Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Semester Thesis
at the Department of Information Technology
and Electrical Engineering

IPv6 for Wireless Sensor Networks

SS 2009

Lars Schor

Advisors: Philipp Sommer
Roland Flury
Professor: Prof. Dr. Roger Wattenhofer

Zurich
19th June 2009

Abstract

The Internet of Things predicts a world in which each thing is connected to the Internet. Integration of a wireless sensor in each thing is one approach to realize this idea. The 6LoWPAN standard proposes a solution to use the Internet Protocol on sensor nodes and to connect low-power sensor devices with the Internet.

In this semester thesis, 6LoWPAN is implemented on the newest generation of low-power sensor nodes. For this purpose, an already existing 6LoWPAN implementation for TinyOS is ported to the new platforms.

In the vision of Web of Things, each device can be accessed through the Web. A RESTful API for sensor nodes is developed that allows a computer to access the sensor data of a low-power hardware device using the standard Web protocol. Based on a mashup application, the possibilities of the Web of Things are demonstrated.

Acknowledgements

First of all I would like to express my sincere gratitude to Prof. Dr. Roger Wattenhofer for giving me the opportunity to write this semester thesis in his research group.

I would also like to thank my advisors Philipp Sommer and Roland Flury for their constant support during this semester thesis. They always helped me to solve the difficult problems of this thesis. Without their assistance, this work would never have been possible.

Furthermore, I should also like to extend special thanks to Prof. Dr. Miklos Maroti from the University of Szeged (Hungary) for his support during the development of the modified driver for the RF2xx transceiver.

My special thanks go to my family and my girlfriend for supporting and motivating me during this thesis.

Contents

List of Figures

1 Introduction

1.1 Motivation

In an automated home, all objects can be centrally controlled. The baking oven starts 20 minutes before you get home, the fridge orders the missing yoghurts in the Internet in order that each day starts with your favorite breakfast. The power consumption is minimized by centrally monitoring the standby power and automatic disabling unused devices. The Internet of Things, a vision in which each device is connected with the Internet, will completely change our lives.

So far, in the absence of a single standard for the Internet of Things, gateways are used to convert the commands of a user-friendly interface to a language each single device understands. The Web of Things defines the missing standard and proposes a solution to address each device directly over the Web.

The newest generation of sensor nodes offers enough performance to realize the idea of the Web of Things with minimal energy consumption. Nowadays, Wireless Sensor Networks (WSNs) employ proprietary communication protocols, making it therefore difficult to connect the nodes to the Internet. The recently launched IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) standard proposes a solution to use the Internet Protocol (IP) on sensor nodes and to integrate these low-power devices into the Internet. Unfortunately, no implementation of 6LoWPAN exists for the newest generation of sensor nodes.

Current realizations of the Web of Things exist only for high performance sensor nodes using an advanced gateway to translate the Web data to sensor node readable data frames. Standard networking tools like ping or further protocols cannot be used unless the gateway offers these specific functionalities.

This semester thesis combines the solutions described above. To support the Internet of Things, 6LoWPAN is implemented on the newest generation of energy-saving sensor nodes having only eight Kbytes of RAM. Based on this IP stack, the sensor nodes offer Web services, which can be accessed over the Web.

1.2 Contributions

- An existing 6LoWPAN implementation for TinyOS is ported to the newest generation of energy-saving sensor nodes in particular the MeshBean900, the Pixie and the Crossbow IRIS. The performance and advantages of the new sensor nodes are discussed with regard to the 6LoWPAN implementation.
- A RESTful application-programming interface (API) is developed for sensor nodes. To present the data offered by the sensor nodes in a user-friendly way, we have programmed a Web application assuming this functionality.
- In order to point out the various possibilities that will be enabled by the Web of Things, a mashup application has been implemented. It allows a simple and graphical monitoring of a WSN.

1.3 The Internet of Things

In the vision of Internet of Things, each thing is connected to a network, namely the Internet [1]. A thing might be any object of the daily life as shoes or cars. The vision does not only assume that the fridge is connected to the network, but also all yoghurts and all other things inside the fridge are part of the Internet. Already realized applications of Internet of Things are for example household appliances or wearable computing.

Communication is the key element in the Internet of Things. The goal is to integrate everyday physical objects into communication networks: “from anytime, any place connectivity for anyone, we will now have connectivity for anything” [1].

From the technology point of view, a simple and cost-efficient system to connect things to the network is required. The simplest devices which offer these functionalities are radio-frequency identification (RFID) devices. Nevertheless, as they only provide a limited functionality, other platforms are needed to connect all things with the Internet. Another approach is to use mobile phones, which have high power consumption and thus only have a short lifetime without human intervention. Sensor nodes and actuators are often mentioned in this area. They provide various functionalities and low power consumption, thus ideally adjusted to the requirements of the Internet of Things.

1.4 The Web of Things

While the vision of Internet of Things addresses the way of connecting the devices, the Web of Things community considers the integration of the things into the Web [2]. Each thing should be available using standard Web mechanisms by supporting protocols like Hypertext Transfer Protocol (HTTP), Representational State Transfer (REST) or Simple Object Access Protocol (SOAP). This thesis will propose a possible Web of Things implementation for sensor nodes. For this purpose, a RESTful interface for sensor nodes has been developed.

1.5 Related Work

Since the introduction of IEEE 802.15.4 and its establishment as link-layer protocol for sensor nodes, many communication protocols have been designed with respect to this new standard. Examples include the ZigBee standard [3] or WirelessHART [4]. The first one is an open protocol to connect small devices like sensor nodes. It is based on the OSI layer and proposes a standard for all layers above the MAC layer. Since its introduction in 2005, many products have been developed for ZigBee. WirelessHART is another communication standard based on IEEE 802.15.4, but its mainly fields of application are industrial purposes. For instance, a possible application of WirelessHART is measuring instruments. The variety of these standards is also a crucial factor in this area. Both protocols are stand-alone standards, thus do not support communication with other networks without using an advanced gateway. A similar approach is also necessary if a node of such a network should be connected to the Internet. This thesis presents a network layer protocol, which is based on IP. This simplifies the development of an advanced gateway as only a tunnel is needed.

The idea of connecting sensor nodes with the Internet was first discussed in

the late nineties (amongst others [5]). Nevertheless, it was concluded that networks of sensor nodes have other requirements than the ones of the Internet and a direct connection of sensor nodes with the Internet is neither meaningful nor easily realizable. When the research about the Internet of Things started in the early 20th century [1, 6], RFID devices were mainly focused primarily by the community. Nevertheless, a first implementation of an Internet Protocol (IP) stack for sensor nodes was programmed for Contiki in 2003 [7]. The stack is based on IPv4 and triggered a small revolution in the sensor nodes community. A rethinking has been established and a common standard for IPv6 on WSNs has been worked out [8] (see section 3.1). Shortly afterwards, first possible implementations of 6LoWPAN have been developed for TinyOS (amongst others [9]). The newest and most extensive implementation of 6LoWPAN for TinyOS is currently being developed at the University of California, Berkeley and is called Berkeley IP Information for low-power networks (blip) [10]. It supports almost any functionalities of 6LoWPAN and is presented in detail in section 3.2. During this project, we have ported blip to our sensor node platform, especially to a new RF transceiver and used blip as IP stack in our implementations. Several applications based on blip are currently under development. An AC monitor [11] is one of the first projects using blip. Nodes send the currently measured AC power of an external device to a Web server using the User Datagram Protocol (UDP). In contrast to our project, no approaches to use the Transmission Control Protocol (TCP) or to react on requests of the Internet are integrated in the AC monitor.

First approaches to integrate Web services in WSNs were done in 2007 and one agreed about the fact that Representational State Transfer (REST) is the most suitable protocol to implement Web services on sensor nodes. Stribu [2] described a possible implementation of Web services on sensor nodes using the REST protocol to communicate with the world, however only from a theoretical point of view. The connection of the sensor nodes to the Internet should mainly be realized through service gateways. A first real implementation of Web applications on sensor nodes has been presented for Contiki [12]. A simple Web server with a few pages has been implemented on sensor nodes using the mentioned Contiki IP stack.

The ETH Zurich has developed in collaboration with SAP Research Zurich during the last few months a first concrete approach of the Web of Things vision [13, 14]. Based on REST and JavaScript Object Notation (JSON), a sensor node delivers its data on request to the sender. The implementation platform for this project has been Sun SPOT devices and the communication with the Internet occurs using a gateway as the Sun SPOT does not support the IP stack. Although the first paper has been published in a later point of this thesis, several ideas of this paper influenced this thesis like the idea of a Web front-end for easily observing the Web service responses. There

are two big differences between the implementation of their project and the one used for this thesis. Our target platform, the MeshBean900, is much less powerful than a Sun SPOT and uses TinyOS as operating system (OS) while the Sun SPOT is built on the Squawk Java Virtual Machine. Even though both projects are based on the same ideas, we implemented different functionalities in our application like recording or mashups in combination with a world map.

ArchRock presented in a tutorial at the IPSN 2009 a workable demonstration of an HTTP layer for TinyOS. They implemented a Web server on a sensor node, but did not provide any Web services. As only the interfaces are publicly available, but not the concrete implementation of the HTTP layer, a comparison with this project is not meaningful.

1.6 Outline

The remaining part of this semester thesis is organized in three chapters. Chapter 2 presents the background technologies used in this project. First, the target platform is shortly introduced. Nowadays, the IEEE 802.15.4 protocol is the classic link-layer protocol used for such platforms and is introduced in the second part of the chapter. An overview about IPv6, the future IP and a discussion about its advantages compared to some conventional network protocols uses in WSNs are given in the last part of the chapter.

Chapter 3 discusses a proposed standard for IPv6 on sensor networks. After introducing 6LoWPAN in the first part of the chapter, an implementation of this standard in TinyOS and its porting to our target platform are presented in the second part. A performance comparison between the original implementation and the ported one concludes the chapter.

The Web of Things is the subject of chapter 4. A short introduction to the basic technologies used for Web services is given at the beginning. The chapter continues with the presentation of a concrete implementation of Web services on sensor nodes. Finally, some performance evaluations of our application are discussed.

2 Background

A first goal of this project is to use the Internet Protocol (IP) layer to provide a common network layer for sensor nodes. This network layer should allow a direct connection between these nodes and the Internet. Nowadays, the Internet undergoes a huge change because the standard IP protocol Internet Protocol Version 4 (IPv4) is being upgraded to the Internet Protocol Version 6 (IPv6). Considering the functionalities and properties of both protocols, it is quite evident that IPv6 is much better suited for WSNs than IPv4, even if the adaptation of IPv6 to the common link layer protocol IEEE 802.15.4 comes up with many challenges and does not seem to be straightforward.

Before considering a potential implementation of IPv6 on WSN in chapter 3, this chapter presents all background technologies and the hardware used in this thesis. A short introduction to TinyOS, the operating system (OS) of the sensor nodes is given in section 2.1. Nodes based on the RF transceiver RF212 are used as target platform for the implementation. Therefore, a technical overview about these sensor nodes is given in section 2.2 while an upcoming link layer standard for sensor nodes, the IEEE 802.15.4 protocol is presented in section 2.3. Afterwards, this chapter changes the focus to the network layer and IPv6 is introduced in section 2.4. After introducing all technologies, the last section answers the question why one should use IPv6 for WSN.

2.1 TinyOS

There exists several OS for WSNs. TinyOS [15] is one of the most widely used and offers a broad variety of functionalities for sensor nodes. In this section, the architecture of TinyOS is discussed and an overview about the features already implemented in TinyOS that may be helpful for this project is provided.

Programmed in nesC, TinyOS features a component-based architecture for developing applications for sensor nodes. It considers the memory and computation constraints of such small devices as its core OS uses only 400 bytes of memory [16]. Support for low-power operations and communication over IEEE 802.15.4 are additional features that make TinyOS suitable for this project.

TinyOS is a layered OS with interfaces and components. Each component provides a set of services that are specified by interfaces. By connecting these components together, an application defines its features. As the compiler only integrates the wired components into the binary, small and efficient applications are generated. Two components communicate with each other through interfaces. For this purpose, a component defines the interfaces it provides and those it uses. It can only use functionalities of the interfaces it uses and has to implement the features of those interfaces it provides. The former are called commands while the later are denoted as events. One differs between two types of components in TinyOS, the modules and the configurations. By connecting interfaces used by components with interfaces provided by other components, a configuration wires the components together. Modules are used to implement commands and events. The effective program code is therefore written in modules.

This thesis uses the latest stable release of TinyOS, version 2.1.0. TinyOS has already been ported to the sensor nodes used in this project. Furthermore, there exists an IEEE 802.15.4 communication stack for the radio chip RF212, which can be used by the sensor nodes.

2.2 Hardware

Two target platforms are considered for this project, the MeshBean900 and the Pixie. Both platforms are based on the ZigBit900 [17] module from Atmel, which contains the ATmega1281V Microcontroller and the AT86RF212 Transceiver from Atmel. After introducing this radio transceiver in subsection 2.2.1, an overview about the two platforms is given in subsection 2.2.2.

2.2.1 Radio Module RF212

The radio module RF212 from Atmel [18] is an RF transceiver optimized for IEEE 802.15.4. It is integrated in the 800/900 MHz-Band and fulfills most requirements of the IEEE 802.15.4-2006 standard like binary quadrature phase shift keying (BPSK) and offset quadrature phase shift keying (QPSK). The former allows transmitting with data rates of 20 and 40 Kbit/s while the later can operate with rates of 100 Kbit/s and 250 Kbit/s. The radio chip has a low current consumption and integrates an AES 128-bit hardware accelerator and a MAC hardware accelerator.

