards, receiving a unicast DAO can trigger sending a unicast DAO to a DAO parent. On receiving a unicast DAO message with updated information, a node should send a DAO after a specific delay (DelayDAO) in order to aggregate DAO information from other nodes for which it is a DAO parent. Nodes can trigger their sub-DODAG to send DAO messages. Each node maintains a DAO trigger sequence number (DTSN) and if a node hears that one of its DAO parents increment its DTSN, it should schedule a DAO message transmission. In case of non-storing mode the node has also to increment its own DTSN. In the case of triggered DAOs, select-

ing a proper DelayDAO can greatly reduce the number of DAOs transmitted. Such a scheduling could be approximated by setting DelayDAO infersely proportional to the Rank.
In non-storing mode, RPL routes messages downward using IP source routing. DAOs are sent directly to the root along a default route installed as part of the parent selection. A node uses DAOs to report its DAO parents to the DODAG root, who can piece together a downward route to a node by using a DAO parent set from each node in the route. The purpose of this per-hop calculation is to minimize traffic when DAO parents change.
In storing mode, RPL routes messages downward by the IPv6 destination address. On receiving a unicast DAO, a node must compute if the DAO would change the set of prefixes that the node itself advertises. If so, the node must generate a new DAO message and transmit it. When a node generates a new DAO, it should unicast it to each of its DAO parents and it must not unicast the DAO message to nodes that are not DAO parents. When a node removes a node from its DAO parent set, it should send a No-Path message to invalidate the existing route. Unlike in non-storing mode, these DAOs do not communicate information about the routes themselves: that information is stored within the network and is implicit from the IPv6 source address. Because this information is stored within each node's routing tables, in storing mode DAOs are communicated directly to DAO parents, who store this information.
A DAO message from a node contains one or more Target Options, each of which specifies either a prefix advertised by the node, a prefix of addresses reachable outside the LoWPAN, the address of destination in the node's sub-DODAG or a multicast group that a node in the sub-DODAG is listening to. The Path Control field of the Transit Information option allows a node to bound how many downward routes will be generated to it. In non-storing mode the root can determine the downward route by aggregating the information from each received DAO which includes the Path Controll indications of preferred DAO parents.
A special case of DAO operation, distinct from unicast DAO operation, is multicast DAO operation which may be used to populate "1-hop" routing table entries. A multicast DAO message must be used only to advertise information about the node itself and not to relay connectivity information learned. It can be used to enable direct P2P communication without needing the DODAG to relay the packets.

4.7 RPL Security

RPL supports three security modes: unsecured, pre-installed and authenticated.
In the unsecured one, RPL uses basic DIS, DIO, DAO and DAO-ACK messages which do not have security sections. It is important to note that, as network could be using other security mechanism such as link-layer security, unsecured mode does not imply all messages are sent without any protection.
In the Pre-installed mode, RPL uses secure messages and, in order to join a RPL instance, a node must have a pre-installed key. Nodes use this to provide message confidentiality, integrity and authenticity. A node may, using this preinstalled key, join the RPL network as either a host or a router.
The Authenticated mode uses, as in the pre-installed mode, secure messages and a pre-installed key. The main difference is that a node may join the network as a host only and, in order to join the network as a router, a node must obtain a second key from a key authority, who can authenticate that the requester is allowed to be a router before providing it with the second key.
As the cryptographic basis for RPL security, CCM - Counter with CBC-MAC (Cipher Block Chaining Message Authentication Code) is used and CCM uses AES-128 as its underlying cryptographic algorithm. To join a secure RPL network, a node either listens for secure DIOs or triggers secure DIOs by sending a secure DIS.
RPL nodes send Consistency Check (CC) messages to protect against replay attacks and synchronize counters. Consistency Check messages allow nodes to issue a challenge-response to validate a node's current Counter value. Because the CC nonce is generated by the challenger, an adversary replaying messages is unlikely to be able to generate a correct response.
When secured RPL messages have to be transmitted, a RPL node must set the security section (T, Sec, KIM, LVL) in the outgoing RPL packet to describe the protection level and security settings that are applied. The counter value used in constructing the AES-128 CCM nonce to secure the outgoing packet must be an increment of the last counter transmitted to the particular destination address. When a secured RPL message is received, the node uses the RPL security control fields to determine the necessary packet security processing.
Message authentication codes (MAC) and signatures are calculated over the entire unsecured IPv6 packet. MAC and signature calculation are performed before any compression that lower layers may apply. The signature scheme in RPL for security mode is an instantiation of the RSA algorithm (RSASSA-PSS) which uses a public key pair (n, e), where n is a 2048-bit or 3072-bit

RSA modulus and where $e = 2^{16} + 1$. It uses CCM mode as the encryption scheme with M=0 (as a stream-cipher). RSA signatures of this form provide sufficient protection for RPL networks.

4.8 Packet forwarding and loop avoidance

When forwarding a packet to a destination, precedence is given to selection of a next-hop successor in the following way:

- If the packet header specifies a source route by including a RH4 header then that route has to be used. If the node fails to forward the packet with that specified source route, than the packet should be dropped.
- If there is an entry in the routing table matching the destination that has been learned from a multicast destination advertisement, that that successor has to be used.
- If there is an entry in the routing table matching the destination that has been learned from a unicast destination advertisement, than that successor has to be used.
- If there is a DODAG Version offering a route to a prefix matching the destination, then select one of those DODAG parents as a successor according to the OF and routing metrics.
- Any other as-yet-unattempt DODAG parent may be chosen for the next attempt to forward a unicast packet when no better match exists.

It is important to note that the chosen successor can't be the neighbor that was the predecessor of the packet (split horizon), except in the case where it is intended for the packet to change from an upward to a downward direction.

RPL loop avoidance mechanisms are kept simple and designed to minimize churn and states. RPL includes a reactive loop detection technique that protects from meltdown and triggers repair of broken paths. This mechanism uses RPL Packet Information that is transported within the data packets, relying on an external mechanism that places in the RPL Packet Information in an IPv6 hop-by-hop option header. An important bit flag in the RPL Packet Information is the Down 'O' bit, which indicates whether the packet is expected to progress Up or Down and is useful to detect DODAG inconsistencies. If the direction of a packet does not match the rank relationship, a receiver can detect an inconsistency by checking the 'O' bit: if it is set (to

down) from a node of a higher rank or if it is cleared (for up) from a node of a lesser rank.
One inconsistency along the path is not considered a critical error and the packet may continue. If a second detection along the path of a same packet occurs, then the packet must be dropped. This process is controlled by the Rank-Error bit associated with the packet. When an inconsistency is detected on a packet, if the Rank-Error bit was not set then the Rank-Error bit is set, while if it was set then the packet must be discarded and the trickle timer must be set.
DAO inconsistency loop recovery is a mechanism that applies to storing mode of operation only. In non-storing mode the packets are source routed to the destination and DAO inconsistencies are not corrected locally. Instead, an ICMP error with a new code "Error in Source Routing Header", a message with the same format as the "Destination Unreachable Message", is sent back to the root. A DAO inconsistency happens when a router has a downward route that was previously learned from a DAO message via a child, but that downward route is no longer valid in the child. Upon receiving a packet with a Forwarding-Error bit set, the node must remove the routing states that caused forwarding to that neighbor, clear the Forwarding-Error bit and attempt to send the packet again, which must be sent to an alternate neighbor.

4.9 Maintenance of routing adjacency

The selection of successors, along the default paths up in the DODAG, leads to the formation of routing adjacencies that require maintenance. RPL does not define any "keep-alive" mechanism to detect routing adjacency failures: this is because in many cases such a mechanism would be too expensive in terms of bandwidth and even more importantly energy (a battery operated device could not afford to send periodic "keep-alive"). Such a mechanism should preferably be reactive to traffic in order to minimize the overhead to maintain the routing adjacency and focus on links that are actually being used. Examples of reactive mechanisms that can be used include the "Neighbor Unreachability Detection" mechanism and the “Layer 2 triggers” derived from events such as association states and L2 acknowledgement.

4.10 Objective Functions

An Objective Function (OF), in conjunction with routing metrics and constrains, allows for the selection of a DODAG to join, and a number of peers in that DODAG as parents. The OF is used to compute an ordered list of parents and is also responsible to compute the rank of the device within the DODAG Version. The Objective function is indicated in the DIO message using an Objective Code Point (OCP) and is expected to follow the same abstract behavior at a node:

- The parent selection is triggered each time an event indicates that a potential next hop information is updated, which might happen upon the reception of a DIO message, a timer elapse, all DODAG parents are unavailable or a trigger indicating that the state of a candidate neighbor has changed.
- An OF scans all the candidate neighbors on all the possible interfaces of the node to check whether they can act as a router for a DODAG.
- An OF computes the rank of a node for comparison by adding to the rank of the candidate a value representing the relative locations of the node and the candidate in the DODAG version.
- As it scans all the candidate neighbors, the OF keeps the current best parent and compares its capabilities with the current candidate neighbor. The OF defines a number of tests that are critical to reach the objective.
- When the scan is complete, the preferred parent is elected and the node's rank is computed as the preferred parent rank plus the step in rank with that parent.

4.11 Initialization Mode

Any operations and parameters should be configured or negotiated dynamically rather than manually. This is especially valid in LoWPAN where the number of devices may be large and manual configuration is infeasible. This has been taken into account in the design of RPL whereby the DODAG root provides a number of parameters to the devices joining the potential sources of misconfiguration.
When a node is first powered up, the node may decide to stay silent, waiting

to receive DIO messages from DODAG of interest and not send any multicast DIO messages until it has joined a DODAG, or may decide to send one or more DIS messages as an initial probe for nearby DODAGs and in the absence of DIO messages in reply after some configurable period of time, the node may decide to root a floating DODAG and start sending multicast DIO messages.

Chapter 5

A Multi-Hop WSN implementation with ContikiRPL

5.1 Introduction

Different prototypes of waste basket filling level sensors were tested in order to compare the precision and the overall complexity of the system. The filling level data, collected at fixed time intervals, were then transmitted through a radio transmitter and, for all the prototypes, the ICradio Stick 2.4G and the ICradio Module 2.4G were used for the transmission (Figure 5.1).
The ICradio Stick 2.4G is a compact USB-dongle, specified for IEEE 802.15.4/ZigBee network applications. It is based on the AVR ATmega1281 controller and the AT86RF230 2.4GHz radio chip from Atmel. The ICradio Module 2.4G is a compact and flexible to use radio, specified for IEEE 802.15.4/ZigBee network applications. It's excellent for evaluation purposes, since most of the IO-pins of the ATmega128 are accessible on pinheads. It is based on the AVR ATmega1281 controller and the AT86RF230 2.4GHz radio chip from Atmel.

The Operative System chosen was Contiki OS 2.5, which is fully compatible with the AVR platform. Unfortunately, the ATmega 1281 controller and the AT86RF230 radio chip were never been ported on Contiki, so it was necessary to create a new platform specifically for this module. A new Contiki platform was created starting from the AVR-Zigbit platform and the AVR-Raven platform. A new folder was created: $/Contiki/platform/avr-icm230\_12/$ and some changes were also made in the $/Contiki/cpu/avr/radio/rf230bbfiles$ in order to use correctly the radiotransmitter of the Atmel Atmega 1281 Modules and USB Sticks.

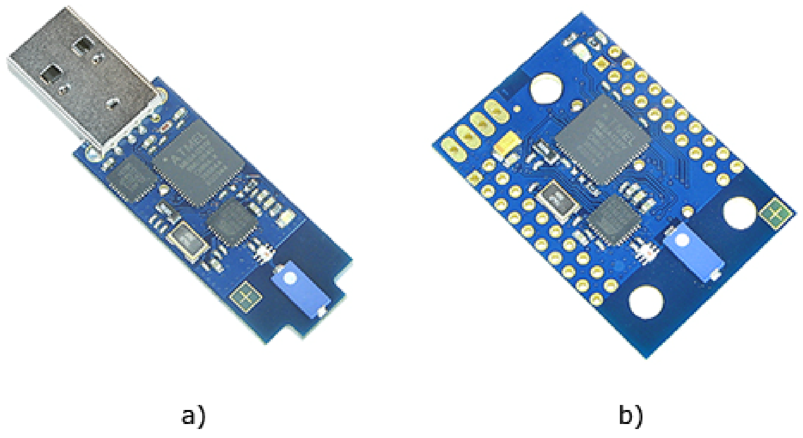


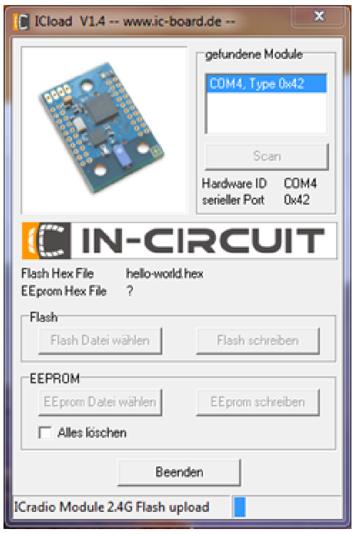
Figure 5.1: a) ICradio Stick 2.4G; b) ICradio Module 2.4G

5.2 Hello World example

"Hello world" is the first example that has been tried. First, the source code was compiled for the new platform that was created:

```
$ cd examples/hello world
$ make TARGET=avr-icm230_12 hello-world.hex
```

The .hex output file was then USB-flashed on the module with a proprietary software, *ICloud* (Figure 5.2)



COM4, Type 0x42

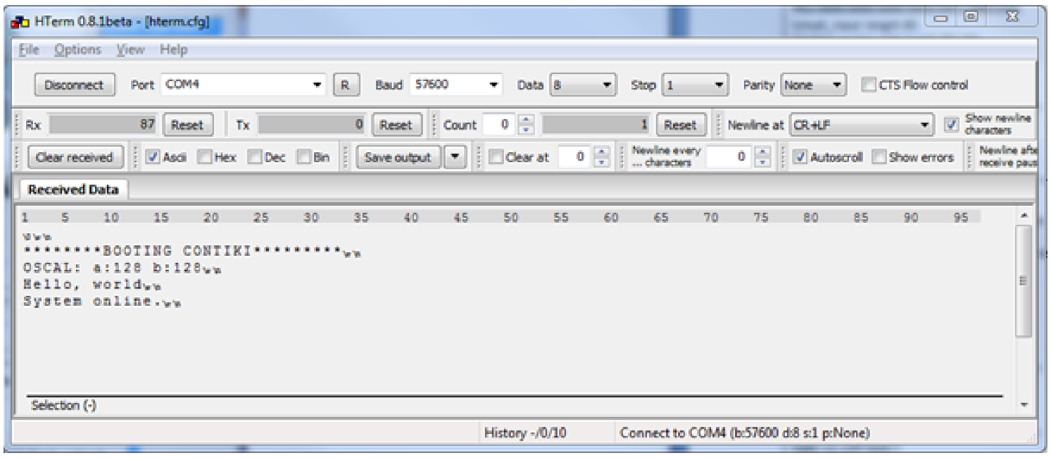
Hardware ID COM4
serieller Port 0x42

Flash Hex File hello-world.hex
EEprom Hex File ?

ICradio Module 2.4G Flash upload

Figure 5.2: ICradio Module "hello world" flash upload

With a software named *HTerm* it was possible to read the serial line output after the setup of some parameters like the COM port and the baud rate (in this case the baud rate was chosen to 57600) as shown in Figure 5.3.



Port COM4 · Baud 57600 · Data 8 · Stop 1 · Parity None · CTS Flow control [ ]
Rx 87 · Tx 0 · Count 0 · 1 · Newline at CR+LF · Show newline characters [✓]
Ascii [✓] · Hex [ ] · Dec [ ] · Bin [ ] · Clear at [ ] 0 · Newline every ... characters 0 · Autoscroll [✓] · Show errors [ ]

```
*********BOOTING CONTIKI**********
OSCAL: a:128 b:128
Hello, world
System online.
```

Selection (-)
History -/0/10 · Connect to COM4 (b:57600 d:8 s:1 p:None)

Figure 5.3: Serial line output of the Hello World example with HTerm

5.3 IPv6 Udp Sender & Receiver

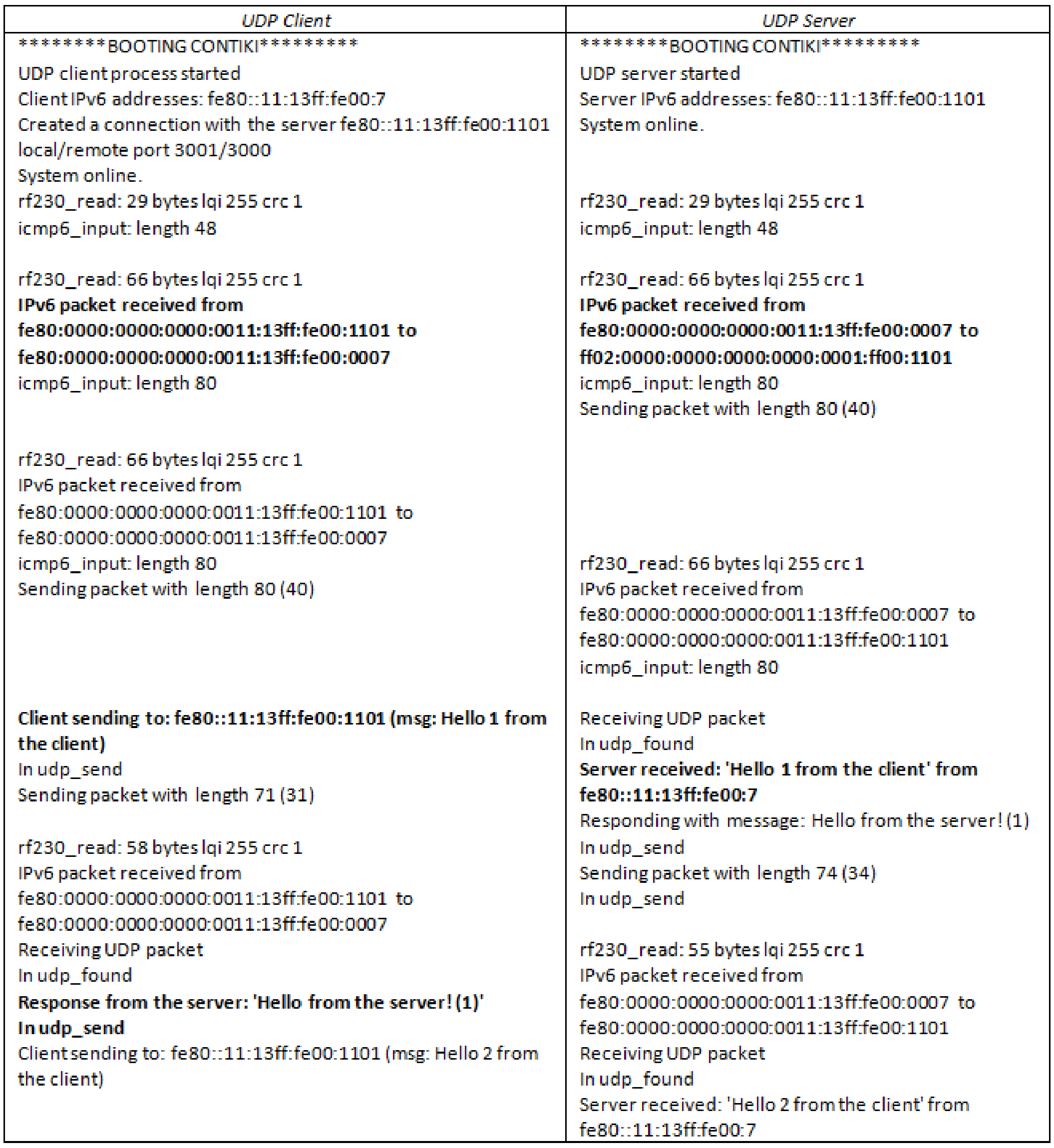
With this example it was possible to send from the "udp-client" module an UDP string to the "udp-server" and to get an answer back when the packet was successfully received.

```
$ cd examples/udp-ipv6/
$ make TARGET=avr-icm230_12 udp-server.hex udp-client.hex
```