During the project, we realized that the IPv6 implementation mainly depends on the radio transceiver. As the RF212 transceiver is part of a product family, one has decided to program the IPv6 implementation for the whole RF2xx transceiver family to support also other platforms. One of them is the Crossbow IRIS, which is based on the Atmel AT86RF230 radio module. Its main difference to the RF212 is that it operates in the 2.4 GHz band, thus supporting higher data rates. Additionally, the new Ateml AT86RF231 transceiver can easily be supported as soon as it is available in TinyOS.

2.2.2 MeshBean900 / Pixie

Both the MeshBean900 and the Pixie platform are based on the ZigBit900 wireless module. The latter uses an ATmega1281 microcontroller that features 128 Kbytes flash memory and 8 Kbytes RAM. The MeshBean900 is a development board produced by MeshNetics, which already implements a temperature and illumination sensor.

While the MeshBean900 platform is commercial, the Pixie platform is developed by the Computer Engineering and Network Laboratory (TIK) at the ETH Zurich and is not publicly available. Unlike the MeshBean900 platform, which provides many potentially unused functionalities, the Pixie platform only implements the most fundamental features. No sensors are integrated by default and its on-board radio antenna is much weaker than the one on the MeshBean900 board.

2.3 Communication for LR-WPANs: IEEE 802.15.4

As low-rate wireless personal area networks (LR-WPANs) have different requirements than a classic wireless network, a new physical and link layer standard for communication in LR-WPANs has been introduced by the IEEE Computer Society. The IEEE 802.15.4 protocol defines a physical and media access control (MAC) layer for such networks. Most new sensor nodes sup-

port this protocol by implementing hardware accelerators. The ZigBit900 wireless module used for our project supports IEEE 802.15.4-2006. The standard is defined by the IEEE Computer Society in [19].

The physical and MAC layer defined in the IEEE 802.15.4 standard are summarized in subsection 2.3.1 while an overview about the data and acknowledgement frame is given in subsection 2.3.2. Furthermore, IEEE 802.15.4 defines much more elements like network models, coordination and security for LR-WPANs. As they are of low relevance for this project, they are not considered in this thesis. The encouraged reader may refer to [19] for further information.

2.3.1 The Physical and MAC Layer

The physical layer (PHY) has the following tasks: management of the physical RF transceiver, channel selection as well as energy and signal management. For this purpose, PHY provides the data and physical layer management service. The radio transceiver can operate either in the 868/915 MHz band or in the 2450 MHz band. The 868 MHz band is only for Europe, the 915 MHz band only for North America and the 2450 MHz band for worldwide use. In the 868/915 MHz band, one can use either BPSK or QPSK while the 2450 MHz band is restricted to QPSK. This allows data rates up to 250 Kbit/s.

The transmission of MAC frames across the PHY layer is the main purpose of the MAC layer. Its further tasks are the channel access, the frame validation and the acknowledged frame delivery. Each node has either a short 16-bit or 64-bit extended address that is used as address on the MAC layer.

2.3.2 Data and Acknowledgement Frame

The implementation of the frames results in the biggest restriction for transmitting IPv6 data packets. The IEEE 802.15.4 protocol differs between four types of frames: the beacon buffer, the data frame, the acknowledgement frame and the MAC command frame. The maximum size of the PHY payload, i.e. the MAC packet, is 127 bytes. The beacon buffer is used by a coordinator to transmit beacons and the MAC command frames have the functionality to handle all MAC peer entity control transfers. Both are of low relevance for the higher-level protocol and are fully explained in [19].

The data frame is outlined in Fig. 2.1. This frame is used for all data transfers. The first two fields of the MAC Header (MHR) are the Frame Control and the Data Sequence Number (DSN). Furthermore, it consists of the Addressing Field that may contain the PAN identifier of the source and

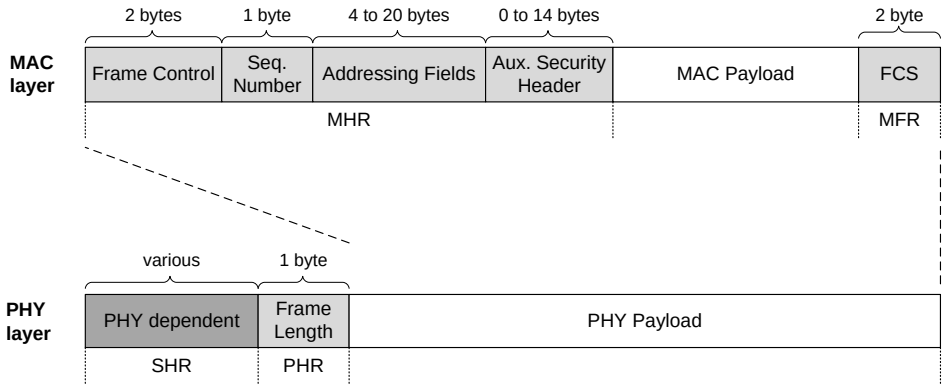


Figure 2.1: The 802.15.4 data frame subdivided into the PHY and MAC layer.

the target node and both node addresses. Using PAN identifiers (two bytes per identifier) and the extended addresses, the address field has a length of 20 bytes. Nevertheless, if one may only use both short 16-bit addresses, the length of the address field will be reduced to four bytes. An Auxiliary Security Header might be the last field of the MHR. The MAC Footer (MFR) contains only the Frame Check Sequence (FCS). The MAC packet is passed to the PHY layer, which adds a PHY Header (PHR) and a Synchronization Header (SHR). The PHR consists only of the Frame Length that contains the length of the PHY payload in octets. Considering the worst case, i.e. if all headers use their maximum size, the size of the data payload is only 88 bytes.

Fig. 2.2 outlines the acknowledgement frame that is used for confirming successful frame reception. The PHY layer is similar to the Data Frame while the MAC layer is reduced. An acknowledgement frame does not contain any payload and the MHR consists only of the MAC Frame Control field and the DSN. The MFR contains only the FCS field similar to the data frame.

2.4 Internet Protocol Version 6

IPv6 is the successor of the current Internet Protocol IPv4 and addresses most limitations of IPv4. The biggest modification of the further IP is its large address space of 2^{128} addresses that corresponds to around $5 \cdot 10^{28}$ IPv6 addresses for each person alive. As one has addresses in abundance, IPv6 enables the basic idea of the Internet of Things community, i.e. to configure each object with an IP address.

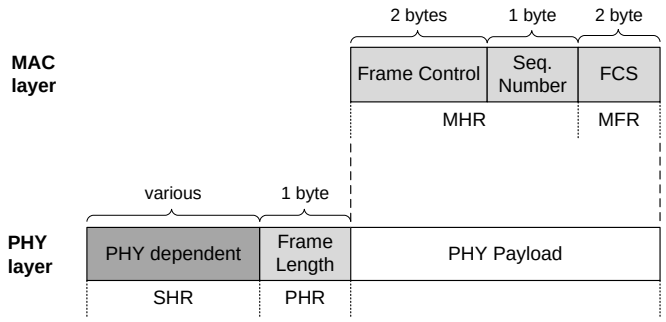


Figure 2.2: The 802.15.4 acknowledgement frame subdivided into the PHY and MAC layer.

IPv6, a layer 3 best-effort transport protocol that is defined in RFC2460 [20], is described in this section. In particular, the most important points concerning an IPv6 implementation on sensor nodes are emphasized. The advantages of IPv6 compared to IPv4 are outlined in subsection 2.4.1, an overview about the addressing format used in IPv6 is given in subsection 2.4.2 while the header format of IPv6 packets is described in subsection 2.4.3. The last subsection points out the most important protocols, which run over IPv6.

2.4.1 The New Internet Protocol

IPv4 is a widely accepted IP and is used for any connections with the Internet. The protocol is implemented in most network components and almost any communication software is based on IPv4. In this subsection, the question why it is necessary to switch to a new IP, which requires an upgrade of all software and most hardware components, is discussed.

The current problems of IPv4 are known since several years. In 1992, the Internet Engineering Task Force (IETF) issued a call for proposals for the next IP [21]. Three years later, the first RFCs were published, which define the recommendations, the address allocation, the specifications and the architecture (RFC 1752, RFC 1881, RFC 1883 and RFC 1884) of a future IP.

According to RFC 1752 [22], the main reason for introducing a new IP was "that the Internet address space would become an increasingly limiting resource". IPv4 uses 32-bit addresses, which enables to assign 2^{32} addresses. This corresponds to around four billion addresses. Nevertheless, mainly two reasons prevent the use of all these addresses:

- As an IP address consists of a network number and a host number, not all allocated addresses are really used. Consider for example a network, which allocates 64 IP addresses but only consists of seven real devices. Only nine addresses (the router, all devices and the broadcast address) are really used and 55 addresses or 86% of the addresses are not used but allocated.
- Many addresses are used for specific functionalities like multi-casting and cannot be allocated by users.

Stallings [21] points out additional requirements for a new IP. The header and fragmentation principle used in IPv4 does not address performance requirements as IPv4 has many fields in a header of variable length. IPv6 introduces a fixed-length header and does not allow routers to fragment packets. Furthermore, IPv4 does not support network services and does not provide any security capabilities on the IP layer. An interesting remark is that these disadvantages of IPv4 have been considered in the early nineties when most private households and companies were not yet connected to the Internet.

2.4.2 Addressing

As mentioned before the biggest drawback of IPv4 is its small address space. In this subsection, the new address concept described in RFC 4291 [23] is introduced. The representation format of an IPv6 address, the different address types and the address configuration principles used in IPv6 are each described in the following.

Addressing Model and Representation

An IPv6 address has a length of 128 bits. According to [23] the preferred form of an IPv6 address is `x:x:x:x:x:x:x:x` where each `x` represents one to four hexadecimal digits and is called a group. An example for a well-formatted address is `ABCD:EF01:2345:6789:ABCD:EF01:2345:6789`.

There exist several rules to simplify the address. The first rule concerns the leading zeros in a group: any leading zeros in a group can be omitted. Furthermore, it is possible to replace one or any number of consecutive groups of zeros with two colons. So the address `FF01:0:0:0:0:0:0:101` may be written as `FF01::101`. Note that this substitution is only allowed once in an address.

Address prefixes are handled in IPv6 similar to IPv4. The prefix is represented as `ipv6-address/prefix-length` with `ipv6-address` any valid notation

Table 2.1: Address types of IPv6 according to RFC 4291 [23].

Address type	Binary prefix	IPv6 notation
Unspecified	00...0 (128 bits)	::/128
Loopback	00...1 (128 bits)	::1/128
Multicast	11111111	FF00::/8
Link-Local Unicast	1111111010	FE80::/10
Global Unicast	(everything else)	

of an IPv6 address as described before and `prefix-length` a decimal value, which specifies how many bits comprise the address.

One more remark to the URL notation of an IPv6 address as it is used in Internet Browsers: The address is written in brackets such as

```
http://[ABCD:EF01:2345:6789:ABCD:EF01:2345:6789]
```

to not mistake the last group as the transport protocol port.

Address Types

RFC 4291 differs between several types of IPv6 address classes, which are described in the following. Additionally Table 2.1 provides some examples of these address types. The type of an address can be identified by considering its higher-order bits.

Unicast A single network interface is identified by a unicast address. One differs between

- Global Unicast,
- Site-Local Unicast (which are now deprecated) and
- Link-Local Unicast.

Anycast The IPv6 anycast address has no comparable address in IPv4. The address is assigned to more than one interface. A packet sends to an anycast address is routed to the nearest interface with this address.

Multicast A multicast address identifies multiple nodes together, i.e. it is an identifier for a group of interfaces. If one sends a packet to a multicast address, the message is delivered to all interfaces identified by that address.

Furthermore, each address type can be divided into subgroups. For this project, the following multicast addresses are relevant:

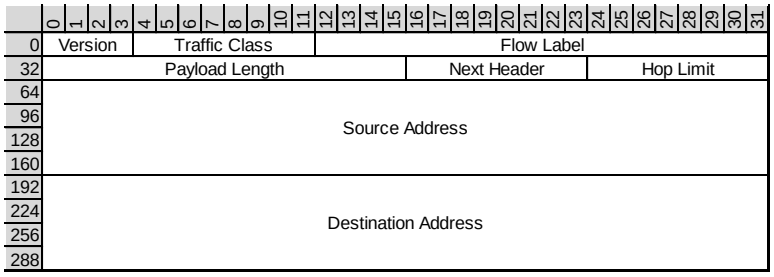


Figure 2.3: IPv6 Header Format according RFC2460 [20].

- `FF02::1` (link-local all-nodes address), which addresses all nodes on the link and
- `FF02::2` (link-local all-routers address), which addresses all routers on the link.

Address Configuration

IPv6 supports two types of address configuration: an updated version of the well-known Dynamic Host Configuration Protocol (DHCP) known as DHCPv6, which is defined in RFC 3315 [24] or a stateless address autoconfiguration mode defined in RFC 2462 [25]. DHCPv6 works similarly to DHCP for IPv4 and defines the stateful address configuration. For further information, one may refer to [24].

For sensor nodes, the newly introduced stateless autoconfiguration mode is much more interesting than DHCP. Using its layer-two MAC address, an IPv6 stack can generate itself a link-local address. The nodes can use this address together with the Neighbor Discovery Protocol (NDP, see subsection 2.4.4) to search for routers in its local network by sending messages to the multicast address `FF02::2`. Using the NDP, routers distribute information about the address range the node can use to assign itself a unicast address. The Duplicate Address Detection (DAD) mechanism prevents that the same address is assigned twice to different nodes.

2.4.3 Header Format

As already mentioned, a design goal of IPv6 was to simplify the header format. The new header format has to deal with the new addressing format and should be of fixed size. This subsection gives an introduction to the IPv6 header, which is described in RFC2460 [20]. Fig. 2.3 outlines the new header format while the individual fields are described in the following.

The first field contains the IP version number and is always set to the value six. The traffic class and the flow label fields are included to support quality of services. The first one sets the packet priority and the second one can be used for the Quality of Service (QoS) management but is currently unused. As the name implies, the payload length field contains the length of the IPv6 payload. The type of header, which follows immediately the IPv6 header, is identified by the next header field. Each node that forwards the packet decrements the hop limit field by one. The source and the destination addresses contain the according 128-bit IPv6 address.

To support optional Internet-layer information, IPv6 introduces the concept of extension headers. Each of these headers contains a next header field, which are used to link the headers together. Each extension header is optional and contains information, which supports the IPv6 transmission. There exist extension headers for various purposes like the routing, the fragmentation or the authentication header.

2.4.4 Support Protocols

To perform all operations described so far, IPv6 makes use of several other protocols. With an IPv6 implementation for sensor networks in mind, the two most important support protocols are the Internet Control Message Protocol for IPv6 (ICMPv6) and the NDP. In this subsection, both protocols are shortly introduced and its purpose in an IPv6 network is presented.

Internet Control Message Protocol for IPv6

ICMPv6 is described in RFC 4443 [26]. It is the successor of the well-known IPv4 protocol ICMP. The main purpose of ICMPv6 is to report errors encountered in processing packets. Furthermore, it is used for network diagnostics and other inter-layer functions. The well-known command `ping` is based on ICMP.

An ICMPv6 message consists of a message body and three header fields: the type, the code and the checksum. The first one is used to indicate the type of the message and to automatically determine the format of the remaining data. The message types Echo Request and Echo Reply are the most well known message types. A ping request is bases on these two message types.

Neighbor Discovery Protocol

The specifications of the NPD, which is newly introduced in IPv6, are described in RFC 4861 [27]. The goal of the protocol is to solve problems related

Table 2.2: Comparison between an IEEE 802.15.4 network (LoWPAN) and a typical IPv6 network.

	IEEE 802.15.4 Network	Typical IPv6 Network
Packet Size	Maximum payload of a physical layer packet: 127 bytes.	Maximum Transmission Unit (MTU): at least 1280 bytes.
Addressing	16-bit short or IEEE 64-bit extended MAC addresses.	128-bit IPv6 addresses.
Bandwidth	Typically 250 Kbps.	54 Mbps (802.11g) / 100 Mbps (Ethernet)
Power	Low, most devices run on battery.	No constraint, most devices are connected to a power network.
Reliableness	Connection is often unreliable.	Connection is often almost static.
Sleeping	Devices conserve energy by sleeping for long time.	Devices do not sleep and are always connected.

to the link layer. This includes the router and prefix discovery, the address autoconfiguration and resolution, the neighbor unreachability detection and the duplicate address detection.

To solve the above-described problems, five different ICMP packet messages are defined. The Router Solicitation message is used from a host that becomes enabled. It sends out these messages to request routers to generate router advertisement messages. The Router Advertisement message is used from a router to show their presence together with various link and Internet parameters. The message also contains the network prefix. The identification of the link-layer address of a neighbor node happens by analyzing the Neighbor Solicitation messages. The response to such a message is the Neighbor Advertisement message. The last ICMP packet type is the Redirect message, which can be used by routers to inform hosts about a better first hop for a destination.

2.4.5 Comparison of IPv6 and IEEE 802.15.4

The last section has given an introduction to the typical link-layer protocol for WSN while IPv6, a network layer protocol starting to be used in the biggest worldwide network, the Internet is described in this section. Before answering the question why one should use IPv6 for sensor nodes, an IEEE 802.15.4 network is compared with a typical IPv6 network in this subsection.

In Table 2.2, the most important differences between both networks are

summarized. The IEEE 802.15.4 protocol is optimized for networks with low-energy devices, while IPv6 should support high performance networks. The connections in a WSN are often unreliable, so IEEE 802.15.4 uses small packets and short addresses. This problem is almost unknown for IPv6 networks. To achieve high performance, such a network has to support long packets.

Nevertheless, in the next section, it is shown that IPv6 is better suited to WSN than all other network layer protocols irrespective of all these differences between a typical IEEE 802.15.4 network and an IPv6 network. The demands on a possible IPv6 implementation for WSNs follow directly from this comparison and are presented in subsection 3.1.1.

2.5 Connecting Sensor Nodes to the Internet

The newest approach of the WSN community is to connect their nodes directly to the Internet. The goal is to enable the ideas of the Internet of Things community and to standardize the different network protocols used in WSNs. The IEEE 802.15.4 protocol and IPv6 are introduced in the last few sections while in this one, the current approaches of introducing an IP on WSNs is discussed and the question why IPv6 is the right protocol for WSNs is answered.

At the beginning of the 21th century, the research community thought that WSNs have different requirements as the ones of the Internet and need a reconsidering of the overall structure of the services [5]. The assumption was that the layered architecture could not be used anymore because of the resource constraints. They thought that the required robustness and scalability could only be achieved by using localized algorithms. Furthermore, it was assumed that a WSN device might not need an identity, as the naming will be data-centric.

The idea was brought out by the fact that sensor nodes are connected with a serial interface to the outside [28]. For this purpose, an application level gateway might be installed at the root and the connection could be compared with USB. Nevertheless, application level gateways also provide many problems. Adam et al. [29] summarizes that “protocol gateways are inherently complex to design, manage, and deploy”. Adjustments to gateways are hard to manage, for each new functionality of the sensor node, the gateways have also to be updated.

Nowadays the IEEE 802.15.4 protocol is widely accepted as physical and MAC layer protocol for WSNs. A potential network layer protocol has to respect the constraints that results from the MAC layer protocol. As already mentioned, the properties of IP, especially of IPv6 do not fully match with

whose of the IEEE 802.15.4 protocol. Nevertheless, the Internet Protocol for Smart Object (IPSO) Alliance [30] proposes to use IP also for smart objects [29]. The support of a wide range of applications, the stability and the high scalability are its main reasons. The alliance argues that the development of a few lightweight IP stacks in the last few months have been given a proof-of-concept of IP on sensor nodes. The big advantages in the alliance's point of view is that sensor nodes can be accessed from anywhere and anything using IP and that IP is independent of the used physical and MAC layer protocol.

The above argumentation justifies using IP on sensor nodes but does not answer the question why one should use IPv6 on such devices. The answer is given by Hui et al. who claim that "IPv6 is better suited to the needs of WSNs than IPv4 in every dimension" [28]. They argue that IPv6 allows the implementation of more efficient network architectures than the current solutions and proving their theory with the following mechanisms of IPv6: sample listening, hop-by-hop feedback and collection routing. Because of the structure of the IPv6 address, a better compression is possible compared to IPv4 addresses. Their rationale is completed with the remark about the support protocols of IPv6 like autoconfiguration and that IPv6 implements features that are required in WSNs. The authors regard the address scalability, visibility and unattended operation as examples of these features.

The paper [28] proposes to connect WSN to other IP networks through one or more border routers, which have the function to forward IP datagrams between different media. This solves the main problem mentioned at the beginning of this section, i.e. the adjustment of the gateway after each sensor node modification. The border router forwards IP datagrams on the link-layer level. Once developed, the border router has never to be adjusted.

2.6 Summary

The MeshBean900 and the Pixie are new modern platforms for sensor nodes. Their RF212 transceiver supports the link-layer protocol IEEE 802.15.4. Nowadays, this protocol is used by almost any modern sensor node platform for communication. It is designed for short packets and unreliable networks.

IPv6, the future IP, is designed for huge networks. Using a large address space and stateless address autoconfiguration, it fulfills many requirements of the Internet of Things community. It can be shown that the use of IPv6 on low-power devices simplifies many network tasks in particular the connection of the low-power network with the Internet. As IPv6 has a large header and the MTU is at least 1280 bytes, header compression and fragmentation is necessary to implement IPv6 in WSNs.

3

IPv6 on Sensor Nodes: 6LoWPAN

As the last chapter showed, implementing IPv6 on sensor nodes simplifies the task of connecting a huge amount of nodes together to one big network and enables to realize the ideas of the Internet of Things community. Unfortunately, it is not possible to write a simple network layer with all nodes supporting IPv6. This chapter presents a proposed standard for IPv6 over IEEE 802.15.4 networks [8]. In the first section, the proposed standard is described while an implementation of 6LoWPAN for TinyOS and its adaption for the radio driver RF212 is presented in section 3.2. Finally, the adapted version is evaluated with the original implementation in section 3.3.

3.1 6LoWPAN

6LoWPAN addresses the problems described in the last chapter and allows the transmission of IPv6 packets over an IEEE 802.15.4 network. The main idea of 6LoWPAN is to introduce an adaptation layer to enable IPv6 communication in WSNs. Whereas the requirements of 6LoWPAN are presented in subsection 3.1.1, the adaptation layer is explained in detail in subsection 3.1.2. Furthermore, the different routing and forwarding protocols of 6LoWPAN are the subject of the last subsection.

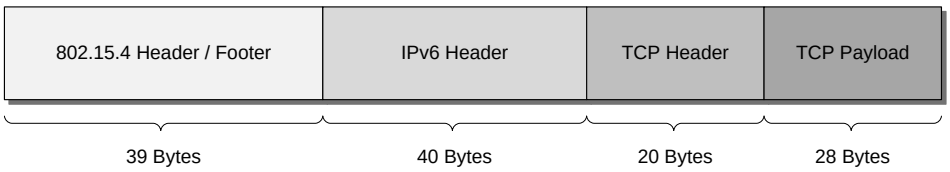


Figure 3.1: IPv6 vs. 802.15.4: header sizes. In the worst case, one may only be able to use a TCP payload of 28 bytes per IEEE 802.15.4 packet when IPv6 is used according to the standard (according to IEEE 802.15.4-2006).

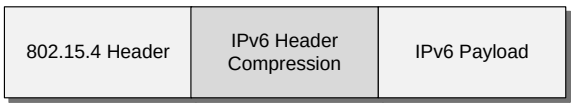
3.1.1 Requirements

The unequal properties of IPv6 and IEEE 802.15.4 cause many requirements to the 6LoWPAN protocol, which have to be considered in order that the underlying network still fulfills the needs of a modern WSN. In this subsection, these requirements are presented by giving an overview about the RFC 4919 [31].

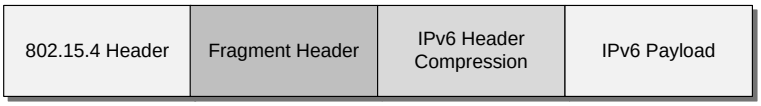
The most important characteristics of the IEEE 802.15.4 standard [19] are its low bandwidth, the requirements for low power and the maximum link-layer packet size of 127 bytes. Implementing IPv6 unaltered over 802.15.4 would result in extremely small packet payloads for higher-level protocols as the following calculation shows. According to IEEE 802.15.4-2006, in the worst case the maximum size of an IEEE 802.15.4 frame is 88 bytes. The IPv6 header has a size of 40 bytes, which results in 41 bytes for upper-layer protocols like TCP or UDP. The length of the TCP header is another 20 bytes. Thus, only 28 bytes per packet are available for application-layer protocols (see Fig. 3.1).

The above example points up the most important requirement of 6LoWPAN: a compression standard for the IPv6 header as well as for the upper layer headers. As IPv6 has a minimum MTU of 1280 bytes, a fragmentation and reassembly layer has to be introduced. Furthermore, the routing protocol should not impose an overhead on data packets. As most devices of a WSN have only a restricted performance, a possible routing protocol should also preserve the computation power and the memory utilization.

Further requirements result from the network topology. To reduce the configuration overhead, a stateless address autoconfiguration is preferable. Various security requirements should also be addressed by the 6LoWPAN protocol, in particular countermeasures against man-in-the-middle attacks and denial of services attacks. Nevertheless, as such countermeasures are still not addressed by current implementations of 6LoWPAN, this report does not provide further ideas to solve the security requirements.



(a) 6LoWPAN packet using only the IPv6 header compression subheader, which has a size of only three bytes in the best case.



(b) 6LoWPAN packet using the IPv6 header compression subheader and the fragmentation header.

(c) 6LoWPAN packet using the IPv6 header compression subheader, the fragmentation header and the mesh addressing header.

Figure 3.2: Examples of various 6LoWPAN header stacks. 6LoWPAN is based on header encapsulation, only the required headers have to be included.

3.1.2 Adaptation Layer and Header Compression

To address the above requirements, the 6LoWPAN protocol [8] introduces the adaptation layer. By using header compression and fragmentation, a 6LoWPAN packet needs a much smaller header than an IPv6 packet would use. After introducing the 6LoWPAN packet format, further details to the IPv6 header compression, the fragmentation header and the mesh address header are given in this subsection.

Introduction

The adaptation layer of 6LowPAN consists mainly of three components: the header compression, the fragmentation and the layer-two forwarding [32]. It uses stateless and shared-context compression to reduce the length of the IPv6 header to a few bytes. The key ideas of the 6LoWPAN adaptation layer are the assumption of shared context like the common network prefix and the use of only a subset of IPv6 functionality.