It is important to note that in the source code it was necessary to set the client/server IPv6 address and communication port.

```
 // This is the udp-client source code.
 // The destination address follows:
 #else
 // uip_ip6addr(ipaddr,0xfe80,0,0,0,0x6466,0x9999,0x9999,0x6666);
 uip_ip6addr(ipaddr,0xbbbb,0,0,0,0x0011,0x13ff,0xfe00,0x1101);
#endif /* UDP_CONNECTION_ADDR */
```

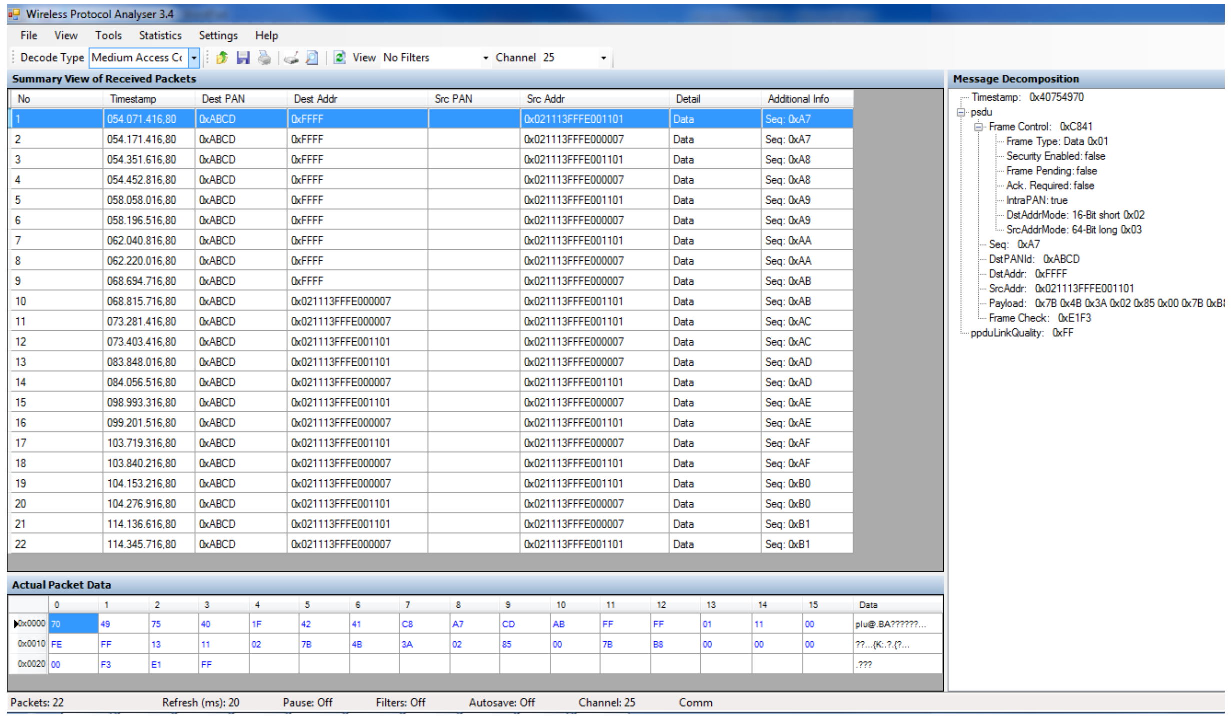
From the serial line output it was possible to read the messages exchange:



UDP Client	*UDP Server*
********BOOTING CONTIKI********* UDP client process started Client IPv6 addresses: fe80::11:13ff:fe00:7 Created a connection with the server fe80::11:13ff:fe00:1101 local/remote port 3001/3000 System online.	********BOOTING CONTIKI********* UDP server started Server IPv6 addresses: fe80::11:13ff:fe00:1101 System online.
rf230_read: 29 bytes lqi 255 crc 1 icmp6_input: length 48	rf230_read: 29 bytes lqi 255 crc 1 icmp6_input: length 48
rf230_read: 66 bytes lqi 255 crc 1 **IPv6 packet received from** **fe80:0000:0000:0000:0011:13ff:fe00:1101 to** **fe80:0000:0000:0000:0011:13ff:fe00:0007** icmp6_input: length 80	rf230_read: 66 bytes lqi 255 crc 1 **IPv6 packet received from** **fe80:0000:0000:0000:0011:13ff:fe00:0007 to** **ff02:0000:0000:0000:0000:0001:ff00:1101** icmp6_input: length 80 Sending packet with length 80 (40)
rf230_read: 66 bytes lqi 255 crc 1 IPv6 packet received from fe80:0000:0000:0000:0011:13ff:fe00:1101 to fe80:0000:0000:0000:0011:13ff:fe00:0007 icmp6_input: length 80 Sending packet with length 80 (40)	
	rf230_read: 66 bytes lqi 255 crc 1 IPv6 packet received from fe80:0000:0000:0000:0011:13ff:fe00:0007 to fe80:0000:0000:0000:0011:13ff:fe00:1101 icmp6_input: length 80
Client sending to: fe80::11:13ff:fe00:1101 (msg: Hello 1 from the client) In udp_send Sending packet with length 71 (31)	Receiving UDP packet In udp_found **Server received: 'Hello 1 from the client' from fe80::11:13ff:fe00:7** Responding with message: Hello from the server! (1) In udp_send Sending packet with length 74 (34) In udp_send
rf230_read: 58 bytes lqi 255 crc 1 IPv6 packet received from fe80:0000:0000:0000:0011:13ff:fe00:1101 to fe80:0000:0000:0000:0011:13ff:fe00:0007 Receiving UDP packet In udp_found **Response from the server: 'Hello from the server! (1)'** **In udp_send** Client sending to: fe80::11:13ff:fe00:1101 (msg: Hello 2 from the client)	rf230_read: 55 bytes lqi 255 crc 1 IPv6 packet received from fe80:0000:0000:0000:0011:13ff:fe00:0007 to fe80:0000:0000:0000:0011:13ff:fe00:1101 Receiving UDP packet In udp_found Server received: 'Hello 2 from the client' from fe80::11:13ff:fe00:7

Figure 5.4: Serial line output of the UDP client and server

With a 802.15.4 USB Stick and a software called *Wireless Protocol Analyzer* it was possible to sniff the packets in a specific channel (in this case, channel 25 was used) as shown in the figure below.



Wireless Protocol Analyser 3.4

File View Tools Statistics Settings Help

Decode Type: Medium Access C(View: No Filters Channel: 25

Summary View of Received Packets

No	Timestamp	Dest PAN	Dest Addr	Src PAN	Src Addr	Detail	Additional Info
1	054.071.416,80	0xABCD	0xFFFF		0x021113FFFE001101	Data	Seq: 0xA7
2	054.171.416,80	0xABCD	0xFFFF		0x021113FFFE000007	Data	Seq: 0xA7
3	054.351.616,80	0xABCD	0xFFFF		0x021113FFFE001101	Data	Seq: 0xA8
4	054.452.816,80	0xABCD	0xFFFF		0x021113FFFE000007	Data	Seq: 0xA8
5	058.058.016,80	0xABCD	0xFFFF		0x021113FFFE001101	Data	Seq: 0xA9
6	058.196.516,80	0xABCD	0xFFFF		0x021113FFFE000007	Data	Seq: 0xA9
7	062.040.816,80	0xABCD	0xFFFF		0x021113FFFE001101	Data	Seq: 0xAA
8	062.220.016,80	0xABCD	0xFFFF		0x021113FFFE000007	Data	Seq: 0xAA
9	068.694.716,80	0xABCD	0xFFFF		0x021113FFFE000007	Data	Seq: 0xAB
10	068.815.716,80	0xABCD	0x021113FFFE000007		0x021113FFFE001101	Data	Seq: 0xAB
11	073.281.416,80	0xABCD	0x021113FFFE000007		0x021113FFFE001101	Data	Seq: 0xAC
12	073.403.416,80	0xABCD	0x021113FFFE001101		0x021113FFFE000007	Data	Seq: 0xAC
13	083.848.016,80	0xABCD	0x021113FFFE001101		0x021113FFFE000007	Data	Seq: 0xAD
14	084.056.516,80	0xABCD	0x021113FFFE000007		0x021113FFFE001101	Data	Seq: 0xAD
15	098.993.316,80	0xABCD	0x021113FFFE001101		0x021113FFFE000007	Data	Seq: 0xAE
16	099.201.516,80	0xABCD	0x021113FFFE000007		0x021113FFFE001101	Data	Seq: 0xAE
17	103.719.316,80	0xABCD	0x021113FFFE001101		0x021113FFFE000007	Data	Seq: 0xAF
18	103.840.216,80	0xABCD	0x021113FFFE000007		0x021113FFFE001101	Data	Seq: 0xAF
19	104.153.216,80	0xABCD	0x021113FFFE000007		0x021113FFFE001101	Data	Seq: 0xB0
20	104.276.916,80	0xABCD	0x021113FFFE001101		0x021113FFFE000007	Data	Seq: 0xB0
21	114.136.616,80	0xABCD	0x021113FFFE001101		0x021113FFFE000007	Data	Seq: 0xB1
22	114.345.716,80	0xABCD	0x021113FFFE000007		0x021113FFFE001101	Data	Seq: 0xB1

Message Decomposition

```
Timestamp:   0x40754970
psdu
   Frame Control:   0xC841
      Frame Type: Data 0x01
      Security Enabled: false
      Frame Pending: false
      Ack. Required: false
      IntraPAN: true
      DstAddrMode: 16-Bit short 0x02
      SrcAddrMode: 64-Bit long 0x03
   Seq:   0xA7
   DstPANId:   0xABCD
   DstAddr:   0xFFFF
   SrcAddr:   0x021113FFFE001101
   Payload:   0x7B 0x4B 0x3A 0x02 0x85 0x00 0x7B 0xB
   Frame Check:   0xE1F3
ppduLinkQuality:   0xFF
```

Actual Packet Data

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Data
0x0000	70	49	75	40	1F	42	41	C8	A7	CD	AB	FF	FF	01	11	00	plu@.BA??????...
0x0010	FE	FF	13	11	02	7B	4B	3A	02	85	00	7B	B8	00	00	00	??...{K:.?.{?...
0x0020	00	F3	E1	FF													.???

Packets: 22 Refresh (ms): 20 Pause: Off Filters: Off Autosave: Off Channel: 25 Comm

5.4 RPL Collect: Compiling Sink node & Sender node

In this example each node was sending sensor data to a specific sink node. The source code had to be modified in order to set the sender/sink IPv6 address and communication port and then compiled.

```
$ cd examples/ipv6/rpl-collect
$ make TARGET=avr-icm230_12 udp-sink.hex udp-sender.hex
```

The RPL Debug Messages are useful to understand the topology formation of the network. It was possible to enable them in the file $/Contiki/core/net/rpl/rpl-icmp6.c$

```
//#define DEBUG DEBUG_NONE
#define DEBUG DEBUG_FULL
```

With the following command it was possible to check the memory usage of the hex file:

```
$ avr-size -C udp-sink.avr-icm230_12
AVR Memory Usage
----------------
Device: Unknown

Program:  51448 bytes
(.text + .data + .bootloader)

Data:    4476 bytes
(.data + .bss + .noinit)

EEPROM:    8 bytes
(.eeprom)
```

In case the sender node was going to be a leaf node, it wass possible to enable a flag to change the RPL ranking calculation and force it to have always infinite rank in order to reduce the power consumption. The flag which has to be enabled is in the file $/Contiki/platform/avr-icm230_12/contiki-conf.h$ as follows:

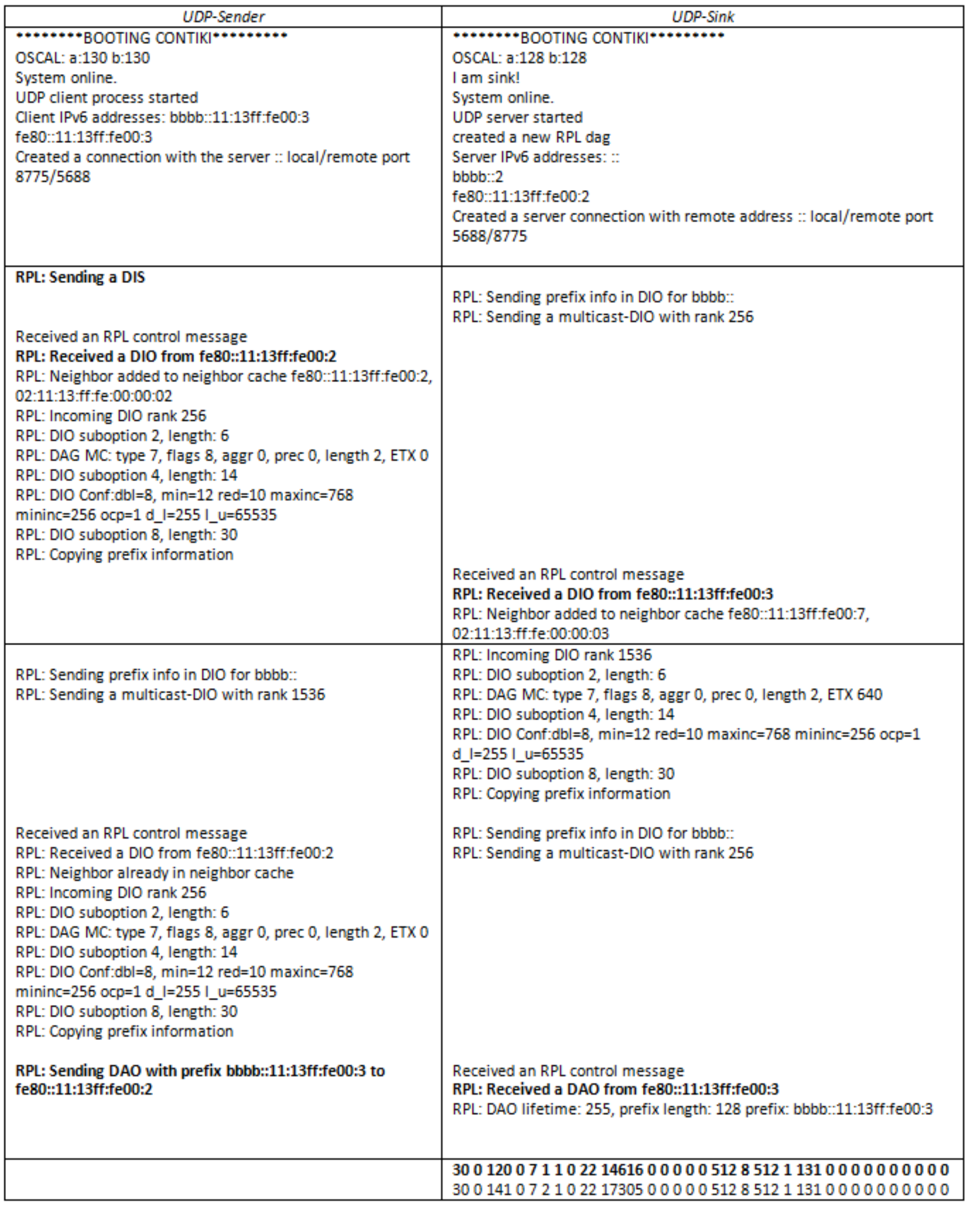
```
RPL_CONF_LEAF_NODE  1
```

It was also necessary to edit part of the $/Contiki/core/net/rpl/rpl-dag.c$ source code. The differences between the original rpl-dag.c file and the edited file, obtained with the $diff$ linux command follows:

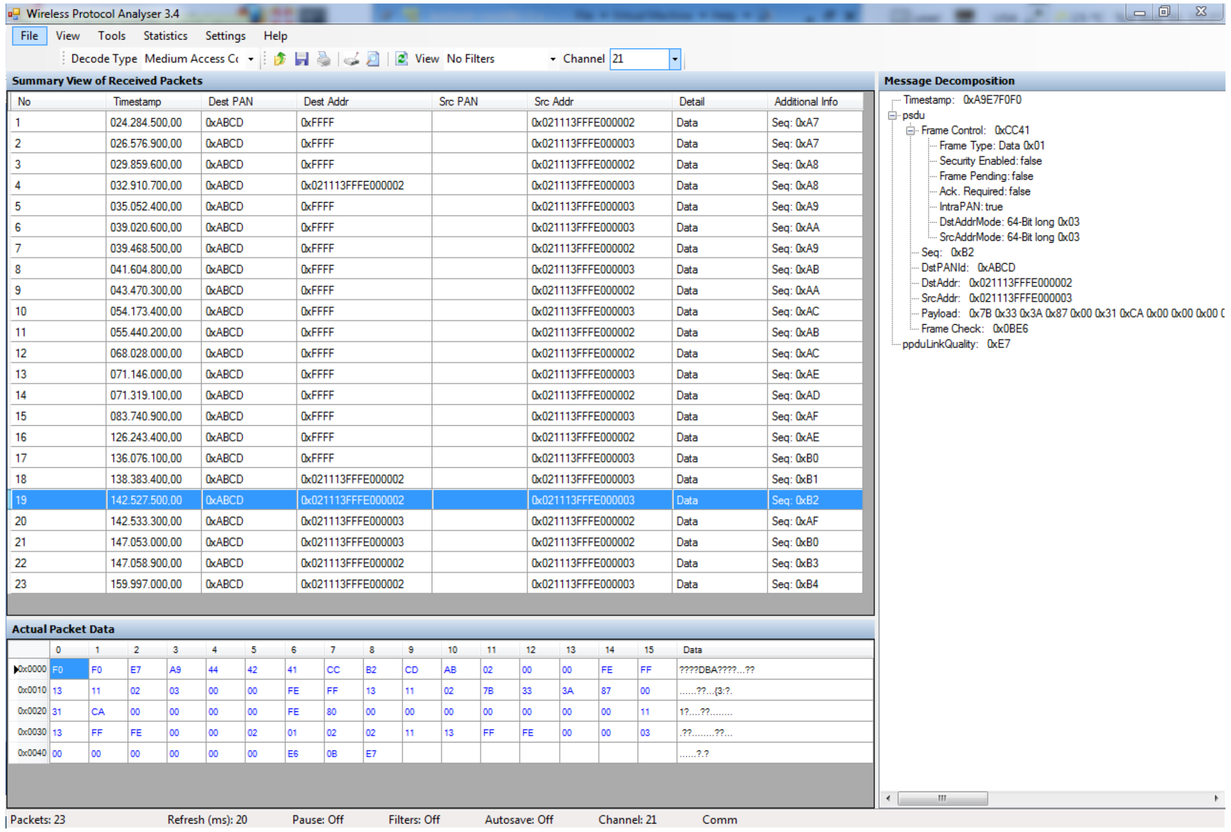
```
diff --git core/net/rpl/rpl-dag.c core/net/rpl/rpl-dag.c index 09
    b900a..02cceb8 100644
--- core/net/rpl/rpl-dag.c
+++ core/net/rpl/rpl-dag.c
@@ -373,6 +373,7 @@

   /* Update the DAG rank, since link-layer information may have
      changed the local confidence. */
+#if !RPL_CONF_LEAF_NODE
   dag->rank = dag->of->calculate_rank(best, 0);
 if(dag->rank < dag->min_rank) {
    dag->min_rank = dag->rank;
@@ -383,6 +384,7 @@
    remove_parents(dag, 0);
    return NULL;
   }
+#endif /* !RPL_CONF_LEAF_NODE */

   return best;
 }
@@ -517,8 +519,10 @@
   /* copy prefix information into the dag */
   memcpy(&dag->prefix_info, &dio->prefix_info, sizeof(
      rpl_prefix_t));

+#if !RPL_CONF_LEAF_NODE
   dag->rank = dag->of->calculate_rank(p, dio->rank);
   dag->min_rank = dag->rank; /* So far this is the lowest rank we
       know of. */
+#endif /* !RPL_CONF_LEAF_NODE */

   PRINTF("RPL:␣Joined␣DAG␣with␣instance␣ID␣%u,␣rank␣%hu,␣DAG␣ID␣"
      ,
      dio->instance_id, dag->rank);
@@ -555,8 +559,10 @@
    dag->rank = INFINITE_RANK;
   } else {
    rpl_set_default_route(dag, from);
+#if !RPL_CONF_LEAF_NODE
    dag->rank = dag->of->calculate_rank(NULL, dio->rank);
dag->min_rank = dag->rank;
+#endif /* !RPL_CONF_LEAF_NODE */
    rpl_reset_dio_timer(dag, 1);
    if(should_send_dao(dag, dio, p)) {
     rpl_schedule_dao(dag);
@@ -637,6 +643,7 @@
    return 1;
   }
```

```
+#if !RPL_CONF_LEAF_NODE
   if(DAG_RANK(old_rank, dag) != DAG_RANK(dag->rank, dag)) {
    if(dag->rank < dag->min_rank) {
     dag->min_rank = dag->rank;
@@ -656,6 +663,7 @@
     from the choice of it as a parent would be too high. */
    return 0;
   }
+#endif /* !RPL_CONF_LEAF_NODE */

   return 1;
 }
```

The serial line output and the Wireless Protocol Analyzer are useful to see how the RPL is working, as shown in the following two figures:



UDP-Sender	*UDP-Sink*
********BOOTING CONTIKI********* OSCAL: a:130 b:130 System online. UDP client process started Client IPv6 addresses: bbbb::11:13ff:fe00:3 fe80::11:13ff:fe00:3 Created a connection with the server :: local/remote port 8775/5688	********BOOTING CONTIKI********* OSCAL: a:128 b:128 I am sink! System online. UDP server started created a new RPL dag Server IPv6 addresses: :: bbbb::2 fe80::11:13ff:fe00:2 Created a server connection with remote address :: local/remote port 5688/8775
RPL: Sending a DIS Received an RPL control message **RPL: Received a DIO from fe80::11:13ff:fe00:2** RPL: Neighbor added to neighbor cache fe80::11:13ff:fe00:2, 02:11:13:ff:fe:00:00:02 RPL: Incoming DIO rank 256 RPL: DIO suboption 2, length: 6 RPL: DAG MC: type 7, flags 8, aggr 0, prec 0, length 2, ETX 0 RPL: DIO suboption 4, length: 14 RPL: DIO Conf:dbl=8, min=12 red=10 maxinc=768 mininc=256 ocp=1 d_l=255 l_u=65535 RPL: DIO suboption 8, length: 30 RPL: Copying prefix information	RPL: Sending prefix info in DIO for bbbb:: RPL: Sending a multicast-DIO with rank 256 Received an RPL control message **RPL: Received a DIO from fe80::11:13ff:fe00:3** RPL: Neighbor added to neighbor cache fe80::11:13ff:fe00:7, 02:11:13:ff:fe:00:00:03
RPL: Sending prefix info in DIO for bbbb:: RPL: Sending a multicast-DIO with rank 1536 Received an RPL control message RPL: Received a DIO from fe80::11:13ff:fe00:2 RPL: Neighbor already in neighbor cache RPL: Incoming DIO rank 256 RPL: DIO suboption 2, length: 6 RPL: DAG MC: type 7, flags 8, aggr 0, prec 0, length 2, ETX 0 RPL: DIO suboption 4, length: 14 RPL: DIO Conf:dbl=8, min=12 red=10 maxinc=768 mininc=256 ocp=1 d_l=255 l_u=65535 RPL: DIO suboption 8, length: 30 RPL: Copying prefix information **RPL: Sending DAO with prefix bbbb::11:13ff:fe00:3 to fe80::11:13ff:fe00:2**	RPL: Incoming DIO rank 1536 RPL: DIO suboption 2, length: 6 RPL: DAG MC: type 7, flags 8, aggr 0, prec 0, length 2, ETX 640 RPL: DIO suboption 4, length: 14 RPL: DIO Conf:dbl=8, min=12 red=10 maxinc=768 mininc=256 ocp=1 d_l=255 l_u=65535 RPL: DIO suboption 8, length: 30 RPL: Copying prefix information RPL: Sending prefix info in DIO for bbbb:: RPL: Sending a multicast-DIO with rank 256 Received an RPL control message **RPL: Received a DAO from fe80::11:13ff:fe00:3** RPL: DAO lifetime: 255, prefix length: 128 prefix: bbbb::11:13ff:fe00:3
	30 0 120 0 7 1 1 0 22 14616 0 0 0 0 0 512 8 512 1 131 0 0 0 0 0 0 0 0 0 0 30 0 141 0 7 2 1 0 22 17305 0 0 0 0 0 512 8 512 1 131 0 0 0 0 0 0 0 0 0 0



Decode Type Medium Access Cc   View No Filters   Channel 21

Summary View of Received Packets

No	Timestamp	Dest PAN	Dest Addr	Src PAN	Src Addr	Detail	Additional Info
1	024.284.500,00	0xABCD	0xFFFF		0x021113FFFE000002	Data	Seq: 0xA7
2	026.576.900,00	0xABCD	0xFFFF		0x021113FFFE000003	Data	Seq: 0xA7
3	029.859.600,00	0xABCD	0xFFFF		0x021113FFFE000002	Data	Seq: 0xA8
4	032.910.700,00	0xABCD	0x021113FFFE000002		0x021113FFFE000003	Data	Seq: 0xA8
5	035.052.400,00	0xABCD	0xFFFF		0x021113FFFE000003	Data	Seq: 0xA9
6	039.020.600,00	0xABCD	0xFFFF		0x021113FFFE000003	Data	Seq: 0xAA
7	039.468.500,00	0xABCD	0xFFFF		0x021113FFFE000002	Data	Seq: 0xA9
8	041.604.800,00	0xABCD	0xFFFF		0x021113FFFE000003	Data	Seq: 0xAB
9	043.470.300,00	0xABCD	0xFFFF		0x021113FFFE000002	Data	Seq: 0xAA
10	054.173.400,00	0xABCD	0xFFFF		0x021113FFFE000003	Data	Seq: 0xAC
11	055.440.200,00	0xABCD	0xFFFF		0x021113FFFE000002	Data	Seq: 0xAB
12	068.028.000,00	0xABCD	0xFFFF		0x021113FFFE000002	Data	Seq: 0xAC
13	071.146.000,00	0xABCD	0xFFFF		0x021113FFFE000003	Data	Seq: 0xAE
14	071.319.100,00	0xABCD	0xFFFF		0x021113FFFE000002	Data	Seq: 0xAD
15	083.740.900,00	0xABCD	0xFFFF		0x021113FFFE000003	Data	Seq: 0xAF
16	126.243.400,00	0xABCD	0xFFFF		0x021113FFFE000002	Data	Seq: 0xAE
17	136.076.100,00	0xABCD	0xFFFF		0x021113FFFE000003	Data	Seq: 0xB0
18	138.383.400,00	0xABCD	0x021113FFFE000002		0x021113FFFE000003	Data	Seq: 0xB1
19	142.527.500,00	0xABCD	0x021113FFFE000002		0x021113FFFE000003	Data	Seq: 0xB2
20	142.533.300,00	0xABCD	0x021113FFFE000003		0x021113FFFE000002	Data	Seq: 0xAF
21	147.053.000,00	0xABCD	0x021113FFFE000003		0x021113FFFE000002	Data	Seq: 0xB0
22	147.058.900,00	0xABCD	0x021113FFFE000002		0x021113FFFE000003	Data	Seq: 0xB3
23	159.997.000,00	0xABCD	0x021113FFFE000002		0x021113FFFE000003	Data	Seq: 0xB4

Actual Packet Data

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Data
0x0000	F0	F0	E7	A9	44	42	41	CC	B2	CD	AB	02	00	00	FE	FF	????DBA????...??
0x0010	13	11	02	03	00	00	FE	FF	13	11	02	7B	33	3A	87	00	......??...{3:?.
0x0020	31	CA	00	00	00	00	FE	80	00	00	00	00	00	00	00	11	1?....??........
0x0030	13	FF	FE	00	00	02	01	02	02	11	13	FF	FE	00	00	03	.??........??...
0x0040	00	00	00	00	00	00	E6	0B	E7								......?.?

Message Decomposition

```
Timestamp:   0xA9E7F0F0
psdu
   Frame Control:   0xCC41
      Frame Type: Data 0x01
      Security Enabled: false
      Frame Pending: false
      Ack. Required: false
      IntraPAN: true
      DstAddrMode: 64-Bit long 0x03
      SrcAddrMode: 64-Bit long 0x03
   Seq:   0xB2
   DstPANId:   0xABCD
   DstAddr:   0x021113FFFE000002
   SrcAddr:   0x021113FFFE000003
   Payload:   0x7B 0x33 0x3A 0x87 0x00 0x31 0xCA 0x00 0x00 0x00 0
   Frame Check:   0x0BE6
ppduLinkQuality:   0xE7
```

Packets: 23   Refresh (ms): 20   Pause: Off   Filters: Off   Autosave: Off   Channel: 21   Comm

5.5 RPL Border Router

An Atmel AVR Raven USB Stick (also known as Jackdaw) was used as a Border Router, following the Contiki-Wiki configuration tutorial.

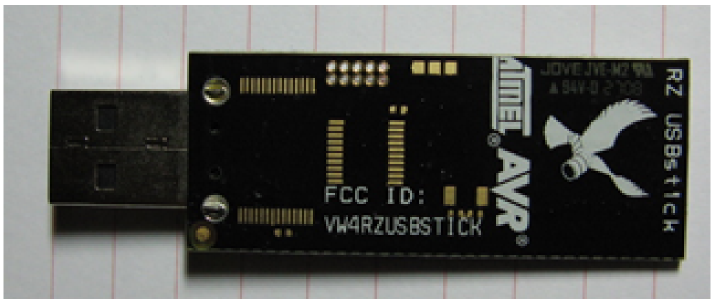


Figure 5.5: Atmel AVR Raven USB Stick

The AVR Raven USB Stick is normally a repeater with minimal IPv6 routines implemented via *fakeuip.c*, but RPL can be added by including the full uIPv6 stack in the makefile */Contiki/examples/ravenusbstick/Makefile.ravenusbstick*

```
#CONTIKI_NO_NET=1
UIP_CONF_IPV6=1
```

The firmware had to be compiled and flashed in the Jackdaw Stick. A linux application named *dfu-programmer* was used:

```
$ cd examples/ravensubstik
$ make
$ sudo dfu-programmer at90usb1287 erase
$ sudo dfu-programmer at90usb1287 flash ravenusbstick.hex
$ sudo dfu-programmer at90usb1287 start
```

In order to turn on the Border router, a *make start* command was necessary:

```
$ cd examples/ravenusbstick/
$ sudo dfu-programmer at90usb1287 start
```

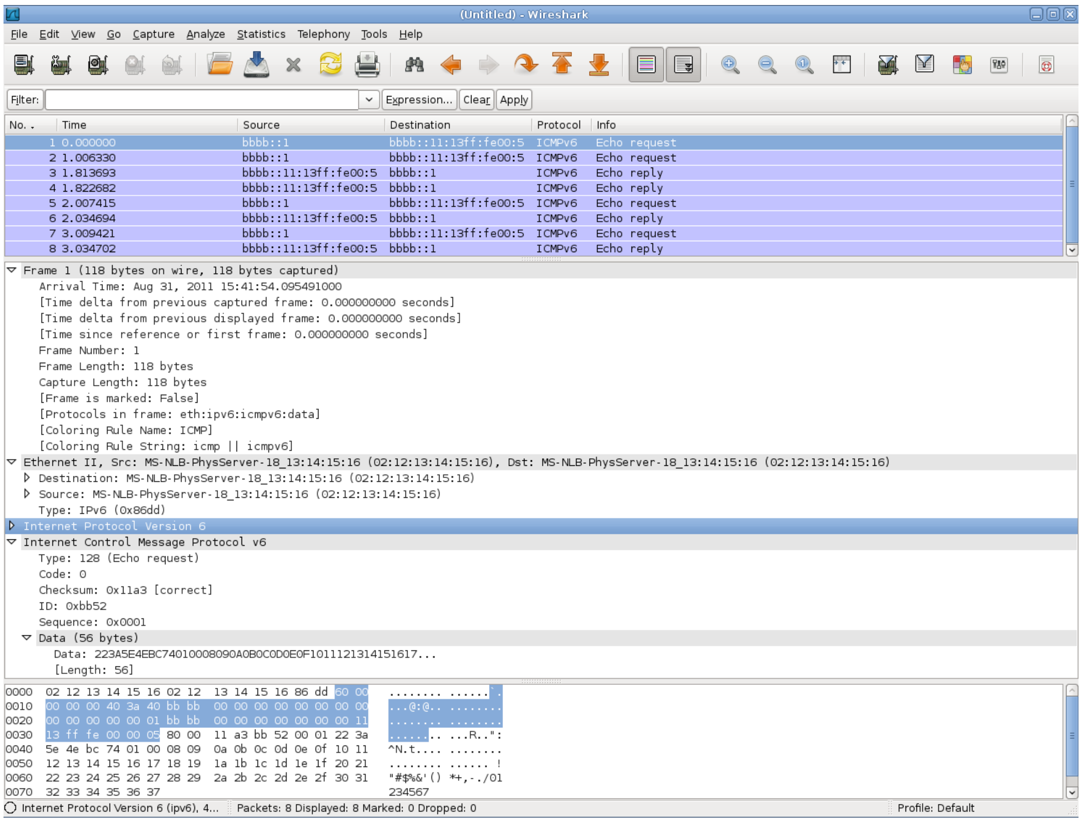
By default, RNDIS does the IPv6 address resolution protocol of Neighbor Solicitation/Neighbor Advertisement, and RPL doesn't support link-layer NS/NA broadcasts. Without a NA response, RNDIS will not send any IPv6 addressed packets. Although the Jackdaw could trap the NS and construct a NA reply, it was simple to disable NS/NA on the Linux RNDIS interface:

```
$ ifconfig usb0 -arp
$ ip -6 address add bbbb::0/64 dev usb0
```