Similar to IPv6, 6LoWPAN uses also an encapsulated header format consisting of the IPv6 header compression subheader, the fragment header and the mesh addressing header. Fig. 3.2 outlines the most important variants of the header stack. At the beginning of each header, a header type field identifies the header format.

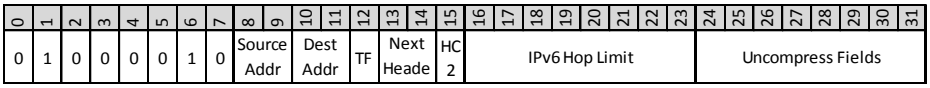


Figure 3.3: IPv6 Header Compression (HC1). The uncompressed fields may use more than one byte but should always be aligned to an octet boundary.

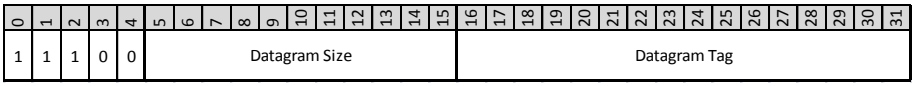
Header Compression

The 6LoWPAN header compression is defined in RFC 4944 [8]. It describes a stateless compression scheme consisting of two parts: the header compression one (HC1) and the header compression two (HC2). HC1 allows compressing the IPv6 header with an original size of 40 bytes into three bytes in the best case. Similarly, the HC2 describes a compression format to reduce the length of the transport protocol header. Both HC1 and HC2 consist of each one encoding byte and non-compressed fields. The latter are dynamically determined for each particular connection.

HC1 reduces the length of the IPv6 header described in subsection 2.4.3 to three bytes in the best case. The compression expects several values for the IPv6 header fields. If the assumptions turned out to be wrong, the non-compressed values of the fields have to follow the encoding field, i.e. carried in-line. The first assumption is that the version is IPv6. The source and destination address is assumed to be link local and to be inferred from IEEE 802.15.4 MAC addresses. Furthermore, the Traffic Class and the Flow Label fields hold the value zero. The packet length is not presented in the header as it can be inferred from the IEEE 802.15.4 length field or from the length field in the fragment header. Last but not least, the Next Header is assumed to be either UDP, TCP or ICMP. As there are no ways to compress the Hop Limit field, it has to be carried in full.

Fig. 3.3 outlines the principle of the compression. The first octet is the dispatch byte, which defines the header format as an HC1 header. The next byte is called HC1 encoding octet. The source address and the destination address bits define whenever the prefix and the interface identifier are carried in-line. The TF bit is zero if the Traffic Class and the Flow Label are not compressed. The Next Header bits are 00 if the fields cannot be compressed and the HC2 encoding bit defines whether a HC2 encoded header immediately follows the HC1 encoding. The Hop Limit field immediately follows the HC2 encoding octet or in the case the latter is not presented, it is appended the HC1 encoding octet. Further non-compressed fields follow the last octet in an order defined in RFC 4944.

6LoWPAN supports also the compression of the transport protocol header,



(a) First Fragment.

(b) Subsequent Fragment.

Figure 3.4: Fragmentation header of 6LoWPAN.

which is denoted as HC2. RFC 4944 only defines a compression format for UDP, which reduce the length of the UDP header from eight octets to four octets in the best case. Several examples of header compressions can be found in [32].

Fragmentation Header

To support the minimum MTU of IPv6, 6LoWPAN introduces the fragmentation header. Whenever the payload is too large to fit into a single IEEE 802.15.4 frame, it will be fragmented into several packets. The fragmentation header for the first fragment is outlined in Fig. 3.4(a) while the header for any subsequent fragment is shown in Fig. 3.4(b). The first field is the Datagram Size, which is the size of the entire IP packet before fragmentation. It is included in each packet to simplify the packet handling in the case of out-of-order arrivals. The Datagram Tag identifies the fragmented payload while the Datagram Offset field defines the offset of the fragment within the original payload.

Mesh Addressing Header

The IEEE 802.15.4 header only contains the source and the destination address of the next hop. If a packet should be transmitted to a node that is not a neighbor of the source, a higher-level protocol needs to implement this functionality. Using IPv6, the originator and final receiver addresses are included in the IPv6 header. Nevertheless, using the compression header this information may be lost. The solution to this problem is to introduce an additional header, the Mesh Addressing header, which is used to support layer-two forwarding. Furthermore, it supports multi-hop forwarding of 6LoWPAN payloads [32]. The header is outlined in Fig. 3.5. The bit V

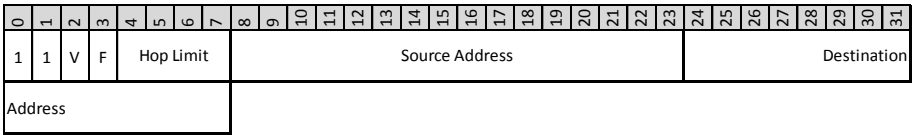


Figure 3.5: Mesh addressing header of 6LoWPAN.

indicates whenever the Source Address is a short 16-bit address or an IEEE extended 64-bit address. The same applies for the F bit and the final destination address. The Hop Limit field is similar to the Compression Header. The Source Address field contains the address of the originator of the 6LoWPAN packet and the Destination Address field the corresponding final destination address.

3.1.3 Routing and Addressing

The underlying network of a 6LoWPAN environment may behave completely different from a typical IPv6 network. In LoWPAN, one may have to consider power issues, multi-hop and other specific features. In this subsection, the routing and forwarding issues of a 6LoWPAN network are considered and the addressing of the nodes is explained. Both are current research topics of the 6LoWPAN community.

Forwarding and Routing

Packet forwarding in a multi-hop environment can be implemented either in the link layer or in the network layer. 6LoWPAN supports both approaches. Forwarding on the network layer is called route over [32]. Its main advantage is that the existing network capabilities of IP can be used. Nevertheless, using route over one cannot utilize the full features of the header compression as the IPv6 addresses may have to be transmitted with each multi-hop packet. An additional problem is that forwarding on the network layer may be slow in reacting to changes in the link state.

To address these problems, 6LoWPAN also supports forwarding on the link layer called mesh under [8]. The source distinguishes between the following two cases: If the destination is directly accessible by the source, the packet is forwarded to the destination. But if no direct reachability between the source and the destination exists, the source node has to include the Mesh Addressing header. The source and the final destination link-layer address are included in the Mesh Addressing header, while the IEEE 802.15.4 header will contain the forwarder's link-layer address. A node, which receives a

frame with a Mesh Addressing header, checks the final destination. If the node itself is not the final destination, it inserts the address of the next hop in the destination field of the IEEE 802.15.4 header and transmits the packet.

Both link-state and distance vector routing protocols do not seem to be well suited for 6LoWPAN networks. Currently the Routing over Low Power and Lossy Links (ROLL) working group within the IEFT is considering this problem [33] and is trying to find a suitable routing protocol.

Addressing

Addressing is mainly based on the stateless address configuration of IPv6, which is described in subsection 2.4.2. The RFC 4944 defines in detail how the Interface Identifier is derived. In general, it is based on the IEEE EUI-64 address of the IEEE 802.15.4 device. In a mesh-under network, the link-local address is used for communication within a LoWPAN and routable addresses are used to communicate outside. Using Route-Over, slight differences have to be considered [32].

The NDP is used in a similar way as in IPv6 and includes prefix discovery and default route configuration. Nevertheless, the protocol is not perfectly adjusted to the 6LoWPAN requirements and may generate an overhead in the number of messages.

3.2 6LoWPAN for TinyOS

The currently most extensive implementation of 6LoWPAN for TinyOS is called blip [10] and is currently being developed at the University of California, Berkeley. Although it implements many functionalities, it only supports platforms with the RF transceiver CC2420 from Texas Instrument[1]. Nevertheless, new platforms often use a transceiver of the RF2xx family from Atmel[2]. Instead of designing a new 6LoWPAN stack for these hardware systems, one has decided to port blip to the RF transceiver family RF2xx. First, blip is introduced by presenting its functionalities and its restrictions. The most important aspects of the porting to the RF transceiver RF2xx are described in subsection 3.2.2 while appendix B gives guidance on how to install blip on a Linux system.

[1] A non-exhaustive enumeration of sensor network hardware systems with the CC2420 RF transceiver includes the Crossbow MicaZ, the Moteiv Telos and the Moteiv Tmote Sky.

[2] This product family includes the AT86RF212, the AT86RF230 and the AT86RF231 transceiver.

3.2.1 Berkeley IP Information (blip)

The functionalities of blip are described in this subsection. First, an overview about blip is given followed by a description of the code design. The routing and forwarding mechanisms of blip are outlined in another part of the subsection while current issues of blip are presented at the end of this subsection.

Introduction

blip is an implementation of 6LoWPAN for TinyOS currently being developed at the University of California, Berkeley[3]. IPv6 neighbor discovery, default route selection and point-to-point routing are the most important features it supports. In addition to the IPv6 functionalities, it supports ICMP, UDP and includes a prototype TCP stack. To connect the nodes with the outside, a tunnel driver for Linux has been included. The computer plays the role of a border router and may forward the packets to the Internet or to another network. The tunnel driver uses `radvd` to implement the NDP. This yields that global, link-local as well as link-local multicast communication is possible when using blip. Addressing, stateless autoconfiguration and header compression are used according to the RFC 4944 [8].

As the sensor network behaves like a normal IP network, well-known computer programs like ping6, tracert6 and nc6 can be used to test and debug the network. Furthermore, blip supports network programming of the sensor nodes using nwprog.

Code Design

The schematic code design of blip is outlined in Fig. 3.6. The network layer mainly comprises three components: the `IPAddress`, the `IPDispatch` and the `IPRouting`. The first one provides the address handling, this includes the IPv6 addresses but also the IEEE 802.15.4 addresses. It implements commands for getting the link local and the global IPv6 address. The routing functionalities of blip are implemented in the `IPRouting` component. It reports the next hop for a packet and implements the NDP commands to build up the routing and forwarding table. The core functionality of the network layer is implemented in the `IPDispatch` component. On the one hand, it handles the reception of packets and forwards them to the correct transport protocol. On the other hand, it is responsible for forwarding a message from the transport layer to the link-layer. It includes the correct route of the packet and buffers the message until the send process is completed.

[3]Unless stated otherwise, this project uses the release with date 3-20-2009.

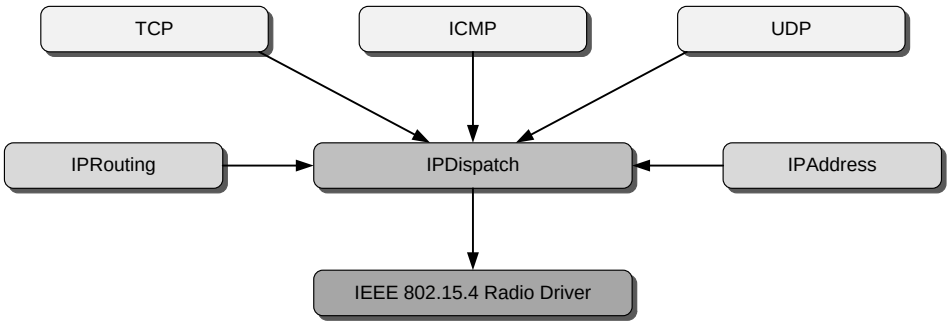


Figure 3.6: Schematic code design of blip. Only the most important components are shown.

Furthermore, it is responsible for the validation of a successful transmission and the fragmentation of packets.

The ICMP component is required to send router solicitations and advertisements. The transport layer comprises a TCP and a UDP component. The latter provides functionalities to easily send and receive UDP packets by one command. The TCP component supports the creation of a socket and allows the use of split-phase connections. It implements a sender and receiver window to guarantee a successful transmission and to support out of order reception of packets. An application that uses the blip IP stack needs to access only the according TCP and UDP interfaces to send and receive messages.

A base station is required to connect the sensor network to a computer, thus to the world. Both, the base station and the computer together, implement the border router functionality. One node has to be programmed with this separate application. The code of this application needs neither the IP layer nor the transport layer, but accesses directly the radio driver. 6LoWPAN packets are directly forwarded to the computer and vice versa.

Routing

There are two basic principles, which define together the routing behavior of the nodes: Each router acts as IP router and every node has a default route towards a border router. From the outside view, a tree with the border router as root is created. The default route of a node is computed by summing up the link-quality indicator (LQI) of the routes.

IPv6 router solicitation and advertisement packets (see subsection 2.4.4) are used to build the tree. A router solicitation message is sent out at boot up

and whenever a default route failure happens. Furthermore, each node sends out advertisement packets whenever it receives router solicitation messages or its hop limit to the border router changes.

blip also supports source routing where the source includes a hop-by-hop way to the destination. As this functionality is non-standard, it is not activated by default. To forward a packet, a node checks if the IPv6 destination is a multicast address. Whenever this condition is fulfilled, the packet is not forwarded. The message is sent to the next hop of the source route whenever the packet contains source route information. If none of the previous conditions is fulfilled, the packet is sent to the default route.

Using software acknowledgements, blip verifies if the packet has received its next hop. If the sender does not receive an acknowledgement message after several retransmissions, it tries to forward the packet to another node in the network hoping that this one can correctly forward the packet.

Issues

The current release of blip contains mainly two known issues [34]: the fragmentation and the buffering. The fragmentation of IP packets used in blip provides several problems, in particular when using multi-hop. The only known solution is to use short messages.

Buffering is a second known issue of blip. It is mainly based on several message buffers and windows, which consume a large amount of memory. This amount can be reduced by minimizing the length of the buffers. Nevertheless, this leads to several problems as messages are dropped if the buffer has no free space. The restriction primarily concerns older platforms, which have only an internal memory of four Kbytes. This project addresses this issue by porting blip to a modern platform that includes eight Kbytes of RAM, thus supporting much larger buffers.

3.2.2 Porting blip to ZigBee900 Platforms

The process of porting blip to the MeshBean900 and Pixie platform can mainly be divided into two parts. The current radio driver of the RF transceiver is extended in a first step. It has to support new interfaces and functionalities like reliable connections. In a second step, blip is adjusted to the new driver. The release with date 3-20-2009 is used as basis of the porting, which is outlined in this subsection. In collaboration with the University of Szeged, the updated driver has been published in the official TinyOS repository [35].

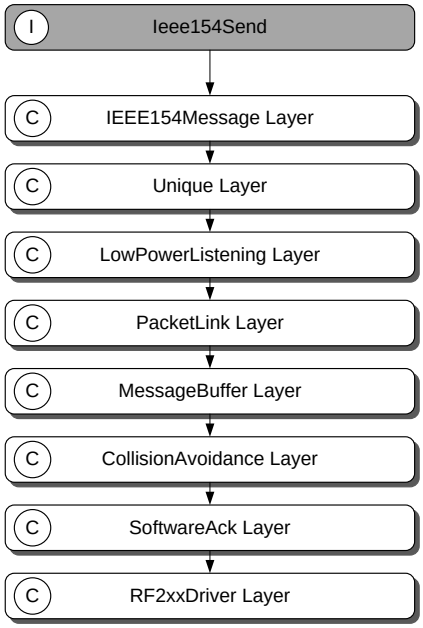


Figure 3.7: The send process of the RF2xx transceiver driver used in blip to provide the `Ieee154Send` interface (schematically). Compared to the standard RF2xx radio driver, the `IEEE154Message` layer has been extended and the `PacketLink` layer has been introduced. Interfaces are marked with Ⓘ while components are marked with Ⓒ.

The New RF Transceiver Driver

As mentioned earlier, blip is directly ported to all RF2xx transceivers. The aim is to support all platforms that use a chip of the RF2xx transceiver family for their communication. This is not difficult as most layers of the RF2xx driver are written independent of the concrete transceiver. To support blip, mainly two new interfaces have to be provided by the radio driver, the `Ieee154Send` and `Ieee154Packet` interface. Furthermore, the implementation of the `Receive` interface has to be adjusted.

The `Ieee154Send` interface is almost similar to the `AMSend` interface, but uses another address type. As outlined in Fig. 3.7, the interface is provided by the `IEEE154Message` layer. This layer has been extended to support the new interfaces of blip. It initiates the data frame, sets the correct payload length and all addressing resources of the IEEE 802.15.4 frame. The `PacketLink` layer has newly been introduced in the send process with the object to guarantee a reliable connection. The layer tries to send the message several times until the next-hop receiver confirms the packet reception. Further-

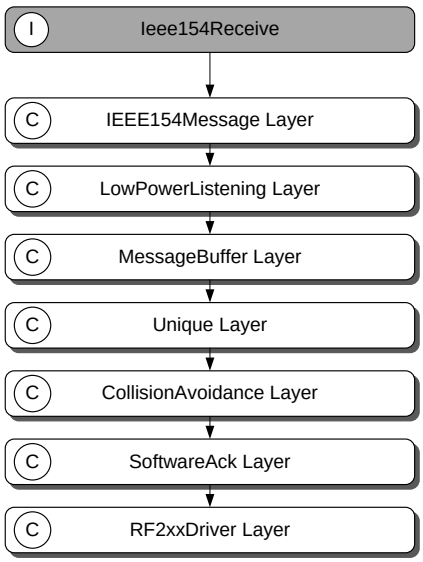


Figure 3.8: The receive process of the RF2xx transceiver driver used in blip to provide the `Ieee154Receive` interface (schematically). Compared to the standard RF2xx radio driver, the `IEEE154Message` layer has been extended. Interfaces are marked with Ⓘ while components are marked with Ⓒ.

more, it allows a higher-level application to check if a packet is successfully transmitted or might be lost.

The receive process, which is outlined in Fig. 3.8, can mainly be copied from the standard RF2xx reception, as the `Ieee154Receive` interface is similar to the standard `Receive` interface. The `IEEE154Message` layer is used to examine whether the link-layer address corresponds to its own address. The newly introduced `Ieee154Packet` interface is responsible for the addressing of a message. It allows setting the source and destination addresses and provides functionalities to handle the PAN identifier. The packet is also provided by the `IEEE154MessageLayer`. Further modifications have been done at the send and receive process to fully support all required functionalities of blip.

Adjust blip to the New Radio Driver

The adjustment of blip to the new RF transceiver driver is straightforward. The base station application and the IPDispatch layer have to be reconnected to the new driver. In addition, few detail implementations of blip have to be adjusted as the CC2420 transceiver behaves different from the RF2xx family in a handful of cases. Consider for example the LQI. For the CC2420

transceiver, the values of the LQI are between 50 and 110 [36] while the RF212 transceiver reports an LQI value of zero up to 255 [18]. Entering a LQI value of 255 in blip would generate an overflow and the corresponding route is reported to be quite worse. This can imply that a node cannot calculate its default route, thus may be unable to communicate with other nodes. By setting the correct preprocessor directive in the ported version of blip, the programmer can choose whenever the radio driver of the RF212, RF230 or CC2420 should be used.

3.3 Comparison

After porting blip to hardware devices with the RF212 transceiver, platforms with the CC2420 radio transceiver are compared with RF212 platforms. The measured round-trip time (RTT) of a ping request is used to evaluate the performance of both platforms. In subsection 3.3.1, the most important results of this performance test are presented. By considering the properties of both target platforms, the advantages and disadvantages of RF212 platforms are evaluated with respect to ones with the CC2420 transceiver in subsection 3.3.2. Self-evident, the comparison is performed with regard to the blip implementation.

3.3.1 Performance

As a first step, the performance of a node with the RF212 transceiver is compared with a node with a CC2420 transceiver. The standard test application of blip, `UDPEcho`, has been installed on all nodes. Ping is a small tool for testing networks and it measures the RTT between a client and a server. It sends out an ICMP packet "echo request" and measures the time until it receives the corresponding ICMP "echo response" message. Furthermore, the tool can be used to test whenever a node is reachable in a network.

IPv6 is supported by ping6, which is used in this evaluation to measure the RTT between the computer acting as border router and a node in the network. Three different types of nodes are used: the MeshBean900, the Pixie and the Moteiv Tmote Sky. The first two nodes running on the ZigBee900 chip while the Moteiv Tmote Sky uses a TI MSP430 microprocessor and a Chipcon CC2420 radio. It includes 10 Kbytes SRAM and 48 Kbytes Flash storage. The MeshBean900 and the Pixie are based on the same architecture, but the MeshBean900 uses a stronger and bigger antenna. For the evaluation, the RTT has measured 150 times per experiment.

The average RTT of the three nodes is outlined in Fig. 3.9. For each node, the single hop and two-hop RTT is measured. In the case of single hopping,

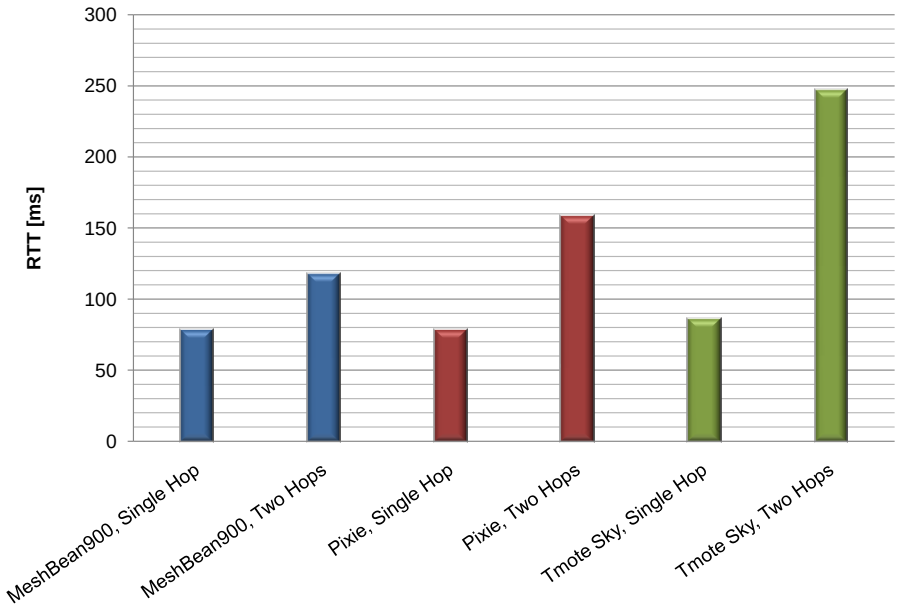


Figure 3.9: The average RTT of a ping request for a MeshBean900, a Pixie and a Tmote Sky node. The diagram differs between the RTT for a single hop and a multi-hop ping request.

the RTTs for both the MeshBean900 and the Pixie are almost the same, no significant difference can be detected. In the average, the RTT of a Tmote Sky device is slightly longer for single hop. The RTT for two hops is striking longer for all nodes. The MeshBean900 has the smallest RTT while the RTT of the Tmote Sky is almost two times longer as the one of the MeshBean900.

To clarify the above results, the variance of both platforms is considered and outlined in Fig. 3.10. One might be aware that a logarithmic scale is used for the RTT variance. This demonstrates the biggest difference between the ZigBee900 devices and the Tmote Sky: the RTT of a MeshBean900 is almost constant while the one of the Tmote Sky has a large variance. The fastest RTT for the Tmote Sky is only 65 milliseconds while many values are around 100 milliseconds. This phenomenon may be based on the fact that the antenna of the ZigBee900 devices is much powerful than the one of the Tmote Sky. The distances between two nodes are almost identical for all experiments. As a stronger antenna results in less retransmission, the nodes with a more powerful antenna have a smaller variance and a faster RTT.

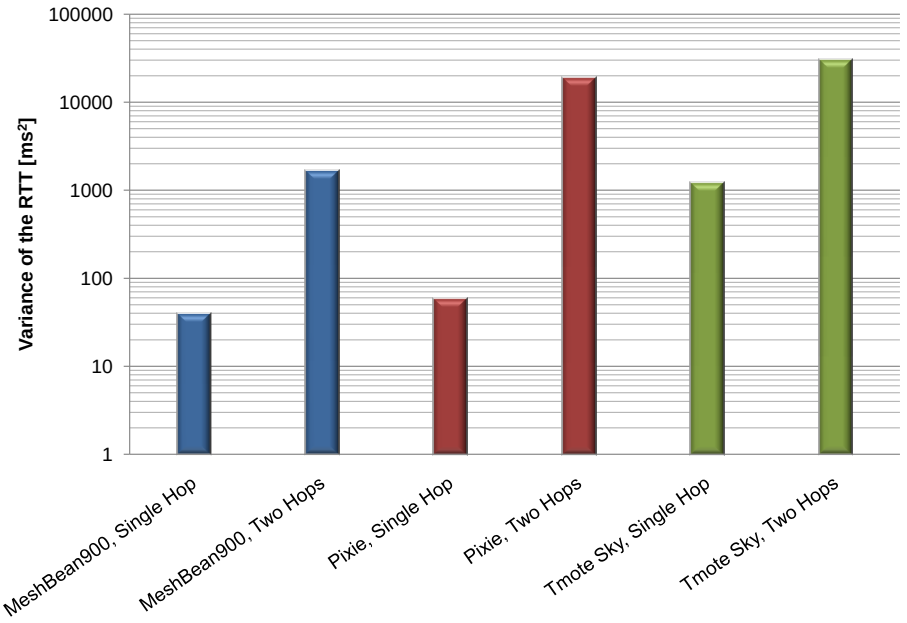


Figure 3.10: The variance of a ping request for a MeshBean900, a Pixie and a Tmote Sky node. The diagram differs between the variance for a single hop and a multi-hop ping request. Be aware that a logarithmic scale is used for the RTT variance.

3.3.2 Advantages of the Porting

To complete the comparison of both RF transceivers, several advantages of RF2xx transceiver platforms are presented compared to ones with the CC2420 transceiver.

blip currently only supports the RF transceiver CC2420. Nevertheless, new platforms are often running on a chip of the RF2xx transceiver family. The most famous hardware device with an RF2xx chip is the Crossbow IRIS, which is now also supported by blip. Both the MeshBean900 and the IRIS platform are running on more powerful microprocessors than the classic CC2420 platforms like the MicaZ. They include a larger memory and a higher clock speed, thus the issue with the buffer sizes mentioned in the last subsection is at least partly solved. Furthermore, various experiments suggest that the RF2xx transceiver is more stable than the CC2420 transceiver.

Almost all modern sensor node platforms that are also supported by the TinyOS are compatible with blip now. This enables developing future TinyOS projects based on 6LoWPAN thus exploiting the advantages of the IP technology.

3.4 Summary

6LoWPAN is a specification of the IETF group to standardize the use of IPv6 over IEEE 802.15.4 networks. Based on header compression and fragmentation, an efficient implementation of IPv6 in WSN has been developed, which is able to support most tools and features of a real IPv6 network.

blip has been presented as an implementation of 6LoWPAN for TinyOS, which supports almost all features of the 6LoWPAN standard. This realization has been ported to the new RF2xx transceiver family during this project and the resulted modifications of the corresponding radio driver have been published to the official TinyOS repository. Various comparisons have shown the advantages of the new platforms compared to the classic hardware devices of blip.