Once the Border Router was turned on, it was possible to ping the interface at the address bbbb::1, the border router at the address bbbb::200 and every neighbor node at its IPv6 address

```
$ ping6 bbbb::1
PING bbbb::1(bbbb::1) 56 data bytes
64 bytes from bbbb::1: icmp_seq=1 ttl=64 time=0.103 ms
64 bytes from bbbb::1: icmp_seq=2 ttl=64 time=0.033 ms
64 bytes from bbbb::1: icmp_seq=3 ttl=64 time=0.032 ms
64 bytes from bbbb::1: icmp_seq=4 ttl=64 time=0.030 ms

$ ping6 bbbb::200
PING bbbb::200(bbbb::200) 56 data bytes
64 bytes from bbbb::200: icmp_seq=1 ttl=64 time=4.11 ms
64 bytes from bbbb::200: icmp_seq=2 ttl=64 time=10.2 ms
64 bytes from bbbb::200: icmp_seq=3 ttl=64 time=10.0 ms
64 bytes from bbbb::200: icmp_seq=4 ttl=64 time=6.93 ms

$ ping6 bbbb::11:13ff:fe00:5
PING bbbb::11:13ff:fe00:5(bbbb::11:13ff:fe00:5) 56 data
   bytes
64 bytes from bbbb::11:13ff:fe00:5: icmp_seq=2 ttl=63
   time=30.7 ms
64 bytes from bbbb::11:13ff:fe00:5: icmp_seq=3 ttl=63
   time=27.5 ms
64 bytes from bbbb::11:13ff:fe00:5: icmp_seq=4 ttl=63
   time=25.5 ms
64 bytes from bbbb::11:13ff:fe00:5: icmp_seq=5 ttl=63
   time=23.5 ms
```

With the help of an application named Wireshark it was possible to analyze the packets during the ping session (Figure 5.6):


(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply

```
No. . Time        Source                  Destination             Protocol Info
    1 0.000000    bbbb::1                 bbbb::11:13ff:fe00:5    ICMPv6   Echo request
    2 1.006330    bbbb::1                 bbbb::11:13ff:fe00:5    ICMPv6   Echo request
    3 1.813693    bbbb::11:13ff:fe00:5    bbbb::1                 ICMPv6   Echo reply
    4 1.822682    bbbb::11:13ff:fe00:5    bbbb::1                 ICMPv6   Echo reply
    5 2.007415    bbbb::1                 bbbb::11:13ff:fe00:5    ICMPv6   Echo request
    6 2.034694    bbbb::11:13ff:fe00:5    bbbb::1                 ICMPv6   Echo reply
    7 3.009421    bbbb::1                 bbbb::11:13ff:fe00:5    ICMPv6   Echo request
    8 3.034702    bbbb::11:13ff:fe00:5    bbbb::1                 ICMPv6   Echo reply
```

```
▽ Frame 1 (118 bytes on wire, 118 bytes captured)
    Arrival Time: Aug 31, 2011 15:41:54.095491000
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 118 bytes
    Capture Length: 118 bytes
    [Frame is marked: False]
    [Protocols in frame: eth:ipv6:icmpv6:data]
    [Coloring Rule Name: ICMP]
    [Coloring Rule String: icmp || icmpv6]
▽ Ethernet II, Src: MS-NLB-PhysServer-18_13:14:15:16 (02:12:13:14:15:16), Dst: MS-NLB-PhysServer-18_13:14:15:16 (02:12:13:14:15:16)
  ▷ Destination: MS-NLB-PhysServer-18_13:14:15:16 (02:12:13:14:15:16)
  ▷ Source: MS-NLB-PhysServer-18_13:14:15:16 (02:12:13:14:15:16)
    Type: IPv6 (0x86dd)
▷ Internet Protocol Version 6
▽ Internet Control Message Protocol v6
    Type: 128 (Echo request)
    Code: 0
    Checksum: 0x11a3 [correct]
    ID: 0xbb52
    Sequence: 0x0001
  ▽ Data (56 bytes)
      Data: 223A5E4EBC74010008090A0B0C0D0E0F1011121314151617...
      [Length: 56]
```

```
0000  02 12 13 14 15 16 02 12  13 14 15 16 86 dd 60 00   ........ ......`.
0010  00 00 00 40 3a 40 bb bb  00 00 00 00 00 00 00 00   ...@:@.. ........
0020  00 00 00 00 00 01 bb bb  00 00 00 00 00 00 00 11   ........ ........
0030  13 ff fe 00 00 05 80 00  11 a3 bb 52 00 01 22 3a   ........ ...R..":
0040  5e 4e bc 74 01 00 08 09  0a 0b 0c 0d 0e 0f 10 11   ^N.t.... ........
0050  12 13 14 15 16 17 18 19  1a 1b 1c 1d 1e 1f 20 21   ........ ...... !
0060  22 23 24 25 26 27 28 29  2a 2b 2c 2d 2e 2f 30 31   "#$%&'() *+,-./01
0070  32 33 34 35 36 37                                  234567
```

```
Internet Protocol Version 6 (ipv6), 4...   Packets: 8 Displayed: 8 Marked: 0 Dropped: 0   Profile: Default
```

Figure 5.6: Wireshark interface - Ping command

Through the debug console output it was possible to get some information about the Jackdaw Border router and the neighbors available. First of all it was necessary to connect with applications like Putty or CU (which was used in this case). In this case CU was used with the following command:

```
$ cu -l /dev/ttyACM0 --nostop -s57600
```

By pressing the key *h* or the key *?* it was possible to view the Jackdaw RPL Border Menu

```
*********** Jackdaw Menu ********
*   [Built Aug 31 2011]         *
*   m     Print current mode    *
*   s     Set to sniffer mode   *
*   n     Set to network mode   *
*   c     Set RF channel        *
*   p     Set RF power          *
*   6     Toggle 6LoWPAN        *
*   r     Toggle raw mode       *
*   d     Toggle RS232 output   *
*   S     Enable sneezer mode   *
*   N     RPL Neighbors         *
*   G     RPL Global Repair     *
*   e     Energy Scan           *
*   D     Switch to DFU mode    *
*   R     Reset (via WDT)       *
*   h,?    Print this menu      *
*                               *
*  Fraunhofher Fokus Institute  *
*********************************
```

The key *m* gives the possibility to see some useful information about the Jackdaw Border Router:

```
Currently Jackdaw:
 * Will send data over RF
 * Will change link-local addresses inside IP messages
 * Will decompress 6LoWPAN headers
 * Will not Output raw 802.15.4 frames
 * Will Output RS232 debug strings
 * USB Ethernet MAC: 02:12:13:14:15:16
 * 802.15.4 EUI-64: 02:12:13:ff:fe:14:15:16
 * Operates on channel 26 with TX power +3.0dBm
 * Current/Last/Smallest RSSI: -88/-88/-91dBm
 * Configuration: 129, USB<->ETH is active
 * Never-used stack > 800 bytes
```

The key *e* performs the energy scan in the sixteen 2.4 GHz ISM Band channels, which is useful to choose the best channel available:

```
Energy Scan:
................
 11: -80dB #########:::
 12: -74dB #################:
 13: -74dB ##################
 14: -80dB ############
 15: -92dB
```

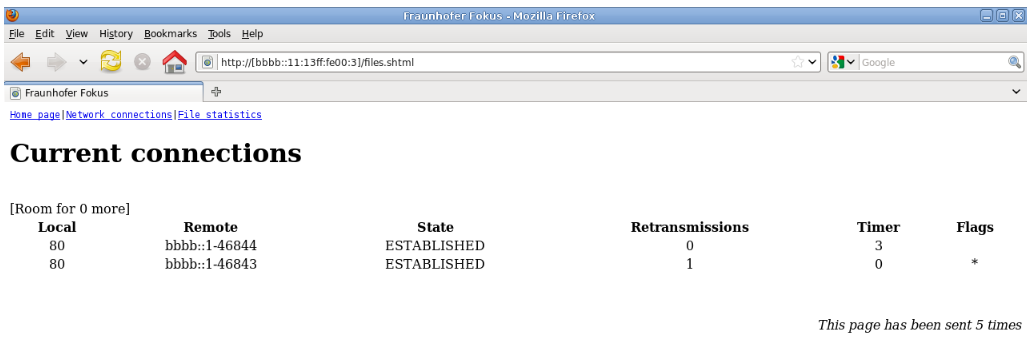
```
  16:  -83dB  : : : : : : : : : :
  17:  -83dB  : : : : : : : : : :
  18:  -74dB  : : : : : : : : : : : : : : : : : : : :
  19:  -56dB #######################################
  20:  -47dB #################################################
  21:  -47dB #################################################
  22:  -59dB ####################################
  23:  -71dB #######################:
  24:  -59dB ####################################
  25:  -59dB ####################################
  26:  -80dB #############:
Done .
```

The key *N* gives the possibility to see the neighbors and routes of the Border Router:

```
Addresses [4 max]
bbbb::200
fe80::12:13ff:fe14:1516

Neighbors [3 max]
fe80::11:13ff:fe00:9
fe80::11:13ff:fe00:3
fe80::11:13ff:fe00:5

Routes [3 max]
bbbb::11:13ff:fe00:3/128 (via fe80::11:13ff:fe00:3)
bbbb::11:13ff:fe00:9/128 (via fe80::11:13ff:fe00:5)
bbbb::11:13ff:fe00:5/128 (via fe80::11:13ff:fe00:5)
```

It is important to note that it is possible to reach and ping the node 9 (bbbb::11:13ff:fe00:9) with a multi-hop transmission through the node 5(bbbb::11:13ff:fe00:5).

If one of the neighbors runs a webserver, the border router allows the access to it through its IPv6 address. The Contiki Webserver-nano interface is shown in the figure below:


Fraunhofer Fokus - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://[bbbb::11:13ff:fe00:3]/files.shtml
Google
Fraunhofer Fokus

Home page|Network connections|File statistics

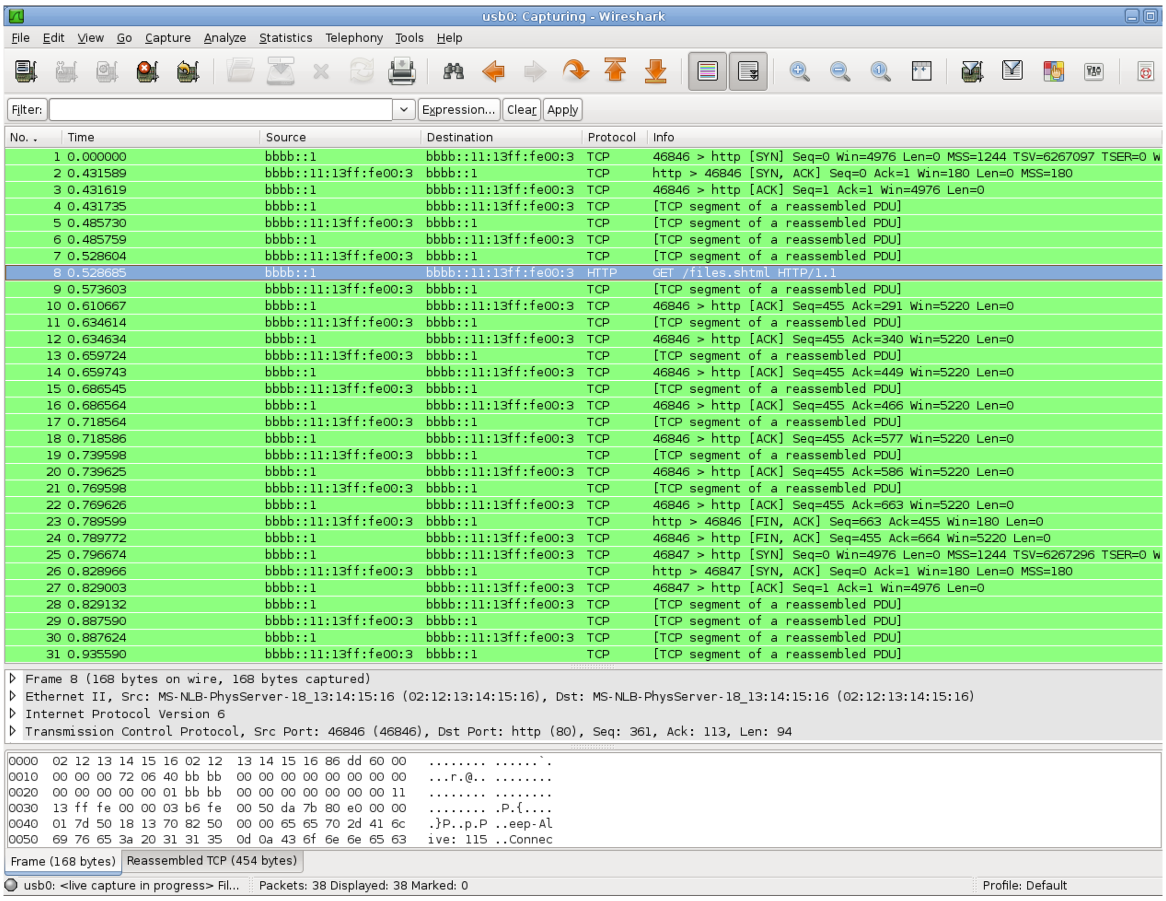
Current connections

[Room for 0 more]

Local	Remote	State	Retransmissions	Timer	Flags
80	bbbb::1-46844	ESTABLISHED	0	3	
80	bbbb::1-46843	ESTABLISHED	1	0	*

This page has been sent 5 times

It is possible to check the packets that goes in and out the AVR Raven USB Stick with a very high level of protocol details with Wireshark on the usb0 interface (Figure 5.7 and 5.8):


usb0: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression...
Clear
Apply

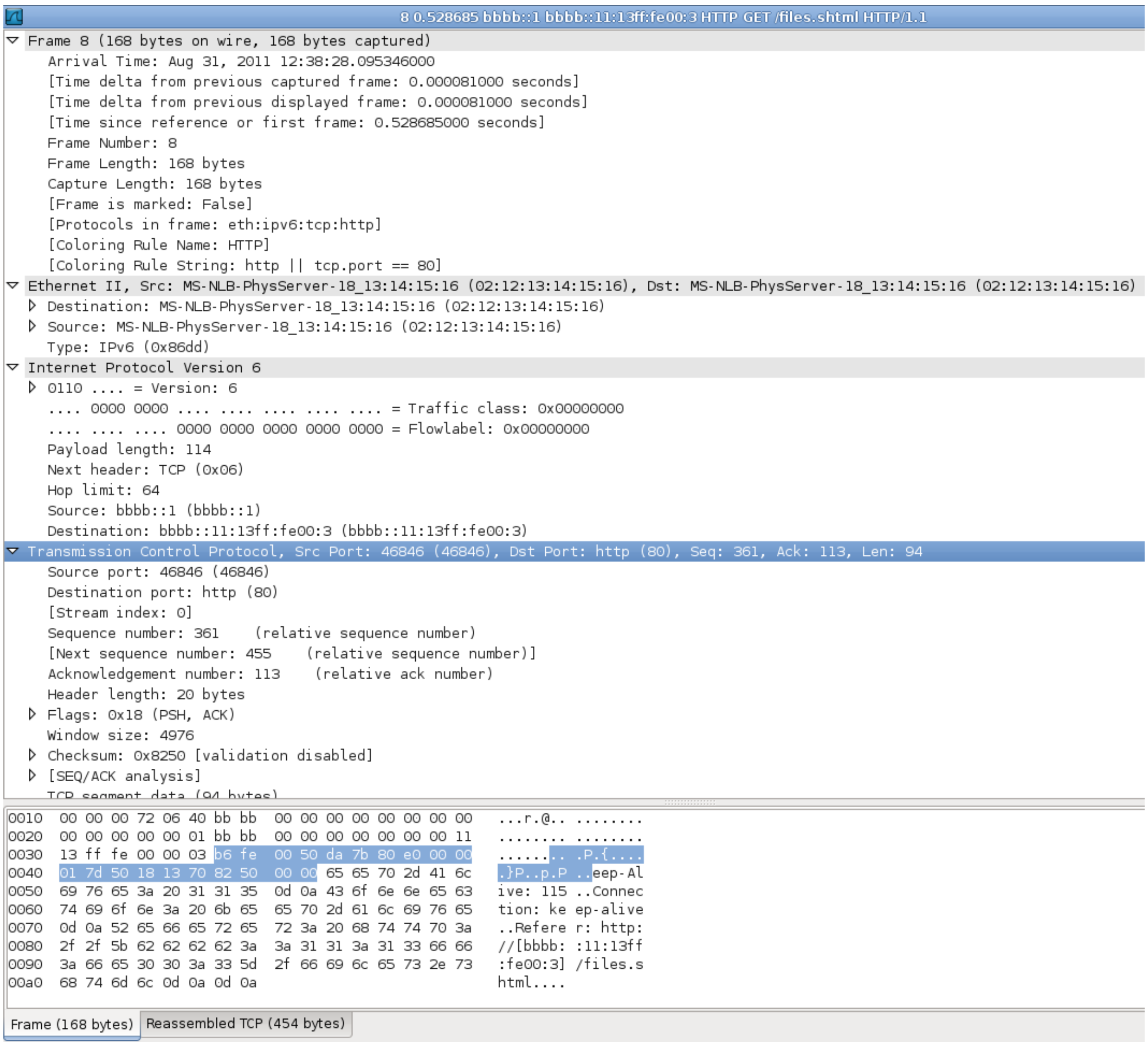
```
No. .  Time       Source                  Destination             Protocol  Info
     1 0.000000   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       46846 > http [SYN] Seq=0 Win=4976 Len=0 MSS=1244 TSV=6267097 TSER=0 W
     2 0.431589   bbbb::11:13ff:fe00:3    bbbb::1                 TCP       http > 46846 [SYN, ACK] Seq=0 Ack=1 Win=180 Len=0 MSS=180
     3 0.431619   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       46846 > http [ACK] Seq=1 Ack=1 Win=4976 Len=0
     4 0.431735   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       [TCP segment of a reassembled PDU]
     5 0.485730   bbbb::11:13ff:fe00:3    bbbb::1                 TCP       [TCP segment of a reassembled PDU]
     6 0.485759   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       [TCP segment of a reassembled PDU]
     7 0.528604   bbbb::11:13ff:fe00:3    bbbb::1                 TCP       [TCP segment of a reassembled PDU]
     8 0.528685   bbbb::1                 bbbb::11:13ff:fe00:3    HTTP      GET /files.shtml HTTP/1.1
     9 0.573603   bbbb::11:13ff:fe00:3    bbbb::1                 TCP       [TCP segment of a reassembled PDU]
    10 0.610667   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       46846 > http [ACK] Seq=455 Ack=291 Win=5220 Len=0
    11 0.634614   bbbb::11:13ff:fe00:3    bbbb::1                 TCP       [TCP segment of a reassembled PDU]
    12 0.634634   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       46846 > http [ACK] Seq=455 Ack=340 Win=5220 Len=0
    13 0.659724   bbbb::11:13ff:fe00:3    bbbb::1                 TCP       [TCP segment of a reassembled PDU]
    14 0.659743   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       46846 > http [ACK] Seq=455 Ack=449 Win=5220 Len=0
    15 0.686545   bbbb::11:13ff:fe00:3    bbbb::1                 TCP       [TCP segment of a reassembled PDU]
    16 0.686564   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       46846 > http [ACK] Seq=455 Ack=466 Win=5220 Len=0
    17 0.718564   bbbb::11:13ff:fe00:3    bbbb::1                 TCP       [TCP segment of a reassembled PDU]
    18 0.718586   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       46846 > http [ACK] Seq=455 Ack=577 Win=5220 Len=0
    19 0.739598   bbbb::11:13ff:fe00:3    bbbb::1                 TCP       [TCP segment of a reassembled PDU]
    20 0.739625   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       46846 > http [ACK] Seq=455 Ack=586 Win=5220 Len=0
    21 0.769598   bbbb::11:13ff:fe00:3    bbbb::1                 TCP       [TCP segment of a reassembled PDU]
    22 0.769626   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       46846 > http [ACK] Seq=455 Ack=663 Win=5220 Len=0
    23 0.789599   bbbb::11:13ff:fe00:3    bbbb::1                 TCP       http > 46846 [FIN, ACK] Seq=663 Ack=455 Win=180 Len=0
    24 0.789772   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       46846 > http [FIN, ACK] Seq=455 Ack=664 Win=5220 Len=0
    25 0.796674   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       46847 > http [SYN] Seq=0 Win=4976 Len=0 MSS=1244 TSV=6267296 TSER=0 W
    26 0.828966   bbbb::11:13ff:fe00:3    bbbb::1                 TCP       http > 46847 [SYN, ACK] Seq=0 Ack=1 Win=180 Len=0 MSS=180
    27 0.829003   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       46847 > http [ACK] Seq=1 Ack=1 Win=4976 Len=0
    28 0.829132   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       [TCP segment of a reassembled PDU]
    29 0.887590   bbbb::11:13ff:fe00:3    bbbb::1                 TCP       [TCP segment of a reassembled PDU]
    30 0.887624   bbbb::1                 bbbb::11:13ff:fe00:3    TCP       [TCP segment of a reassembled PDU]
    31 0.935590   bbbb::11:13ff:fe00:3    bbbb::1                 TCP       [TCP segment of a reassembled PDU]
```

```
▷ Frame 8 (168 bytes on wire, 168 bytes captured)
▷ Ethernet II, Src: MS-NLB-PhysServer-18_13:14:15:16 (02:12:13:14:15:16), Dst: MS-NLB-PhysServer-18_13:14:15:16 (02:12:13:14:15:16)
▷ Internet Protocol Version 6
▷ Transmission Control Protocol, Src Port: 46846 (46846), Dst Port: http (80), Seq: 361, Ack: 113, Len: 94
```

```
0000  02 12 13 14 15 16 02 12  13 14 15 16 86 dd 60 00   ........ ......`.
0010  00 00 00 72 06 40 bb bb  00 00 00 00 00 00 00 00   ...r.@.. ........
0020  00 00 00 00 00 01 bb bb  00 00 00 00 00 00 00 11   ........ ........
0030  13 ff fe 00 00 03 b6 fe  00 50 da 7b 80 e0 00 00   ........ .P.{....
0040  01 7d 50 18 13 70 82 50  00 00 65 65 70 2d 41 6c   .}P..p.P ..eep-Al
0050  69 76 65 3a 20 31 31 35  0d 0a 43 6f 6e 6e 65 63   ive: 115 ..Connec
```

```
Frame (168 bytes)  Reassembled TCP (454 bytes)
usb0: <live capture in progress> Fil...   Packets: 38 Displayed: 38 Marked: 0   Profile: Default
```