4

Web Services

In the vision underlying the Web of Things, all objects of the world are integrated in the existing Web. The most realistic scenario is that each real-world thing provides an API, which allows another device to access its resources. This chapter proposes a solution to integrate sensor nodes into the Web. The 6LoWPAN stack presented in the last chapter enables that each device is directly connected to the Web. Based on this IP stack, a RESTful API for sensor nodes has been developed. It enables other devices to access a node's resources by using its API.

In the first section of this chapter, a short overview about the technologies used in the implementation of the Web services is provides while the implemented API is presented in section 4.2. Finally, section 4.3 proposes another approach to connect the nodes to the Web: The nodes report independently their status to Twitter [37].

4.1 Web Service Technologies

A Web service consists mainly of two technologies. A transfer protocol is used to communicate between the service requester and the service provider while a data format protocol is required to guarantee that both participants interpret the exchanged data identically. Nowadays, either SOAP or REST is often used as data transfer protocol. There exist various protocols for the data format; the most famous ones are the Extensible Markup Language (XML), RSS or the JavaScript Object Notation (JSON). To implement Web

services on sensor nodes, both protocols should be resource saving. REST and JSON are therefore used in this project. REST is introduced in subsection 4.1.1 whereas an overview about JSON is given in subsection 4.1.2.

4.1.1 Representational State Transfer (REST)

REST [38] can be outlined as a collection of network architecture principles. Each data object is referred as a resource and REST describes how these resources can be addressed. When using HTTP, this protocol can be misused to communicate directly with the resource. Although HTTP is often used in conjunction with REST, it can also be implemented using other high-level protocols. One speaks about a RESTful system if it follows the principles of REST.

A group of information a system can provide is denoted as a resource. A host can access the resource using its identifier and perform some actions to this resource. An example for an identifier is the Uniform Resource Identifier (URI) in the context of HTTP. The creation of a new resource or the retrieving of the resource data are two possible actions.

In connection with REST, a Web service is a collection of data that can be accessed using HTTP as its interface. One can address either the collection or any of its members. The four main operations of HTTP can be used to operate on these resources: GET, POST, PUT and DELETE. Each of them executes a corresponding action. For instance, a GET operation can be used to retrieve a list of the members of a collection or to access the data of an addressed member of a collection.

SOAP relies on another application layer protocol for transmission. HTTP and RPCs are two such protocols. Thus, REST is more lightweight as it misuses the application layer protocol for the transmission. Furthermore, the overhead introduced by using the xml language to describe the operations in SOAP does not exist in REST, thus it is often easier to implement. As sensor nodes are often limited in their computational performance, REST seems to address their requirements better than SOAP. This concludes why REST is used as transport protocol for the Web service application developed in this project.

4.1.2 JavaScript Object Notation (JSON)

Using tags to structure data, xml can generate a huge overhead. Furthermore, neither the production nor the parsing of xml is straightforward as it is not a programming language object, but a separate data construct. JSON addresses these problems and defines a small data interchange format, which

Listing 4.1: Example of a JSON object according to RFC 4627 [39].

```
{
  "Image": {
    "Width":  800,
    "Height": 600,
    "Title":  "View from 15th Floor",
    "Thumbnail": {
      "Url":    "http://www.example.com/481989943",
      "Height": 125,
      "Width":  "100"
    },
    "IDs": [116, 943, 234, 38793]
  }
}
```

consists of a small set of formatting rules. Although JavaScript is included in its name, JSON is language independent. Nevertheless, the JSON data are according to the JavaScript definition, thus a JSON object can easily be parsed in JavaScript. That is partly the reason why many Rich Internet applications (RIAs) use JSON as their data format protocol.

JSON is defined in RFC 4627 [39] and uses four primitive (string, numbers, Booleans and null) and two structured (objects and arrays) types. An array is an ordered sequence of zero or more values while an object is an unordered collection of zero or more name and value pairs. A value can be any primitive or structured type. This allows the nesting of several objects and arrays. To differ between types, JSON uses six structural characteristics: the square brackets for arrays, the curly brackets for objects, the colon to separate names from values and the comma to separate values. An example of a JSON object can be found in Listening 4.1.

4.2 RESTful API for Sensor Nodes

The above-described technologies enable the development of an API through whom the sensor nodes can be accessed through the Web thus put the Web of Things in practice. A RESTful API has been implemented on the sensor nodes, which provides the current values of the sensors as a resource. In subsection 4.2.1, the structure of the implementation is presented while the API is described in appendix C.1. The Web application described in subsection 4.2.2 allows someone to access the nodes in a human-friendly way. Furthermore, the most important results of the evaluation of the above describe

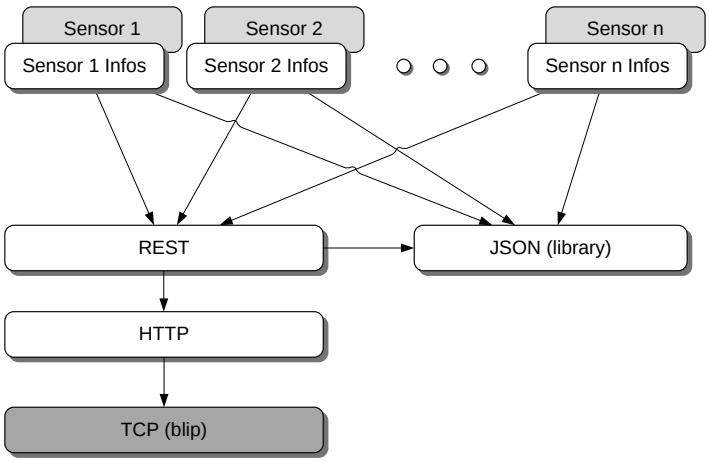


Figure 4.1: Outline of the RESTful API for sensor nodes implemented in TinyOS. A sensor has to implement the REST component to provide its data to the world. Furthermore, an HTTP layer and a JSON library for the creation of a data packet in the JSON format have been integrated.

implementation are presented in subsection 4.2.3. Finally, to demonstrate the possibilities of such an implementation, a mashup application is presented in subsection 4.2.4. This example of an RIA highlights the advantages of the Web of Things.

4.2.1 A RESTful API

To provide Web Services on a sensor node, a corresponding provider has been implemented. Using blip as IP and TCP stack, three additional components have been added to provide a simple way to extend the resources of a node. The principle is schematically outlined in Fig. 4.1. The incoming request is a REST command while the answer is a standard HTTP message, which includes the requested response.

The HTTP layer is responsible for the communication between the TCP and the REST layer. This mainly implies the initialization of the TCP layer, but also provides a functionality to send HTTP responses. On the one hand, it includes functionalities to create a correct HTTP header and on the other hand, it splits the HTTP message in several small packets to minimize the fragmentation problem (see subsection 3.2.1). When the HTTP layer receives a TCP packet, it forwards this packet directly to the REST layer. It analyzes the REST request, filters out the action and the URI of the request and informs the corresponding collection. Furthermore, it

manages all registered collections and answers to a root request. The JSON component provides several commands to generate a valid JSON object and to add parameters to the JSON object. A parameter is a concrete value provided by the resource like the current value of the temperature sensor. To send a JSON answer, the sensor resource has to generate a JSON object in a first step and forwards this data to the REST layer in a second step.

By sending a REST command to the corresponding URI, one can access the resources of the sensor node. The host name of each sensor node is its IPv6 address. Three different types of URIs exist:

- A *root collection* is accessed by sending a GET to the root ("/") and is answered with a list of all collections on this sensor node. An example of a valid URI is `http://[2001:470:1f04:56d::65]/`.
- To access a *collection*, a GET request to the collection address followed by an asterisk has to be send. For instance, `http://[2001:470:1f04:56d::65]/temperature/*` is a correct URI to access the collection `temperature`. Only the GET request is supported for collections and the answer involves a list of all its members.
- By sending the corresponding REST command to a *collection member*, the actual data can be accessed. Depending on the functionality, a member may support GET, PUT and DELETE actions. An example of a valid URI is `http://[2001:470:1f04:56d::65]/temperature/celcius`.

Listening 4.2 outlines a possible answer to a GET request, which was addressed to a member of a collection. The answer includes the name of the resource, which types of REST commands are supported and self-evidently, all parameters of the member. Further information on the API and a detailed description of the answered JSON object can be found in appendix C.1.

4.2.2 Presentation

As neither the manual parsing of a JSON answer nor the manual sending of PUT and DELETE requests is very comfortable, a Web application has been developed to support the user by these tasks. It presents the answers of the sensor nodes in a user-friendly way and provides a simple way to send REST requests. In this subsection, the implementation of the Web application and its functionality are described while an overview about the front-end is given in appendix D.

The front-end has been developed as AJAX (asynchronous JavaScript + XML) Web application using Google Web Toolkit (GWT) in version 1.6 [40].

Listing 4.2: Example of a JSON object sent by the RESTful API in response to a GET request.

```
{
  "device": "Temperatur",
  "method": [
    "G"
  ],
  "param": [
    {
      "n": "value",
      "v": 3392,
      "t": "i",
      "u": 0
    },
    {
      "n": "celcius",
      "v": 26,
      "t": "i",
      "u": 0
    }
  ]
}
```

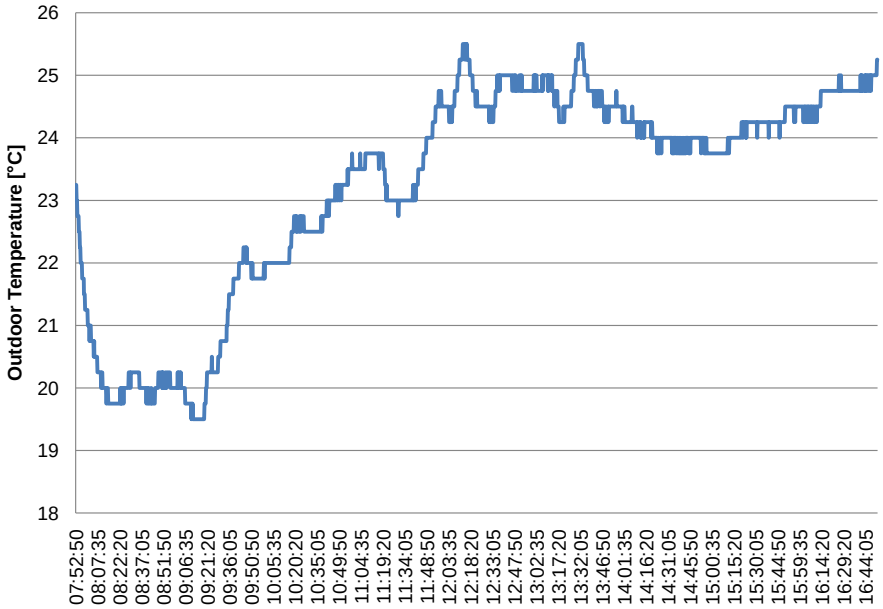


Figure 4.2: Recording of the outdoor temperature using the RESTful API and a MeshBean900 sensor node. The Web application has been used to record the data while the diagram is generated using Microsoft Excel.

It allows the connection to a sensor node, to explore its collections and to view the corresponding parameters. Sending the modified parameters back to the sensor node can be done by one click. Furthermore, various statistic functionalities are implemented like measuring the time until an answer returns. The application provides a record functionality to log the values of a parameter. This is implemented by sending a GET request in a user-selected interval. This can be used to generate diagrams like the one in Fig. 4.2. Additionally, the user can choose if the TCP request should be closed or not after each request.

As the "same origin policy"[1] of JavaScript does not allow sending PUT, DELETE and GET request to other hosts than the current one, server calls are used to communicate with a back-end. The server calls are realized using remote procedure calls (RPC) already included in GWT. The back-end is programmed in Java and simply forwards the messages to and from the sensor nodes.

[1]The same origin policy permits scripts to access data with another origin as the one of the page the script is running. In order that the origins are the same, the domain name, the application layer protocol and the port have to correspond.

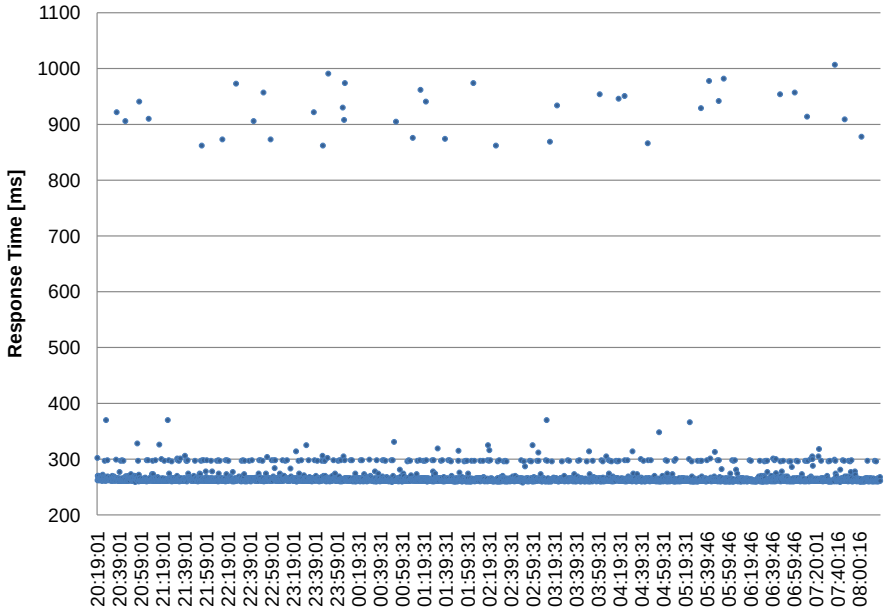


Figure 4.3: Response times of a long-term recording. The response times are mainly around 270 and 900 milliseconds.

4.2.3 Evaluation

Using the recording functionality of the above-described Web application, the RESTful API has been tested and evaluated. The target platform of all evaluations is the MeshBean900. A first test addresses the stability during a long time recording. Furthermore, the overhead introduced by the SYN / FIN handshake and the dependency of the response time on the packet length are examined. The last evaluation compares the response times with the number of hops between the sensor node and the base station.

Stability

To test the stability of the RESTful API, the Web application has requested every 15 seconds the data of the light sensor. No other node has been turned on during the evaluation. The response time of each request has been measured for twelve hours. The response time starts with the first SYN sent by the Web application and stops with the end of the FIN handshake.

The response times are outlined in Fig. 4.3. Total 2884 requests have been sent to the sensor node during the evaluation. Five requests have been lost, respectively not been fully responded and 39 answers have required

longer than 500 milliseconds. This corresponds to only 1.35 percent of all requests. The average response time is 275 milliseconds. One can categories the response time as follows: response times around 270 milliseconds, slow responses around 900 milliseconds and lost responses. In the first category, each TCP packet has been successfully transmitted. If a TCP packet has not been acknowledged, the sensor node has waited a few 100 milliseconds and has retransmitted the packet one more time. This corresponds to the second category with response times around 900 milliseconds. During the transmission of the last category, more failures have happened, and the Web application has not gotten the full response after two seconds, thus marking the response as lost.

The loss of single packets has to do with the fact that no other sensor node can try to transmit the packet to the base station and vice versa. Using the `PacketLink` layer, blip can verify if a packet has been received by the next-hop destination. If the sender does not get an acknowledgement, it automatically tries to transmit the packet by means of other nodes.

Various Message Lengths and TCP Connection Overhead

Self-evident, the average response time of a request depends on the length of the HTTP payload. For the evaluation, various requests have been sent to a sensor node and the response time has been measured. On the one hand, it is distinguished between three different payload lengths (94, 201 and 335 bytes) and on the other hand, between the cases whenever the TCP connection is newly established for each request or not.

The chart outlined in Fig. 4.4 shows that the average response time almost linearly increases with the message length. An HTTP payload length of around 94 bytes corresponds to a single parameter while 335 bytes corresponds to six parameters. Nevertheless, these are only approximate values as the actual length depends on the name and value of the parameters. In case of a payload length of 95 bytes, the HTTP packet of the response has to be fragmented into three 6LoWPAN packets while the response of the packet with a size of 335 bytes has to be fragmented into six 6LoWPAN packets.

The overhead caused by establishing and closing a TCP connection is also outlined in Fig 4.4. In the case of a small HTTP payload, the overhead makes up approximately 25% of the whole answer time. Nevertheless, always closing and opening the connection has two main advantages. On the one hand, the connection is more stable over a long time, as the connection would be reset if only one node produces an erroneous behavior and on the other hand, a sensor node cannot store many open connections. If various clients try to connect to a single node, only a few clients would be able to establish the connection.

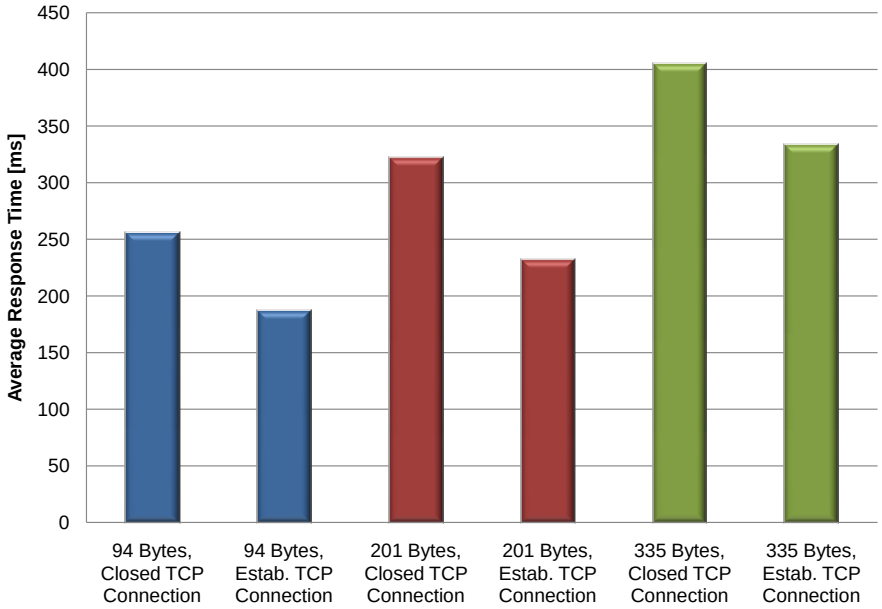


Figure 4.4: The average response time for various message lengths. The number of bytes corresponds to the HTTP payload size. Furthermore, the chart differs whether the TCP connection is closed, i.e. is newly established for each request, or not.

Multi Hopping

The last evaluation focuses on multi hopping. As seen in subsection 3.3.1, the RTT has significantly increased in the case of multi hopping. Similar to the previous tests, a request for a resource has been sent to the sensor node and the response time has been measured. The response has a HTTP payload length of 94 bytes.

Fig. 4.5 outlines the average response time for a one-hop and a two-hop response. The two-hop response time is more than two times longer as the one for the single hop, which has to do with the fact that there exist much more possibilities for transmission errors. The response time of a request for a node, which is three hops away from the base station is not outlined in the chart. Various experiments have shown that the response time increases to around four seconds for this scenario. Apart from the reasons mentioned above, one has to consider the test setup. To achieve multi hopping with two MeshBean900s, the distance between the nodes has to be in the tens of meters. Furthermore, only the nodes, which are necessary for the corresponding test, have been turned on. The huge distance between the nodes results in a higher packet loss rate and retransmissions are necessary.

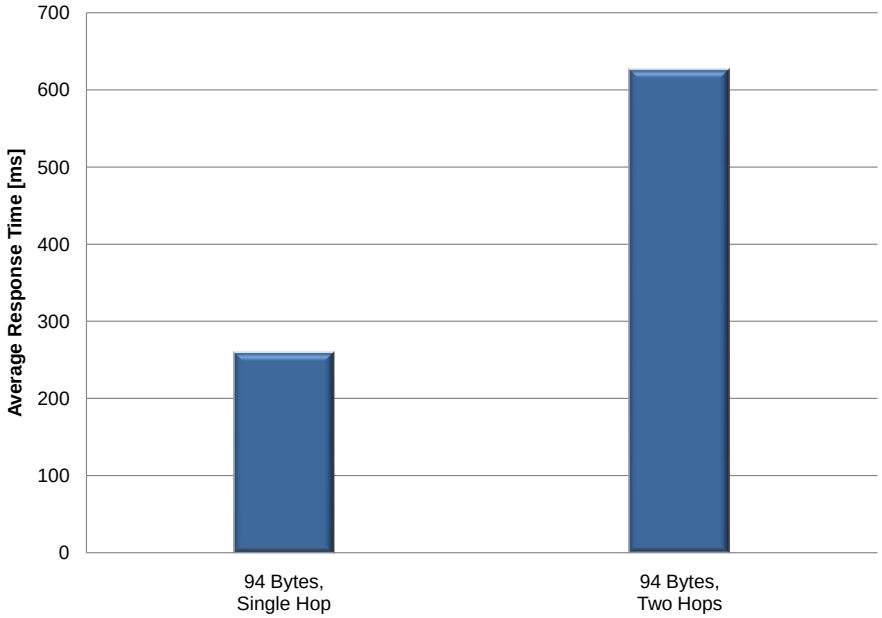


Figure 4.5: The average response time of a one-hop answer and a two-hop answer. In both cases, the HTTP payload of the response has a length of 94 bytes.

By considering the fact that TCP waits a long time until it retransmits an unacknowledged packet, the large average response times for multi hopping can be explained. Similar to the stability evaluation, a better performance might be achieved by using a dense network.

4.2.4 Mashup

A mashup is characterized as a Web application that uses data from more than one external online source. It is often referred to a Web page that uses a map to display data of some objects. In the context of sensors, it is conceivable that all nodes are outlined on a map and further information is provided whenever a user clicks on a marker. [41, 42] are examples of this principle but take into account that the sensors in these projects are either connected to a computer or integrated in a buoy but they are not embedded in a low-power sensor node.

When the sensor node is directly connected to the Web, a mashup application can directly access the data of a node. This enables a simple tracing of the sensor nodes. To prove this statement, a simple mashup application has been implemented using the Google Maps API [43]. The application requests the

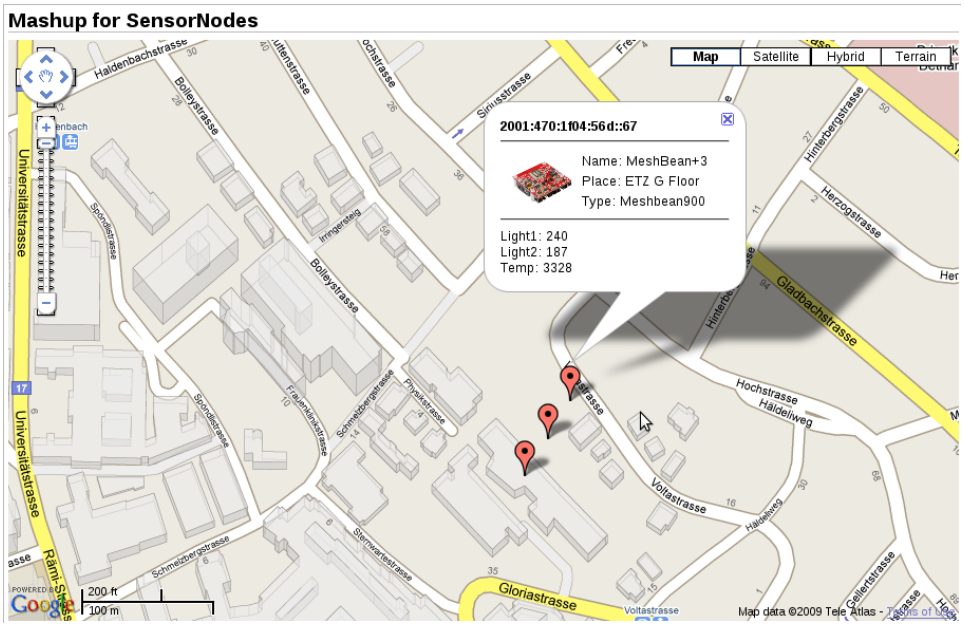


Figure 4.6: Mashup application for sensor nodes. Each marker represents a sensor node. By clicking on the marker, additional information is displayed.

coordinates of all nodes when it starts and displays them on a map. By clicking on a marker, the application requests detailed information about a node and provides them to the user. Fig. 4.6 shows the application while the corresponding API of the sensor nodes is described in appendix C.2.

For instance, various applications of this principle are conceivable in the logistics sector. Each container would include a sensor node that can be accessed through the Web and an operator can monitor the current positions of the containers on a mashup application.

4.3 Sensor Nodes Start Twittering

In the last section, it has been shown how the ideas of the Web of Things can be implemented on sensor nodes. Another node in a network requests some information from a sensor node. The opposite of this implementation is that a node directly reports its status to a server in the Web.

To analyze this scenario, a sensor node application has been developed, which reports is status to Twitter [37]. By clicking on a button, the sensor node sends a corresponding message to Twitter. Fig. 4.7 shows a print screen of the corresponding Twitter account. As the standard TCP implementation

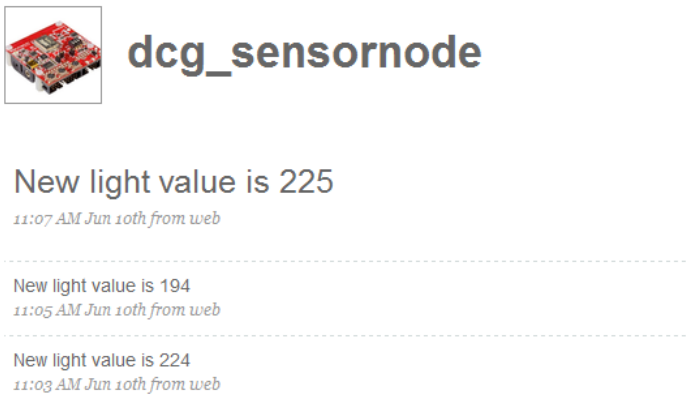


Figure 4.7: Twitter status messages of a sensor node.

of blip has many bugs for the case a sensor node connects to another TCP server, a non-official TCP implementation has been used [44].

As Twitter does not support IPv6, an IPv6 transition mechanism has to be used to access Twitter from the sensor nodes. There exist two standard mechanisms for this purpose: Both NAPT-PT (defined in RFC 4966 [45]) and TRT (defined in RFC 3142 [46]) rely on header transition between an IPv6 and IPv4 packet. As no implementation of both transition mechanism operates free of errors under a modern Linux system, an own solution has been developed in Java. This gateway rewrites the packet as an IPv4 packet and vice versa.