Figure 5.7: Wireshark output during a webserver index request



```
8 0.528685 bbbb::1 bbbb::11:13ff:fe00:3 HTTP GET /files.shtml HTTP/1.1
▽ Frame 8 (168 bytes on wire, 168 bytes captured)
    Arrival Time: Aug 31, 2011 12:38:28.095346000
    [Time delta from previous captured frame: 0.000081000 seconds]
    [Time delta from previous displayed frame: 0.000081000 seconds]
    [Time since reference or first frame: 0.528685000 seconds]
    Frame Number: 8
    Frame Length: 168 bytes
    Capture Length: 168 bytes
    [Frame is marked: False]
    [Protocols in frame: eth:ipv6:tcp:http]
    [Coloring Rule Name: HTTP]
    [Coloring Rule String: http || tcp.port == 80]
▽ Ethernet II, Src: MS-NLB-PhysServer-18_13:14:15:16 (02:12:13:14:15:16), Dst: MS-NLB-PhysServer-18_13:14:15:16 (02:12:13:14:15:16)
  ▷ Destination: MS-NLB-PhysServer-18_13:14:15:16 (02:12:13:14:15:16)
  ▷ Source: MS-NLB-PhysServer-18_13:14:15:16 (02:12:13:14:15:16)
    Type: IPv6 (0x86dd)
▽ Internet Protocol Version 6
  ▷ 0110 .... = Version: 6
    .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 114
    Next header: TCP (0x06)
    Hop limit: 64
    Source: bbbb::1 (bbbb::1)
    Destination: bbbb::11:13ff:fe00:3 (bbbb::11:13ff:fe00:3)
▽ Transmission Control Protocol, Src Port: 46846 (46846), Dst Port: http (80), Seq: 361, Ack: 113, Len: 94
    Source port: 46846 (46846)
    Destination port: http (80)
    [Stream index: 0]
    Sequence number: 361    (relative sequence number)
    [Next sequence number: 455    (relative sequence number)]
    Acknowledgement number: 113    (relative ack number)
    Header length: 20 bytes
  ▷ Flags: 0x18 (PSH, ACK)
    Window size: 4976
  ▷ Checksum: 0x8250 [validation disabled]
  ▷ [SEQ/ACK analysis]
    TCP segment data (94 bytes)

0010  00 00 00 72 06 40 bb bb  00 00 00 00 00 00 00 00   ...r.@.. ........
0020  00 00 00 00 00 01 bb bb  00 00 00 00 00 00 00 11   ........ ........
0030  13 ff fe 00 00 03 b6 fe  00 50 da 7b 80 e0 00 00   ........ .P.{....
0040  01 7d 50 18 13 70 82 50  00 00 65 65 70 2d 41 6c   .}P..p.P ..eep-Al
0050  69 76 65 3a 20 31 31 35  0d 0a 43 6f 6e 6e 65 63   ive: 115 ..Connec
0060  74 69 6f 6e 3a 20 6b 65  65 70 2d 61 6c 69 76 65   tion: ke ep-alive
0070  0d 0a 52 65 66 65 72 65  72 3a 20 68 74 74 70 3a   ..Refere r: http:
0080  2f 2f 5b 62 62 62 62 3a  3a 31 31 3a 31 33 66 66   //[bbbb: :11:13ff
0090  3a 66 65 30 30 3a 33 5d  2f 66 69 6c 65 73 2e 73   :fe00:3] /files.s
00a0  68 74 6d 6c 0d 0a 0d 0a                            html....

Frame (168 bytes)   Reassembled TCP (454 bytes)
```

Figure 5.8: Wireshark output - HTTP packet details

5.6 Multi-hop UDP - RPL Collect

The RPL-Collect example was used in order to test the multi-hop packet transmission in an 6LoWPAN network. Node 2 (address bbbb::2) runs the udp-sink.c code, while nodes 3, 5, 8 and 9 (addressesbbbb::11:13ff:fe00:X, where X is the node number)run the udp-sender.c code. The node location in the Fraunhofer FOKUS department is shown in Figure 5.9.

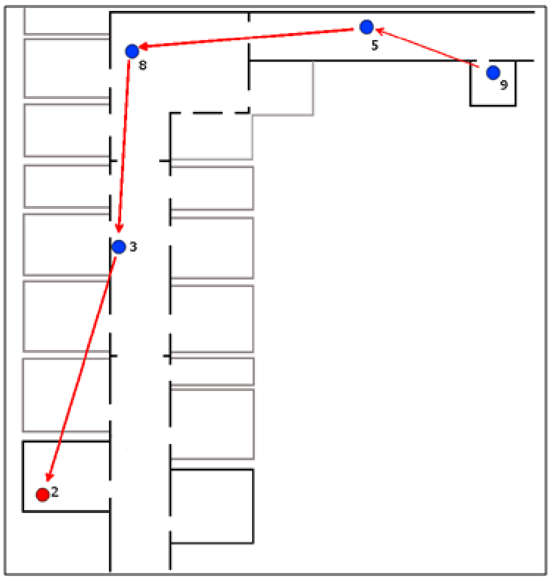


Figure 5.9: RPL Collect multi-hop transmission: first test at the Fraunhofer FOKUS department

The serial line output of the sink node (from now simply node 2, from its address fe80::11:13ff:fe00:2), with the RPL debug messages enabled, follows:

```
********BOOTING CONTIKI*********
OSCAL: a:128 b:128
I am sink!
System online.
UDP server started
created a new RPL dag
Server IPv6 addresses: ::
bbbb::2
fe80::11:13ff:fe00:2
Created a server connection with remote address :: local/remote
    port 5688/8775
RPL: Sending prefix info in DIO for bbbb::
RPL: Sending a multicast-DIO with rank 256
RPL: Sending prefix info in DIO for bbbb::
RPL: Sending a multicast-DIO with rank 256
```

```
Received an RPL control message
RPL: Received a DIS from fe80::11:13ff:fe00:3
RPL: Multicast DIS => reset DIO timer
RPL: Sending prefix info in DIO for bbbb::
RPL: Sending a multicast-DIO with rank 256
Received an RPL control message
RPL: Received a DIO from fe80::11:13ff:fe00:3
RPL: Neighbor added to neighbor cache fe80::11:13ff:fe00:3,
    02:11:13:ff:fe:00:00:03
RPL: Incoming DIO rank 1536
RPL: DIO suboption 2, length: 6
RPL: DAG MC: type 7, flags 8, aggr 0, prec 0, length 2, ETX 640
RPL: DIO suboption 4, length: 14
RPL: DIO Conf:dbl=8, min=12 red=10 maxinc=768 mininc=256 ocp=1
    d_l=255 l_u=65535
RPL: DIO suboption 8, length: 30
RPL: Copying prefix information
RPL: Sending prefix info in DIO for bbbb::
RPL: Sending a multicast-DIO with rank 256
Received an RPL control message
RPL: Received a DAO from fe80::11:13ff:fe00:3
RPL: DAO lifetime: 255, prefix length: 128 prefix: bbbb::11:13ff:
    fe00:3
RPL: Added a route to bbbb::11:13ff:fe00:3/128 via fe80::11:13ff:
    fe00:3
Received an RPL control message
RPL: Received a DIO from fe80::11:13ff:fe00:3
RPL: Neighbor already in neighbor cache
RPL: Incoming DIO rank 512
RPL: DIO suboption 2, length: 6
RPL: DAG MC: type 7, flags 8, aggr 0, prec 0, length 2, ETX 128
RPL: DIO suboption 4, length: 14
RPL: DIO Conf:dbl=8, min=12 red=10 maxinc=768 mininc=256 ocp=1
    d_l=255 l_u=65535
RPL: DIO suboption 8, length: 30
RPL: Copying prefix information
Received an RPL control message
RPL: Received a DIO from fe80::11:13ff:fe00:8
RPL: Neighbor added to neighbor cache fe80::11:13ff:fe00:8,
    02:11:13:ff:fe:00:00:08
RPL: Incoming DIO rank 1792
RPL: DIO suboption 2, length: 6
RPL: DAG MC: type 7, flags 8, aggr 0, prec 0, length 2, ETX 768
RPL: DIO suboption 4, length: 14
RPL: DIO Conf:dbl=8, min=12 red=10 maxinc=768 mininc=256 ocp=1
    d_l=255 l_u=65535
RPL: DIO suboption 8, length: 30
RPL: Copying prefix information
Received an RPL control message
```

```
RPL: Received a DIO from fe80::11:13ff:fe00:3
RPL: Neighbor already in neighbor cache
RPL: Incoming DIO rank 512
RPL: DIO suboption 2, length: 6
RPL: DAG MC: type 7, flags 8, aggr 0, prec 0, length 2, ETX 128
RPL: DIO suboption 4, length: 14
RPL: DIO Conf:dbl=8, min=12 red=10 maxinc=768 mininc=256 ocp=1
    d_l=255 l_u=65535
RPL: DIO suboption 8, length: 30
RPL: Copying prefix information
RPL: Sending prefix info in DIO for bbbb::
RPL: Sending a multicast-DIO with rank 256
Received an RPL control message
RPL: Received a DIO from fe80::11:13ff:fe00:8
RPL: Neighbor already in neighbor cache
RPL: Incoming DIO rank 768
RPL: DIO suboption 2, length: 6
RPL: DAG MC: type 7, flags 8, aggr 0, prec 0, length 2, ETX 256
RPL: DIO suboption 4, length: 14
RPL: DIO Conf:dbl=8, min=12 red=10 maxinc=768 mininc=256 ocp=1
    d_l=255 l_u=65535
RPL: DIO suboption 8, length: 30
RPL: Copying prefix information
RPL: Sending prefix info in DIO for bbbb::
RPL: Sending a multicast-DIO with rank 256
Received an RPL control message
RPL: Received a DIO from fe80::11:13ff:fe00:3
RPL: Neighbor already in neighbor cache
RPL: Incoming DIO rank 512
RPL: DIO suboption 2, length: 6
RPL: DAG MC: type 7, flags 8, aggr 0, prec 0, length 2, ETX 128
RPL: DIO suboption 4, length: 14
RPL: DIO Conf:dbl=8, min=12 red=10 maxinc=768 mininc=256 ocp=1
    d_l=255 l_u=65535
RPL: DIO suboption 8, length: 30
RPL: Copying prefix information
RPL: Sending prefix info in DIO for bbbb::
RPL: Sending a multicast-DIO with rank 256
SENSOR DATA: 1314370348697 30 0 134 0 3 1 1 0 22 14617 0 0 0 0 0
    512 8 512 2 131 0 0 0 0 0 0 0 0 0 0
Received an RPL control message
RPL: Received a DIO from fe80::11:13ff:fe00:3
RPL: Neighbor already in neighbor cache
RPL: Incoming DIO rank 512
RPL: DIO suboption 2, length: 6
RPL: DAG MC: type 7, flags 8, aggr 0, prec 0, length 2, ETX 128
RPL: DIO suboption 4, length: 14
RPL: DIO Conf:dbl=8, min=12 red=10 maxinc=768 mininc=256 ocp=1
    d_l=255 l_u=65535
```

```
RPL: DIO suboption 8, length: 30
RPL: Copying prefix information
RPL: Neighbor fe80::11:13ff:fe00:3 is known. ETX = 1
SENSOR DATA: 1314370362957 30 0 149 0 8 1 2 0 22 14617 0 0 0 0 0
    768 8 768 2 131 0 0 0 0 0 0 0 0 0 0
SENSOR DATA: 1314370369985 30 0 156 0 3 2 1 0 22 17306 0 0 0 0 0
    512 8 512 2 131 0 0 0 0 0 0 0 0 0 0
SENSOR DATA: 1314370377017 30 0 163 0 5 1 3 0 22 14617 0 0 0 0 0
    2048 8 1024 1 131 0 0 0 0 0 0 0 0 0 0
SENSOR DATA: 1314370384045 30 0 170 0 8 2 2 0 22 17306 0 0 0 0 0
    768 8 768 2 131 0 0 0 0 0 0 0 0 0 0
SENSOR DATA: 1314370398077 30 0 184 0 5 2 3 0 22 17306 0 0 0 0 0
    2048 8 1024 1 131 0 0 0 0 0 0 0 0 0 0
SENSOR DATA: 1314370446113 30 0 233 0 3 3 1 0 22 26825 0 0 0 0 0
    512 8 512 2 262 0 0 0 0 0 0 0 0 0 0
```

Multi-hop Messages: node 5 (fe80::11:13ff:fe00:5) receives the RPL messages and the packets from node 9 and forwards them to node 8. This is visible from the RPL debug messages of node 5.

```
RPL: Sending prefix info in DIO for bbbb::
RPL: Sending a multicast-DIO with rank 768
Received an RPL control message
RPL: Received a DAO from fe80::11:13ff:fe00:9
RPL: DAO lifetime: 255, prefix length: 128 prefix: bbbb::11:13ff:
    fe00:9
RPL: Forwarding DAO to parent fe80::11:13ff:fe00:8
Received an RPL control message
RPL: Received a DIO from fe80::11:13ff:fe00:9
RPL: Neighbor already in neighbor cache
RPL: Incoming DIO rank 1024
RPL: DIO suboption 2, length: 6
RPL: DAG MC: type 7, flags 8, aggr 0, prec 0, length 2, ETX 384
RPL: DIO suboption 4, length: 14
RPL: DIO Conf:dbl=8, min=12 red=10 maxinc=768 mininc=256 ocp=1
    d_l=255 l_u=65535
RPL: DIO suboption 8, length: 30
RPL: Copying prefix information
```

5.6.1 Collect View

In order to view information about the packets transmitted, the number of hops, the beacon interval and many others, a Java application named Collect-View is available. It was firstly tested with the RPL-collect example showed in the previous section.

```
$ cd tools/collect-view/
$ ant
$ cd dist/
$ java -jar collect-view.jar /dev/ttyUSB0
```

In the Figure 5.10, the number of hops are shown for each node connected to the Sink, in this case Node 2 (which is connected through the USB0 linux interface to the collect-view application)

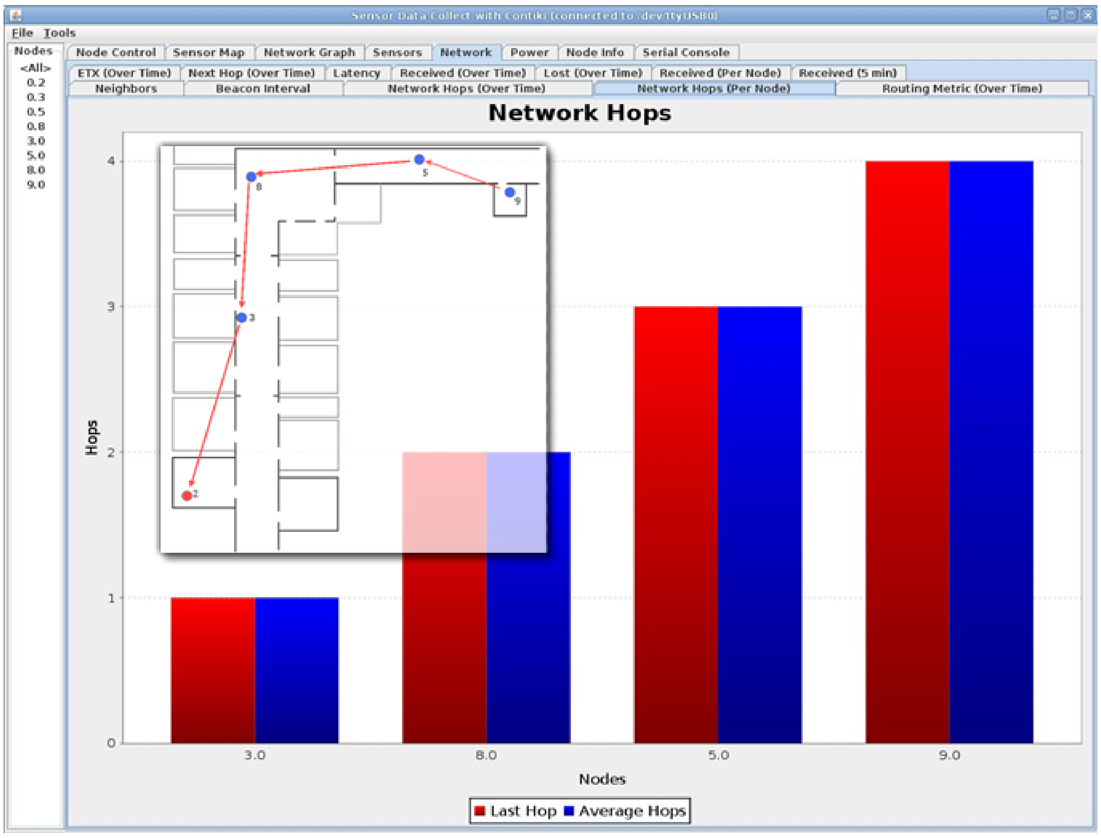
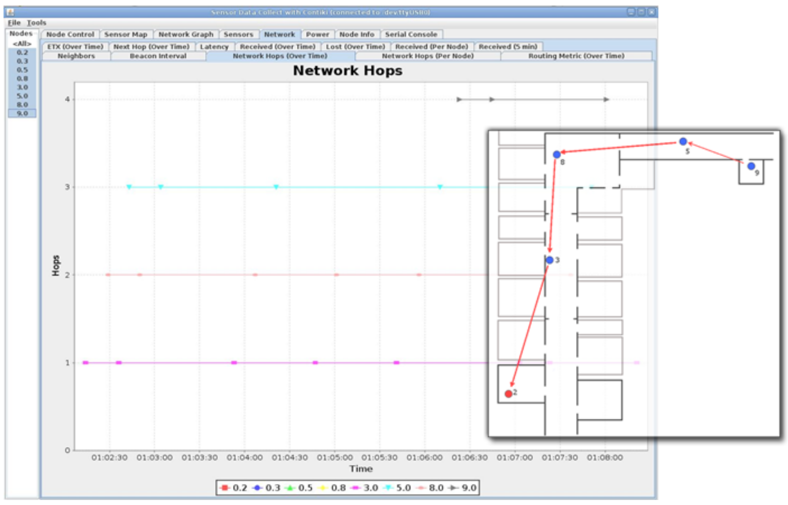
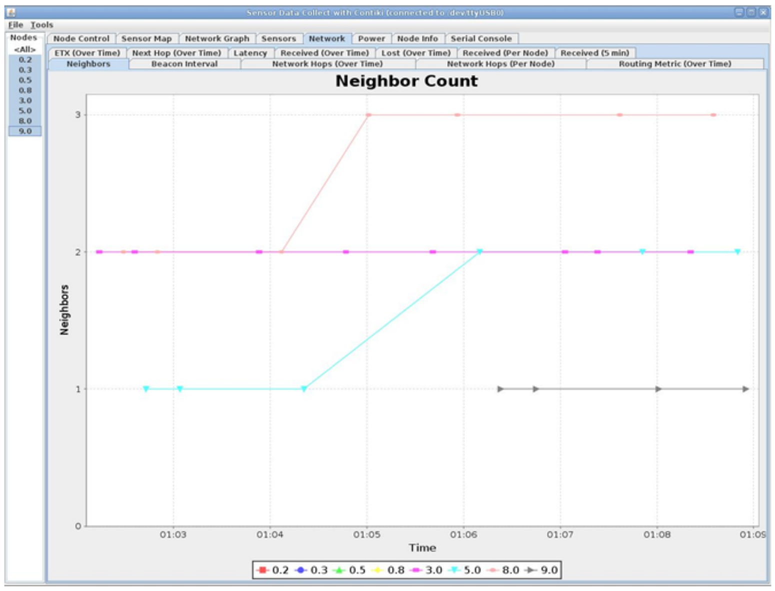


Figure 5.10: Collect View Interface: Network Hops per node with the RPL Collect example



(a) Network Hops per Node



(b) Neighbor Count over time

As it is possible to see in the Figure *b*, node 5 increases its neighbors from one (node 8) to two (node 8 and node 9), while node 8 increases the number of neighbors from two (node 3 and node 5) to three (node 3, 5 and 9).

5.7 Multi-hop WSN for Outsmart: Sensor data collection

5.7.1 Level of fullness detection

The possibility to have information about the level of fullness of every single public waste basket is very attractive and leads to many possible improvement in the waste management in big cities. The way to have a precise estimate about the level of fullness is, on the other hand, not that immediate. The idea of using a weight sensor in order to detect if the weight is larger than a certain threshold is not accurate enough because the weight is not always proportional to the occupied volume. The waste bin, in fact, could be completely full of light objects like paper or empty plastic bottles, as well as the waste basked could be almost empty but at the same time containing an heavy full bottle or some other heavy but small objects.
One of the first techniques implemented and tested in the Fraunhofer FOKUS Laboratories was using the principle of light reflection through different barriers located at different levels of height in the waste bin (Figure 5.11).

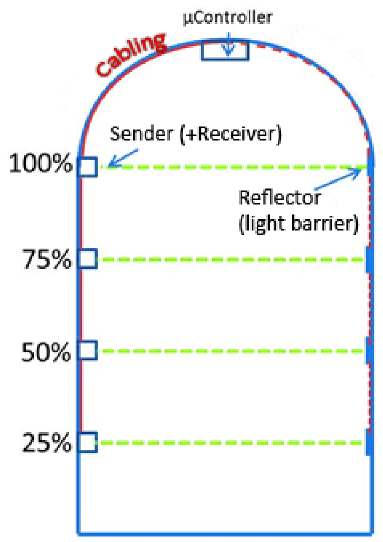


Figure 5.11: Light reflection technique

This technique was presenting several disadvantages: first of all many sensors were necessary, which makes this solution expensive. Secondly, this technology was presenting problems with transparent materials because the light was passing through the object; the space occupied by the transparent object was therefore not detected leading to an unacceptable level of fullness detection error. Moreover, when an object with a big surface (for example a newspaper) was placed inside the waste bin, it was activating many light barriers at the same time even if it was not occupying all the space detected by the sensors. A possible solution could be to place more sensor per level, which point in different directions. This would make the technology even more expensive and would be necessary to place even more cables inside the waste bin.

The second technique considered was the laser triangulation (Figure 5.12), which was characterized by the advantage of the presence of only one sensor on top of the waste bin, but also by the same disadvantage of the light sensor: the problems presented with the transparent materials.

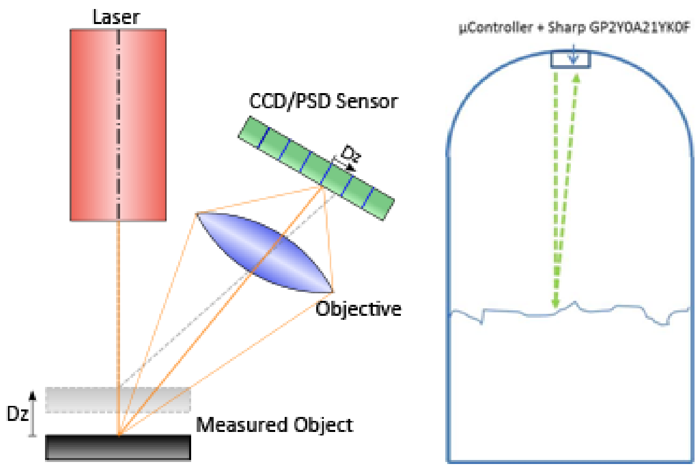


Figure 5.12: Laser triangulation technique

The third sensor used was an ultrasonic sensor (Figure 5.13), which uses the reflection of high frequency (above 18KHz) sound waves to detect parts or distances to the parts. These sensors calculate the time interval between sending the signal and receiving the echo in order to determine the distance to an object.

The distance was therefore calculated as $d = \frac{1}{2}t_x v$, where v is the speed of sound (343.2 m/s) and t_x is the time interval. One of the advantages of ultrasonic sensors is the use sound rather than light for detection, which allows these sensors to work in applications where photoelectric sensors may not suc-

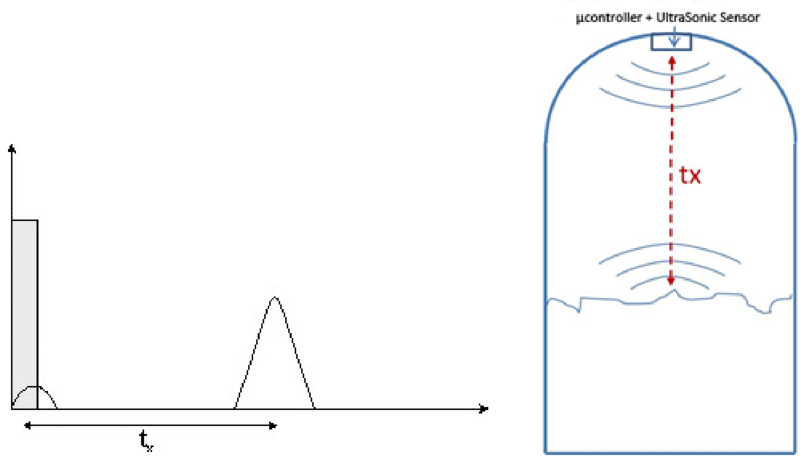
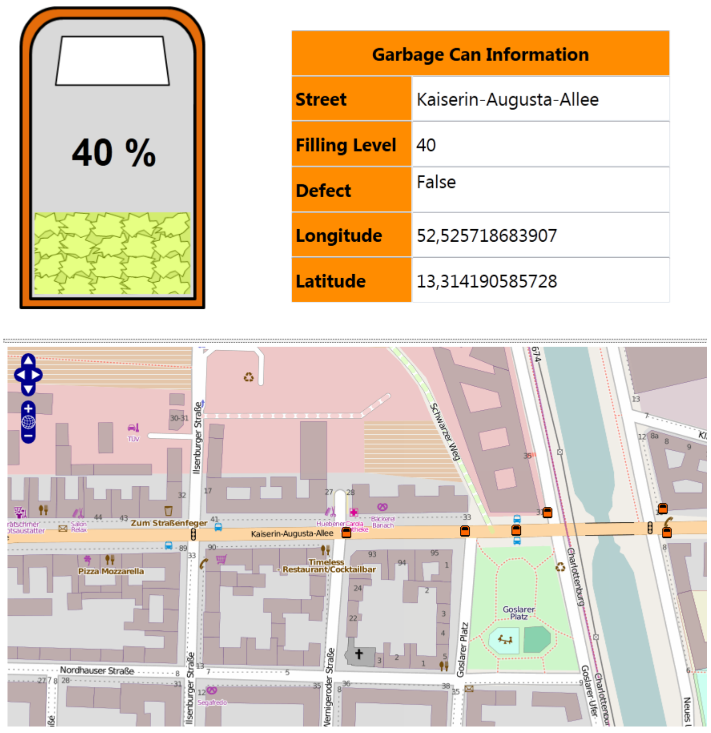


Figure 5.13: Ultrasonic sensor technique

ceed. Ultrasonic sensors are solution not only for clear object detection, but also for liquid level measurement applications, where photo-electrics struggle with because of the target translucence. Objects' color and reflectivity don't affect ultrasonic sensors which are able to operate reliably in high-glare environments. The ability to work in dark environments and the simple data processing in comparison with other sensors allows the ultrasonic sensor not only to extract the object information regardless to the kind of object, but also to recognize the type of object independently from the traslation and/or the rotation.

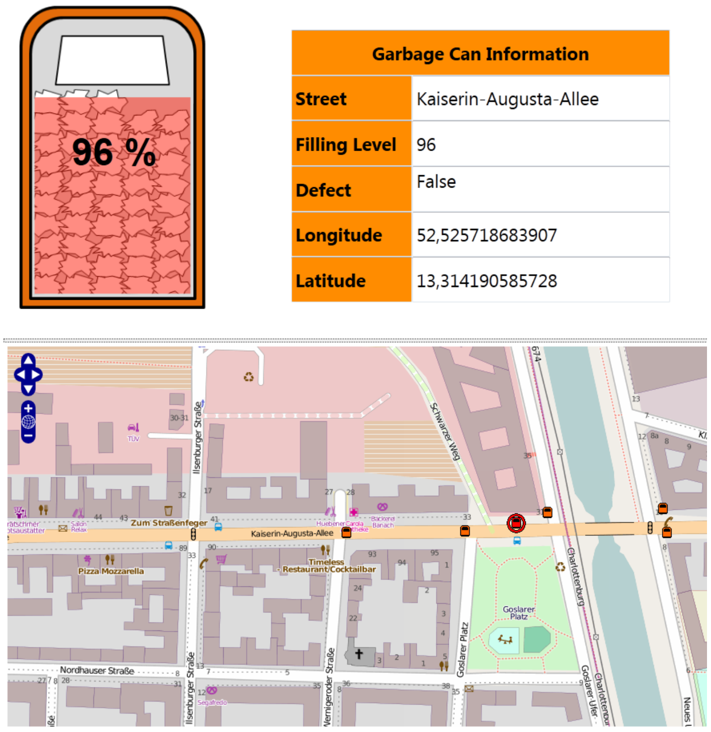
It is important to take into account the non-sensitive zone and the uncertainty zone, where the non-sensitive zone is the interval between the surface of the sensor head and the minimum detection distance resulting from detection distance adjustment. The uncertainty zone is the area close to the sensor where detection is not possible due to the sensor head configuration and reverberations. Anyway the detection may occur in the uncertainty zone due to multi-reflection between the sensor and the object. Another problem which is important to deal with is the fact that different materials, for example soft and hard materials, reflect ultrasonic way very differently, leading to some potential errors in the detection of the level of fullness.

An ad-hoc software was created to remotely control the level of fullness for every single waste basket and was then integrated with Open Street Map (www.openstreetmap.org) , a website which provides a free editable map of the World, as shown in Figure 5.14.



Garbage Can Information	
Street	Kaiserin-Augusta-Allee
Filling Level	40
Defect	False
Longitude	52,525718683907
Latitude	13,314190585728

(a) Waste basket nearly empty



Garbage Can Information	
Street	Kaiserin-Augusta-Allee
Filling Level	96
Defect	False
Longitude	52,525718683907
Latitude	13,314190585728

(b) Waste basket full

Figure 5.14: Waste Basket level of fullnes interface with OpenMap

5.7.2 Multi-Hop Sensor Network Tests

In the first test, node 3 was running the Ultrasonic sensor detector code, while node 8 was running the RPL-Collect udp-sender.c code and was used for packet forwarding in order to test the multi-hop network. The node location in the Fraunhofer FOKUS department is shown in Figure 5.15, where the waste basket is indicated with a green dot.

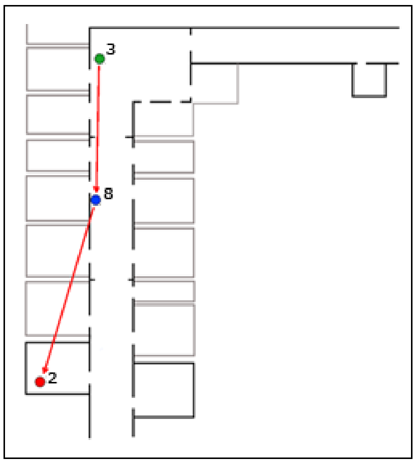
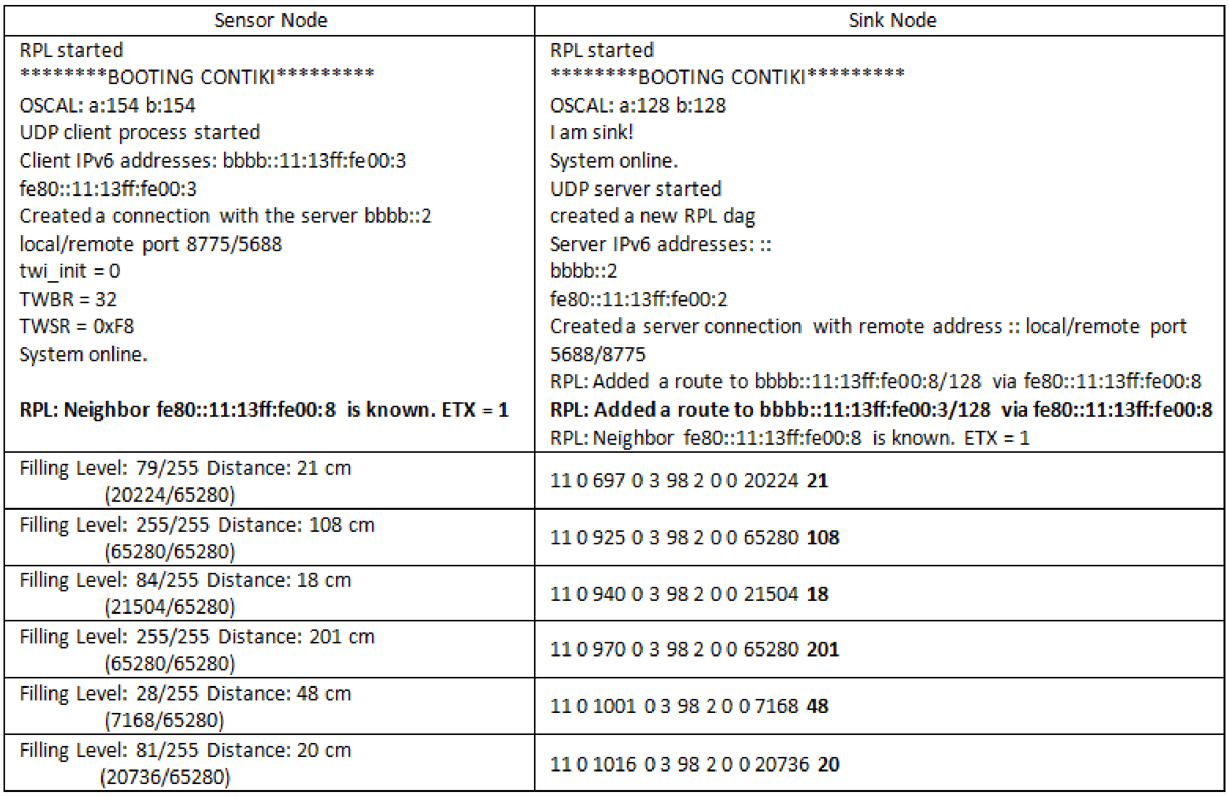


Figure 5.15: Multi-Hop sensor data collection - first experiment node location

The goal of this test was to see if the detected distance (which means the fill level of the waste basket) was transmitted to the sink node with a multi-hop transmission through node 8.

The .C code of the software running on the node equipped with the ultrasonic sensor can be found on Appendix 1.

The serial line output of the sensor node (node 3) and the sink node (node 2) follows.



Sensor Node	Sink Node
RPL started ********BOOTING CONTIKI********* OSCAL: a:154 b:154 UDP client process started Client IPv6 addresses: bbbb::11:13ff:fe00:3 fe80::11:13ff:fe00:3 Created a connection with the server bbbb::2 local/remote port 8775/5688 twi_init = 0 TWBR = 32 TWSR = 0xF8 System online. **RPL: Neighbor fe80::11:13ff:fe00:8 is known. ETX = 1**	RPL started ********BOOTING CONTIKI********* OSCAL: a:128 b:128 I am sink! System online. UDP server started created a new RPL dag Server IPv6 addresses: :: bbbb::2 fe80::11:13ff:fe00:2 Created a server connection with remote address :: local/remote port 5688/8775 RPL: Added a route to bbbb::11:13ff:fe00:8/128 via fe80::11:13ff:fe00:8 **RPL: Added a route to bbbb::11:13ff:fe00:3/128 via fe80::11:13ff:fe00:8** RPL: Neighbor fe80::11:13ff:fe00:8 is known. ETX = 1
Filling Level: 79/255 Distance: 21 cm (20224/65280)	11 0 697 0 3 98 2 0 0 20224 **21**
Filling Level: 255/255 Distance: 108 cm (65280/65280)	11 0 925 0 3 98 2 0 0 65280 **108**
Filling Level: 84/255 Distance: 18 cm (21504/65280)	11 0 940 0 3 98 2 0 0 21504 **18**
Filling Level: 255/255 Distance: 201 cm (65280/65280)	11 0 970 0 3 98 2 0 0 65280 **201**
Filling Level: 28/255 Distance: 48 cm (7168/65280)	11 0 1001 0 3 98 2 0 0 7168 **48**
Filling Level: 81/255 Distance: 20 cm (20736/65280)	11 0 1016 0 3 98 2 0 0 20736 **20**

Figure 5.16: Multi-Hop sensor data collection - first experiment serial line output

As it is possible to see the distance, which represents the filling level of the waste bin, is periodically detected by the sensor and is then transmitted to the sink node through a multi-hop network.

In the second test, the reliability of the sensor data collection is studied when one or more nodes of the network are not available anymore and thus the network topology changes.

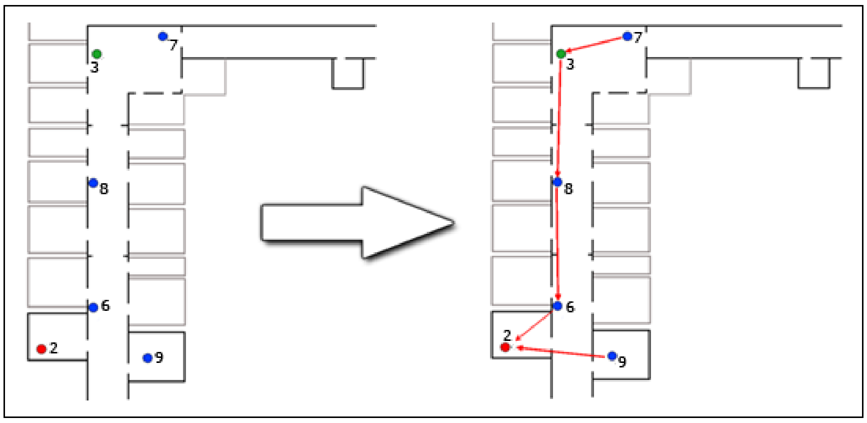


Figure 5.17: Multi-Hop sensor data collection - second experiment node location

First, a network with nodes 9 => 2<= 6 <= 8 <= 3 <= 7 is formed, as shown in Figure 5.17. Then, node 6 is disconnected: node 8 transmits directly to node 2 instead of using node 6 for a multi-hop transmission (Figure 5.18).

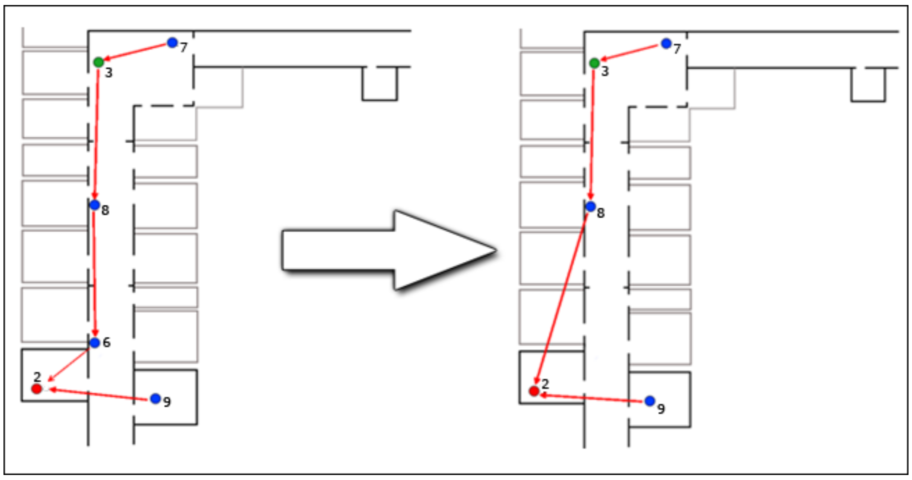
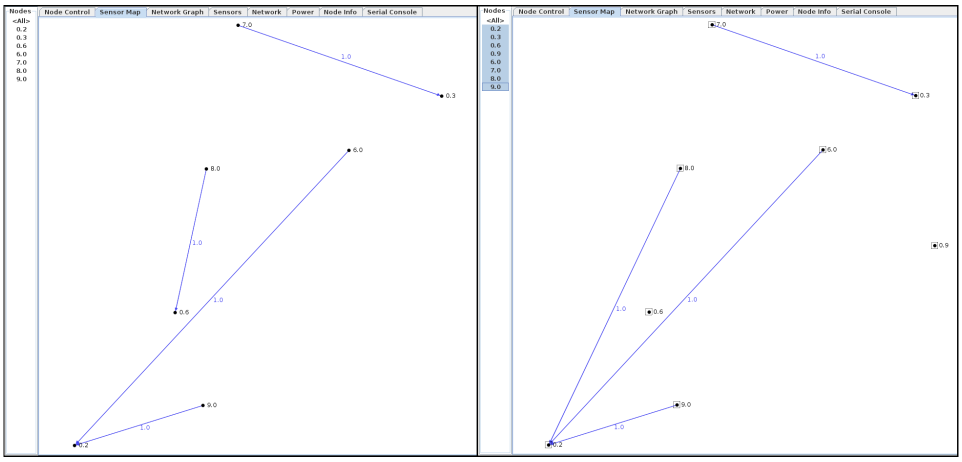
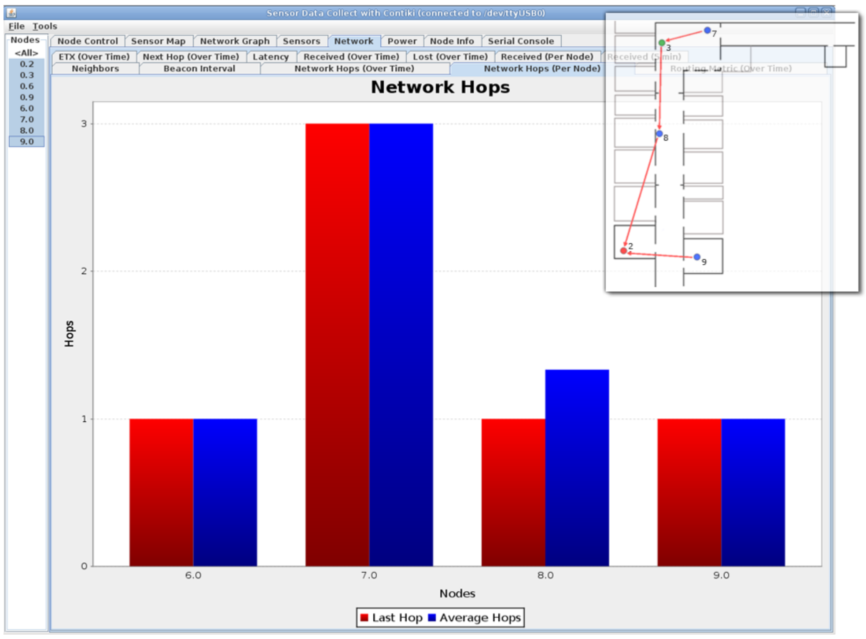


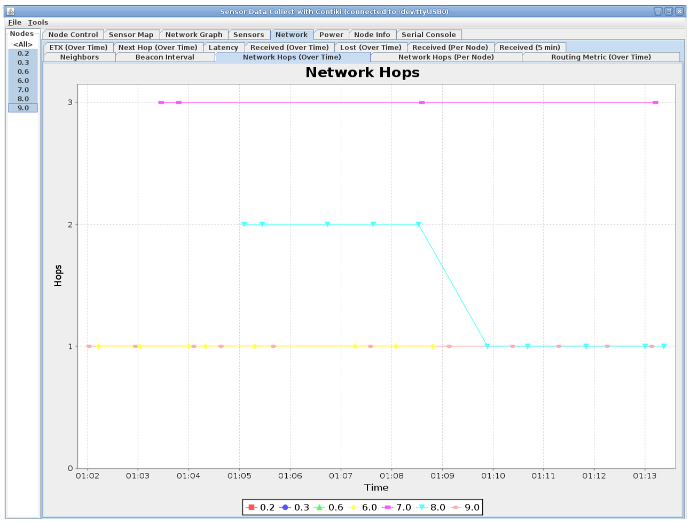
Figure 5.18: Multi-Hop sensor data collection - second experiment



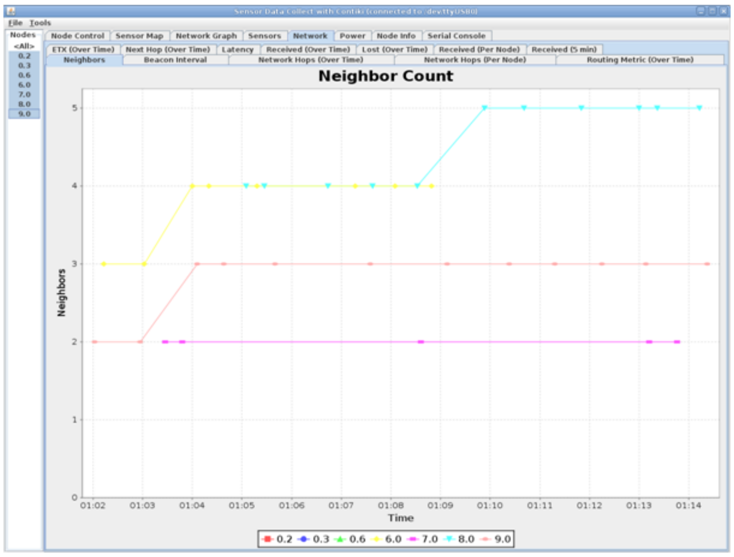
(a) Sensor Map before(left) and after (right) the disappearance of node 6



(b) Number of Hops: only node 8 reduces its hops from 2 to 1

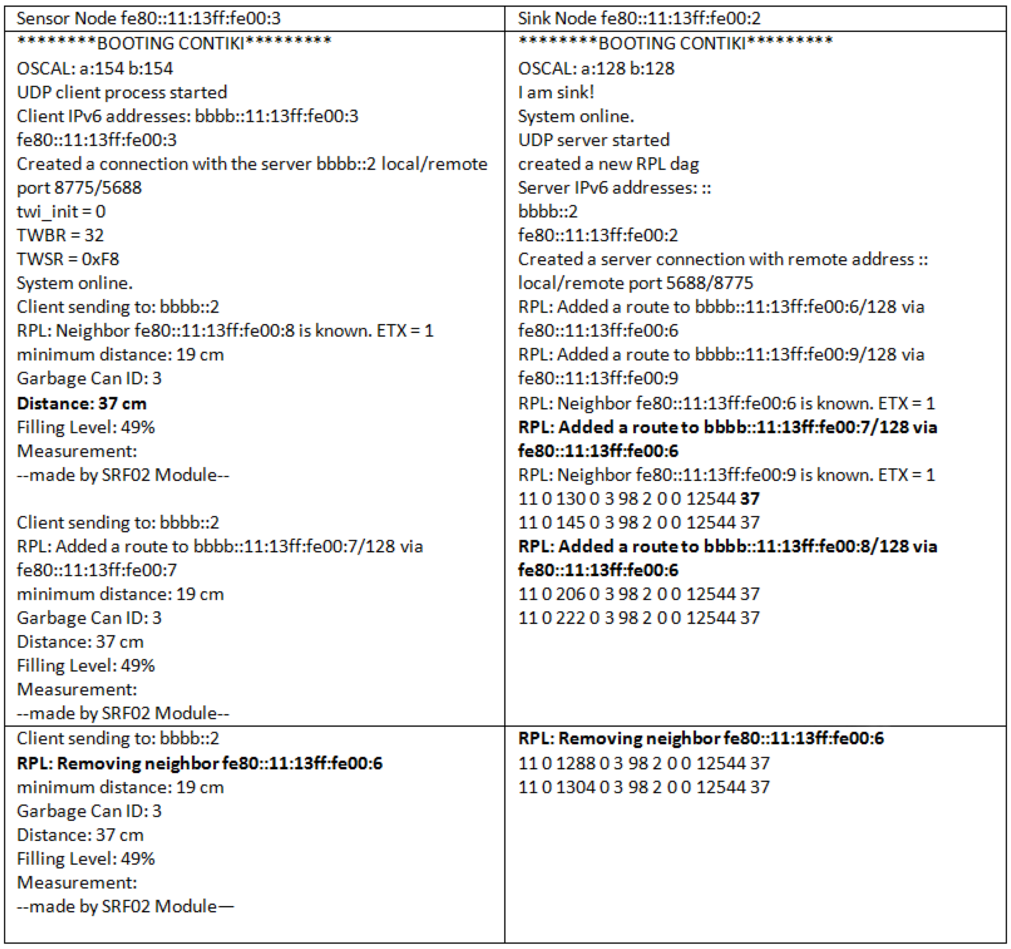


Node 8 transmits now the packets directly to node 2, therefore the number of hops decreases from 2 to 1. On the other hand node 7 doesn't change the number of hops because its packets were transmitted before through node 3 and 6 to node 2, while now they're transmitted through nodes 3 and 8 to node 2.



The number of neighbors of node 8 increases by one (node 3) when node 6 is not available anymore.

The serial line output of the sensor node and of the sink node follows:



Sensor Node fe80::11:13ff:fe00:3

```
********BOOTING CONTIKI*********
OSCAL: a:154 b:154
UDP client process started
Client IPv6 addresses: bbbb::11:13ff:fe00:3
fe80::11:13ff:fe00:3
Created a connection with the server bbbb::2 local/remote
port 8775/5688
twi_init = 0
TWBR = 32
TWSR = 0xF8
System online.
Client sending to: bbbb::2
RPL: Neighbor fe80::11:13ff:fe00:8 is known. ETX = 1
minimum distance: 19 cm
Garbage Can ID: 3
Distance: 37 cm
Filling Level: 49%
Measurement:
--made by SRF02 Module--

Client sending to: bbbb::2
RPL: Added a route to bbbb::11:13ff:fe00:7/128 via
fe80::11:13ff:fe00:7
minimum distance: 19 cm
Garbage Can ID: 3
Distance: 37 cm
Filling Level: 49%
Measurement:
--made by SRF02 Module--
```

```
Client sending to: bbbb::2
RPL: Removing neighbor fe80::11:13ff:fe00:6
minimum distance: 19 cm
Garbage Can ID: 3
Distance: 37 cm
Filling Level: 49%
Measurement:
--made by SRF02 Module—
```

Sink Node fe80::11:13ff:fe00:2

```
********BOOTING CONTIKI*********
OSCAL: a:128 b:128
I am sink!
System online.
UDP server started
created a new RPL dag
Server IPv6 addresses: ::
bbbb::2
fe80::11:13ff:fe00:2
Created a server connection with remote address ::
local/remote port 5688/8775
RPL: Added a route to bbbb::11:13ff:fe00:6/128 via
fe80::11:13ff:fe00:6
RPL: Added a route to bbbb::11:13ff:fe00:9/128 via
fe80::11:13ff:fe00:9
RPL: Neighbor fe80::11:13ff:fe00:6 is known. ETX = 1
RPL: Added a route to bbbb::11:13ff:fe00:7/128 via
fe80::11:13ff:fe00:6
RPL: Neighbor fe80::11:13ff:fe00:9 is known. ETX = 1
11 0 130 0 3 98 2 0 0 12544 37
11 0 145 0 3 98 2 0 0 12544 37
RPL: Added a route to bbbb::11:13ff:fe00:8/128 via
fe80::11:13ff:fe00:6
11 0 206 0 3 98 2 0 0 12544 37
11 0 222 0 3 98 2 0 0 12544 37
```

```
RPL: Removing neighbor fe80::11:13ff:fe00:6
11 0 1288 0 3 98 2 0 0 12544 37
11 0 1304 0 3 98 2 0 0 12544 37
```

Chapter 6

Conclusions and future implementations

After a platform creation and a set of examples used in order to get familiar with the Contiki Operative System, its file system and compilation, its serial line output and its behavior with the RPL routing protocol, with the help of the Collect View graphical interface a multi-hop 6LoWPAN transmission system was tested and then applied to the Outsmart Waste Management project. In particular, an ultrasonic sensor was used to evaluate the filling level of the waste basket every 30 seconds (the optimal time interval between the data collection has still to be analyzed), and the sensed data were transmitted through many hops to a sink node, which was displaying them through the serial line output. The transmission system has shown reliability and fast auto-configuration features, also in case of non-availability of one of the nodes.
One of the implementations that has to be made is the choice of the behavior of the sink node, which could be equipped for example with a bigger storage memory and could collect the data and transmit them only when a waste collection vehicle is in the nearby, in order to inform the vehicle about which waste basket needs to be emptied. Another approach could be to place side by side to the sink node a border router, which could periodically transmit the data collected from a specific area to a central server connected through the internet, allowing the company in charge to manage the waste collection (in this case the BSR company) to have a real-time global vision of the filling level of every single waste basket. In this way, it would be possible to organize in advance the waste collection route plan, in order to save time, gasoline and pollution, but also to have a big improvement in the city cleanness.
The choice of an IPv6 enabled open source Operative System was made to

reduce the implementation cost of the wireless sensor network and to bring the "Internet of Things" vision behind the 6LoWPAN to a real wireless sensor network system which has the possibility to improve not only the quality of the waste management, but also the quality of life of the citizens and of the tourists of a city like Berlin.

Chapter 7

Appendix

```
#include "contiki.h"
#include "contiki-lib.h"
#include "contiki-net.h"
#include "ic_module_twi.h"
#include <util/delay.h>
#include <string.h>

#define DEBUG DEBUG_PRINT
#include "net/uip-debug.h"

#define SEND_INTERVAL   15 * CLOCK_SECOND
#define MAX_PAYLOAD_LEN   40

// ********* my debugs ***********
#define DEBUG_MEASUREMENT_RESULTS 0


// ********* my defines **********

#define API_ID 'b'
#define GARBAGE_CAN_ID 3
#define SRF02_WRITE_ADDRESS 0xE0
#define SRF02_READ_ADDRESS SRF02_WRITE_ADDRESS+1
#define SRF02_CM_MEASUREMENT 0x51
#define SRF02_IN_MEASUREMENT 0x50
#define SRF02_MS_MEASUREMENT 0x52
#define SRF02_AUTOTUNE_RESTART 0x60

// ******** my global variables *******


 typedef struct{
    uint8_t api_id;
    uint32_t garbage_can_id;
    uint8_t fill_level;
    uint8_t data[120];
  }__attribute__((__packed__)) api_data_t;

api_data_t api_data = {
     .api_id=API_ID,
     .garbage_can_id=GARBAGE_CAN_ID,
  };
static struct uip_udp_conn *client_conn;

// ********** my functions **********

void twi_error(uint8_t cnt, uint8_t exp_status){

  volatile uint8_t status = 0;
  uint8_t cnt2 = 0;

  PRINTF("\n%d.␣ERROR:\n", cnt);
  for(cnt2 = 0; cnt2<2; cnt2++)
  {
    if(cnt==0){
      PRINTF("Expeted␣Status:␣");
      status=exp_status;
    }
    else{
      PRINTF("\nReceived␣Status:␣");
      status=TWI_CURRENT_STATUS();
    }
    switch (status)
    {
      case TWI_START_STATUS:
        PRINTF("TWI_START_STATUS");
        break;
      case TWI_REPSTART_STATUS:
        PRINTF("TWI_REPSTART_STATUS");
        break;
      case TWI_SLAW_ACK_STATUS:
        PRINTF("TWI_SLAW_ACK_STATUS");
        break;
      case TWI_SLAW_NACK_STATUS:
        PRINTF("TWI_SLAW_NACK_STATUS");
        break;
      case TWI_SLAR_ACK_STATUS:
  PRINTF("TWI_SLAR_ACK_STATUS");
        break;
      case TWI_SLAR_NACK_STATUS:
        PRINTF("TWI_SLAR_NACK_STATUS");
```

```
          break;
        case TWI_TXDATA_ACK_STATUS:
          PRINTF("TWI_TXDATA_ACK_STATUS");
          break;
        case TWI_TXDATA_NACK_STATUS:
          PRINTF("TWI_TXDATA_NACK_STATUS");
          break;
        case TWI_RXDATA_ACK_STATUS:
          PRINTF("TWI_RXDATA_ACK_STATUS");
          break;
        case TWI_RXDATA_NACK_STATUS:
          PRINTF("TWI_RXDATA_NACK_STATUS");
          break;
        case TWI_ARBLOST_STATUS:
          PRINTF("TWI_ARBLOST_STATUS");
          break;
        default:
          PRINTF("0x%X",status);
          break;
      }
    }
    PRINTF("\n");
    _delay_ms(100);
}

put_block(char *p, char length)
{
    char cnt = 0;
    for(cnt; cnt<length; cnt++){
      putchar(*p);
      p++;
    }
}
/*---------------------------------------------------------------------------*/
PROCESS(udp_client_process, "UDP␣client␣process");
AUTOSTART_PROCESSES(&udp_client_process);
/*---------------------------------------------------------------------------*/
static void
tcpip_handler(void)
{
 char *str;
 if(uip_newdata()) {
str = uip_appdata;
    str[uip_datalen()] = '\0';
printf("Response␣from␣the␣server:␣'%s'\n", str);
 }
}
/*---------------------------------------------------------------------------*/
static void
```

```
timeout_handler(void)
{
 static int seq_id;
 char buf[MAX_PAYLOAD_LEN];


 printf("Client␣sending␣to:␣");
 PRINT6ADDR(&client_conn->ripaddr);
 //sprintf(buf, "Hello %d from the client", ++seq_id);
 memcpy(buf, (uint8_t*)&api_data.api_id,8);

 printf("(msg:");
 put_block(buf, 8);
 printf(")\n");
 uip_udp_packet_send(client_conn, buf, 8);
}
/*---------------------------------------------------------------*/
static void
print_local_addresses(void)
{
 int i;
 uint8_t state;

 PRINTF("Client␣IPv6␣addresses:␣");
 for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
   state = uip_ds6_if.addr_list[i].state;
   if(uip_ds6_if.addr_list[i].isused &&
      (state == ADDR_TENTATIVE || state == ADDR_PREFERRED)) {
    PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
    PRINTF("\n");
   }
 }
}
/*---------------------------------------------------------------*/
#if UIP_CONF_ROUTER
static void
set_global_address(void)
{
 uip_ipaddr_t ipaddr;

 uip_ip6addr(&ipaddr, 0xbbbb, 0, 0, 0, 0, 0, 0, 0);
 uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
 uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
}
#endif /* UIP_CONF_ROUTER */
/*---------------------------------------------------------------*/
static void
set_connection_address(uip_ipaddr_t *ipaddr)
{
```

```
#define _QUOTEME(x) #x
#define QUOTEME(x) _QUOTEME(x)
#ifdef UDP_CONNECTION_ADDR
 if(uiplib_ipaddrconv(QUOTEME(UDP_CONNECTION_ADDR), ipaddr) == 0)
     {
  PRINTF("UDP␣client␣failed␣to␣parse␣address␣'%s'\n", QUOTEME(
     UDP_CONNECTION_ADDR));
 }
#elif UIP_CONF_ROUTER
//aaaa::212:7404:4:404
 uip_ip6addr(ipaddr,0xbbbb,0,0,0,0,0,0,2);
#else
 uip_ip6addr(ipaddr,0xfe80,0,0,0,0x6466,0x6666,0x6666,0x6666);
#endif /* UDP_CONNECTION_ADDR */
}
/
/*---------------------------------------------------------------*/
PROCESS_THREAD(udp_client_process, ev, data)
{
 static struct etimer et;

 uint8_t rbyte = 0;
  uint8_t hbyte = 0;
  uint8_t lbyte = 0;
  uint8_t cnt = 0;
  uint8_t error = 0;
  uint16_t distance = 0;
  uint16_t mindistance = 0;
 uip_ipaddr_t ipaddr;

 PROCESS_BEGIN();
 PRINTF("UDP␣client␣process␣started\n");

#if UIP_CONF_ROUTER
 set_global_address();
#endif

 print_local_addresses();

 set_connection_address(&ipaddr);

 /* new connection with remote host */
 client_conn = udp_new(&ipaddr, UIP_HTONS(5688), NULL);
 udp_bind(client_conn, UIP_HTONS(8775));

 PRINTF("Created␣a␣connection␣with␣the␣server␣");
 PRINT6ADDR(&client_conn->ripaddr);
 PRINTF("␣local/remote␣port␣%u/%u\n",
UIP_HTONS(client_conn->lport),
UIP_HTONS(client_conn->rport));
   PRINTF("twi_init␣=␣%d\n", twi_init(F_CPU, 100000));
   PRINTF("TWBR␣=␣%d\n", TWBR);
   PRINTF("TWSR␣=␣0x%X\n", TWSR);
  etimer_set(&et, SEND_INTERVAL);
  while(1) {
   PROCESS_YIELD();

   for(;;){
        cnt = 0;

        cnt++; if(twi_start()!=TWI_START_STATUS){twi_error(cnt,
            TWI_START_STATUS);continue;}
        cnt++; if(twi_wbyte(SRF02_WRITE_ADDRESS)!=
            TWI_SLAW_ACK_STATUS){twi_error(cnt,TWI_SLAW_ACK_STATUS)
            ;continue;}
        cnt++; if(twi_wbyte(0x04)!=TWI_TXDATA_ACK_STATUS){twi_error
            (cnt,TWI_TXDATA_ACK_STATUS);continue;}
        cnt++; if(twi_start()!=TWI_REPSTART_STATUS){twi_error(cnt,
            TWI_REPSTART_STATUS);continue;}
        cnt++; if(twi_wbyte(SRF02_READ_ADDRESS)!=
            TWI_SLAR_ACK_STATUS){twi_error(cnt,TWI_SLAR_ACK_STATUS)
            ;continue;}
        cnt++; if(twi_rbyte(&hbyte, 1)!=TWI_RXDATA_ACK_STATUS){
            twi_error(cnt,TWI_RXDATA_ACK_STATUS);continue;}
        cnt++; if(twi_rbyte(&lbyte, 0)!=TWI_RXDATA_NACK_STATUS){
            twi_error(cnt,TWI_RXDATA_NACK_STATUS);continue;}
        TWI_STOP_SET();
        mindistance = (hbyte<<8)|lbyte;
#if DEBUG_MEASUREMENT_RESULTS
        PRINTF("minimum␣distance:␣%d␣cm\n", mindistance);
#endif

        cnt++; if(twi_start()!=TWI_START_STATUS){twi_error(cnt,
            TWI_START_STATUS);continue;}
        cnt++; if(twi_wbyte(SRF02_WRITE_ADDRESS)!=
            TWI_SLAW_ACK_STATUS){twi_error(cnt,TWI_SLAW_ACK_STATUS)
            ;continue;}
        cnt++; if(twi_wbyte(0x00)!=TWI_TXDATA_ACK_STATUS){twi_error
            (cnt,TWI_TXDATA_ACK_STATUS);continue;}
        cnt++; if(twi_wbyte(SRF02_CM_MEASUREMENT)!=
            TWI_TXDATA_ACK_STATUS){twi_error(cnt,
            TWI_TXDATA_ACK_STATUS);continue;}
        TWI_STOP_SET();

        _delay_ms(100);
        cnt++; if(twi_start()!=TWI_START_STATUS){twi_error(cnt,
            TWI_START_STATUS);continue;}
```

```
      cnt++; if(twi_wbyte(SRF02_WRITE_ADDRESS)!=
          TWI_SLAW_ACK_STATUS){twi_error(cnt,TWI_SLAW_ACK_STATUS)
          ;continue;}
      cnt++;
      cnt++; if(twi_start()!=TWI_START_STATUS){twi_error(cnt,
          TWI_START_STATUS);continue;}
      cnt++; if(twi_wbyte(SRF02_WRITE_ADDRESS)!=
          TWI_SLAW_ACK_STATUS){twi_error(cnt,TWI_SLAW_ACK_STATUS)
          ;continue;}
      cnt++; if(twi_wbyte(0x02)!=TWI_TXDATA_ACK_STATUS){twi_error
          (cnt,TWI_TXDATA_ACK_STATUS);continue;}
  if(etimer_expired(&et)) {
   timeout_handler();
   etimer_restart(&et);
  } else if(ev == tcpip_event) {
tcpip_handler();
  }
 }

 PROCESS_END();
}
```

Bibliography

[1] Frequentznutzungplan, *Bundesnetzadgentur*. April 2008.

[2] Performance Analysis of Large-Scale Wireless Sensor Network Architecture with Multi-Cluster Configuration, *M. Sugano, Y. Kiri and M. Murata*. April 2008.

[3] A Performance Comparison of Different Topologies for Wireless Sensor Networks, *Shrestha; Liudong Xing*. University of Massachusetts Dartmouth, 2007

[4] Ultra-Low Energy Wireless Sensor Networks in Practice, *M. Kuorilehto, M. Kohvakka,J. Suhonen, Panu, Marko and Timo Hämäläinen*. Tampere University of Technology, Finland

[5] Wireless Sensor Networks-Architectures and Protocols, *Edgar H. Callaway, Jr.*. Auerbach Publications; August 2003

[6] Power Aware Wireless Microsensor Systems, *Chandrakasan, Min, Bhardwaj, S. Cho, and Wang*. Florence, Italy, September 2002

[7] IEEE 802.15.4 Standard for Information technology, *(revision of IEEE 802.15.4-2003)*. 2006

[8] Analysis of the IEEE 802.15.4a Ultra Wide Band Physical Layer through wireless sensor network simulations in Omnet++, *M. Alberts*. 2011-04-06

[9] ZigBee and Zigbee RF4CE Standard 09-5231, *Zigbee Alliance*. 2009

[10] ZigBee Cluster Library Specification, *Zigbee Alliance*. May 2008

[11] Performance Analysis of IEEE 802.15.4 and ZigBee for Large-Scale Wireless Sensor Network Application, *M. Kuorilehto, M. Kohvakka, T. Hämäläinen*. Tampere University of Technology, Finland, 2006

[12] 6LoWPAN: The Wireless Embedded Internet, *Z. Shelby; C. Bormann.* Wiley, 2009

[13] 6LoWPAN implementation, The uIP TCP/IP stack, *(http://www.sics.se/ adam/contiki/docs-uipv6/a01109.html/pages.html).* Contiki Docs, Oct 2008

[14] Zigbee vs Z-Wave, *(http://mobiledevdesign.com/hardware_news/zigbee_zwave_battle_1008).* Mobiledevdesign magazine, 2007

[15] Bluetooth Core V4.0 Specification, *Bluetooth Special Interest Group.* 2010

[16] Extending the Near Field Communication Market Opportunity with DASH7 Wireless Sensor Networking Technology, *JP Norair, Pat Burns.* 2010

[17] Contiki 2.x Reference Manual, *JA Dunkels.* 2007

[18] Rime, A Lightweight Layered Communication Stack for Sensor Networks, *A. Dunkels.* Swedish Institute of Computer Science

[19] ContikiRPL and TinyRPL: Happy Together, *Ko , J. Eriksson , N. Tsiftes , S. Haggerty ,A. Terzis , A. Dunkels and D. Culler.* Apr 2011

[20] Rime, A Lightweight Layered Communication Stack for Sensor Networks, *A. Dunkels.* Jan 2007

[21] Low-Power Wireless IPv6 Routing with ContikiRPL, *Tsiftes, J. Eriksson, A. Dunkels* . Stockholm, Sweden, April 2010.

[22] Interconnecting Smart Objects with IP, The Next Internet, *ean-Philippe Vasseur , Adam Dunkels.* 2010

[23] RPL: IPv6 Routing Protocol for Low Power and Lossy Networks, *draft-ietf-roll-rpl-19.* 2011

[24] Atmel AVR ATmega1281/V, 8-bit Atmel Microcontroller with 64K/128K/256K Bytes In-System Programmable Flash, *Atmel Corporation Datasheet.* 2011

[25] ICradio Module 2.4G Datasheet, *In-Circuit GmbH - http://www.ic-board.de/data/datasheet/ICradio-Module24G_(english).pdf.* Dresden 2009