4.4 Summary

A RESTful API for sensor nodes has been developed using JSON as data format. It allows the reading out of the sensor data on the one hand and the modification of the settings of the low-power device on the other hand. As REST is used as transport protocol, all commands are sent as HTTP packets to the sensor node. To improve the usability of the API, a Web application has been developed that offers a user-friendly presentation of the sensor data. The evaluation of the API has shown that TCP transmissions cause a huge overhead and a dense network is required to achieve reliable transmissions of large TCP packets over multiple hops. To demonstrate the various possibilities of the Web of Things, a mashup application has been programmed. It allows locating the sensor nodes on a map and, by selecting one node, gaining further information about its sensors.

5 Conclusion and Outlook

5.1 Conclusion

During this semester thesis a 6LoWPAN implementation for the newest generation of low-power sensor nodes has been developed. These hardware devices are running with the ATmega1281V Microcontroller and use a radio transceiver of the RF2xx family from Atmel. 6LoWPAN provides the transmission of IPv6 packets over an IEEE 802.15.4 network, thus a WSN. The main advantages of using IPv6 in sensor node networks is that it enables the use of standard networking tools, which were originally developed for the Internet, also for WSNs. Furthermore, it simplifies the task of interlinking WSNs over the Internet, as there is no need for an advanced gateway. The packets can simply be forwarded on the link-layer.

The current most fully developed 6LoWPAN solution for TinyOS is called blip and has been used as basis for this project. blip has been developed for platforms with the RF transceiver CC2420 from Chipcon and implements ICMP, UDP and TCP. This implementation has been ported to sensor node platforms with a radio transceiver of the RF2xx family. The porting is composed of two parts. In a first step, the TinyOS driver for the RF2xx transceiver has been extended to implement the required functionalities of blip. The modified components have been integrated into the official TinyOS repository. The second step of the porting includes the modification of blip to use the new radio stack.

Using this IP stack, the newest generation of sensor nodes is part of the Inter-

net of Things. Sensor nodes can connect to the Internet, but also computers linked to the Internet can access the sensor nodes.

In the Web of Things, each device can be accessed through the Web. A RESTful API has been proposed, which allows a computer to connect to the sensor node using HTTP. It enables the communication between a user and the sensor node based on a high-level RESTful protocol. The answer of a sensor node is formatted as JSON. The API allows the readout of the sensor parameters and the modification of the sensor node's configuration.

A Web application based on AJAX has been developed to provide a graphical user interface for communicating with the sensor nodes. Apart from the exploration of the resources, it provides control mechanisms like turning on or off the leds with one click. Furthermore, additional features like recording of sensor node data have been implemented. The report has highlighted several examples that use the recording feature. Various evaluations have shown that the overhead caused by TCP packets cannot be neglected and dense networks are required to achieve reliable transmissions of large TCP packets over multiple hops.

To present the possibilities of the Web of Things, a mashup application has been developed. Nodes can be placed anywhere on the world but are connected through the Internet to the computer on which the application runs. Each node is marked on a world map thus enabling a simple monitoring of several WSNs together.

The connection of a node with the Web has been presented with the help of a further implementation. Sensor nodes send status messages to Twitter. This experiment has highlighted the problems of current IPv6 implementations. Nowadays, the Web is running with IPv4 and huge efforts are required to connect an IPv6 device to the IPv4 network.

5.2 Outlook

With the release 2.1.1 of TinyOS in August 2009, blip will be integrated into the TinyOS core with the consequence that the packet format of blip is going to change. By introducing a dispatch value, the radio stack can support both Active Messages and other network protocols. It looks as if the main interfaces will change and thus the radio stack driver and the applications have to be adapted.

Compared to the CC2420, the RF212 transceiver integrates additional hardware support, which could extend the 6LoWPAN implementation. For instance, the Advanced Encryption Standard (AES) is supported by hardware and could be used to integrate the Internet Protocol Security (IPsec) in the

6LoWPAN stack.

Large experiments with many nodes and several WSNs connected over the Internet can demonstrate the possibilities of the Web of Things. They can also be used to find out the boundaries of this vision.

At this point, we take the liberty to make an appeal to network administrators to support IPv6 in their networks. Furthermore, big Web sites ought to support IPv6. Google sets a good example by offering its search over an IPv6-only website[1]. We think that IPv6 is important for the long-term development of the Internet, but a switch to IPv6 is almost impossible as long as most services are not offered over IPv6.

[1] `http://ipv6.google.com/`

A

List of Acronyms

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
AES	Advanced Encryption Standard
AJAX	Asynchronous JavaScript + XML
BPSK	Binary Quadrature Phase Shift Keying
DCG	Distributed Computing Group
DHCP	Dynamic Host Configuration Protocol
DSN	Data Sequence Nnumber
FCS	Frame Check Sequence
GWT	Google Web Toolkit
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPsec	Internet Protocol Security
IPSO	Internet Protocol for Smart Objects
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ITU	International Telecommunication Union
JSON	JavaScript Object Notation
LQI	Link-Quality Indicator
LR-WPAN	Low-Rate Wireless Personal Area Network
MAC	Media Access Control
MHR	MAC Header
MFR	MAC Footer

MTU	Maximum Transmission Unit
NDP	Neighbor Discovery Protocol
OS	Operating System
QoS	Quality of Service
QPSK	Offset Quadrature Phase Shift Keying
PAN	Personal Area Network
PHR	PHY Header
REST	Representational State Transfer
RFID	Radio-Frequency Identification
RIA	Rich Internet application
RPC	Remote Procedure Calls
RTT	Round-Trip Time
SHR	Synchronization Header
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
WSN	Wireless Sensor Network
XML	Extensible Markup Language

B

Installing blip under Linux

B.1 Installation

B.1.1 Prerequisites

- Make sure that the latest TinyOS version is installed. The variable `$TOS_ROOT` is assumed to point to the current root folder of the TinyOS installation.
- Download the modified version of blip for the radio stack RF2xx.
- Build the c serial forwarder tools in `$TOS_ROOT/support/sdk/c/sf`. To generate `libmote.a`, `bootstrap`, `configure`, and `make` have to be executed in this order.

B.1.2 Environmental Variables

The following environment variables have to be added to the startup script:

- `LOWPAN_ROOT=<blip root directory>`
- `TOSMAKE_PATH=''$LOWPAN_ROOT/support/make''`

`LOWPAN_ROOT` has to be replaced with the path of the blip root level directory. If blip were included in the original TinyOS installation, one would have to set `LOWPAN_ROOT` to the TinyOS root folder.

B.1.3 Tunnel Driver Installation

Before starting the tunnel driver, a node has to be installed with the base station application. It can be found in `$LOWPAN_ROOT/apps/IPBaseStation`. Further information to the installation of a blip application can be found in subsection B.2.1.

To build and start the tunnel driver, execute the following steps:

- Change the directory to `cd $LOWPAN_ROOT/support/sdk/c/blip`.
- Compile the driver with `make`.
- Execute the tunnel driver with the command `sudo ./ip-driver /dev/ttyUSB0 <baud rate>`. `USB0` has to be replaced with the USB port to which the base station is connected.

In the case the message `sendmsg: Operation not permitted` is shown and the tunnel driver is not correctly started, IPv6 is disabled in the firewall and is only allowed to be used in combination with the loopback interface. This configuration is currently standard in many Linux distributions. To enable the IPv6 traffic under Ubuntu, the line `IPV6=yes` has to be added to the file `/etc/default/ufw`. After restarting the firewall, the tunnel driver should work fine.

Modifying the file `$LOWPAN_ROOT/support/sdk/c/blip/serial_tun.conf` allows someone to use its own networking configuration.

B.2 Test the Configuration

B.2.1 Installing the Test Application

Currently, there exist two test applications. `UDPEcho` is included in the official blip release and implements basic functionalities to response to a ping requests. The `webserver` application has been developed during this project and implements the RESTful API. Both can be found in the application directory `$LOWPAN_ROOT/apps/` of blip.

Before executing the make file to compile and install the sensor nodes, the make file may has to be modified. A preprocessor directive is used that the compiler connects blip with either the RF212 or the RF230 radio stack instead of the one for the CC2420. The following directives are supported:

- `RF212`: Defines that the RF212 radio stack should be used (MeshBean900 and Pixie).
- `RF230`: Defines that the RF230 radio stack should be used (IRIS).

No ID is used for the base station during the installation as the tunnel driver automatically configures this node. In general, the ID 100 ($0x64$) is allocated to the base station. For all other nodes, the ID defines the last block of the IPv6 address. For instance, a node installed with the ID 101 ($0x65$) obtains $0x65$ as the last block of its IPv6 address.

B.2.2 Executing a Network Test

After installing the base station, starting the tunnel driver and installing at least one sensor node with a test application, the sensor network can be tested. Standard networking tools like `ping6`, `tracert6` or `nc6` are supported. Assuming the configuration file of the base station has not been modified, you can ping a node with the ID 65 using the command `ping6 2001:470:1f04:56d::65`.

C

API Descriptions

C.1 RESTful API for Sensor Nodes

C.1.1 HTTP Methods

As explained in subsection 4.1.1, REST differs between requests for collections and for specific members of a collection. An HTTP request consists of a HTTP method and a URI. The HTTP method defines the action the node has to execute while the URI defines the collection or the member of the collection the request is addressed. The HTTP methods GET, PUT and DELETE are supported by this API. PUT and DELETE are only supported by members while GET is supported by members and collections.

GET Method

The GET method is used to receive the data offered by a collection or a member of a collection. It contains the method and the URI in the first line. The request is terminated with a blank line. The example outlined in Listening C.1 requests the parameters of a specific member. If the GET request is successful, the requested data are returned as a JSON object.

Listing C.1: A GET request to the collection `/management/info` to receive the parameters of the member `values`.

```
1 GET /management/info/values HTTP/1.1
2 <blank line>
```

PUT Method

A PUT method is used to update specific parameters of a member. An example of a PUT request is outlined in Listening C.2. It contains the method and the URI in the first line and each parameter is added on a new line. The request is terminated with a blank line. In this example, the parameters `name` and `place` are updated to new values. The data type of the parameters is defined in the answer to a GET request to the corresponding member. If the PUT request is successful, a `200 OK` status message is sent back to the client.

Listing C.2: A PUT request to the member `/management/info/values` to update the parameters `name` and `place`.

```
1 PUT /management/info/values HTTP/1.1
2 name: MySensor
3 place: ETZ F Floor
4 <blank line>
```

DELETE Method

The goal of the DELETE method is to reset the parameters. In the example outlined in Listening C.3, the delete request is used to turn off all leds, but concrete actions depend on the implementation of the member. If the DELETE request is successful, a `200 OK` status message is sent back to the client.

Listing C.3: A DELETE request to the member `/management/info/values` to reset its values.

```
1 DELETE /management/ledList/leds HTTP/1.1
2 <blank line>
```

Listing C.4: The response to a root request generated by the RESTful API.

```
HTTP/1.1 200 OK
Content-Length: 117

{
  "col": [
    "management/leds",
    "sensor/temperature",
    "sensor/light",
    "management/info",
    "mashup/coord",
    "mashup/info"
  ]
}
```

C.1.2 The Responses to a GET Request

The response to a GET request is formatted as JSON object. Depending on the type of the URI, the payload of the response is different. Each answer consists of the HTTP header and the payload. The header consists of the status line and the length of the HTTP payload. The HTTP payload is formatted as a JSON object and contains the data the user requested.

Root Collection

The response to a root request is outlined in Listening C.4. It consists of a JSON object with a single array. This array contains a list of all collections, which are supported by the sensor node. Consider the node with the IPv6 address `2001:470:1f04:56d::65`. The root collection can be accessed using the address `http://[2001:470:1f04:56d::65]/` while the collection `management/leds` can be accessed using the address `http://[2001:470:1f04:56d::65]/management/leds/*`.

Collection

The response to a request for a collection is constructed in the same way as the response to the root collection. The JSON object contains an array with all members of the corresponding collection. Listening C.5 outlines the response for a collection request. The corresponding collection contains the members `led0`, `led1` and `led2`.

Listing C.5: The response to a request for a collection. All members of the collection are listed in an array.

```
1 HTTP/1.1 200 OK
2 Content-Length: 035
3
4 {
5   "res": [
6     "led0",
7     "led1",
8     "led2"
9   ]
10 }
```

Member of a Collection

The response to a request for a collection member consists of three parts: the device name, the supported methods and the parameters. The first object of the JSON response is called "device" and its value is the name of the current member. In the example outlined in Listening C.6, the name is "Information". The object "method" describes the HTTP methods that are supported by this member. Arranged in an array, the following values are possible methods: "G" (GET), "U" (PUT) and "D" (DELETE).

The last object contains the parameters of the member with one entry for each member. The member itself is formatted as a JSON object and contains the name of the parameter ("n"), its value ("v"), its data type ("t") and whenever the parameter is updatable ("u"). Possible values for the data type are "b" for binary, "i" for integer, "f" for float and "s" for string.

C.2 Mashup API

In order that a device can be displayed on the world map of the mashup application, it has to implement two collections both with one member called `values`:

- `mashup/coord`: This collection has to provide the coordinates of the sensor node. Its member has to return the parameters `lat` and `long` defining the latitude and longitude coordinate.
- `/mashup/info`: By clicking on a marker on the world map, further information about the sensor node is displayed. All parameters of

Listing C.6: Example of a response to a GET request generated by a sensor node.

```
HTTP/1.1 200 OK
Content-Length: 203

{
  "device": "Information",
  "method": [
    "G",
    "U"
  ],
  "param": [
    {
      "n": "name",
      "v": "Device",
      "t": "s",
      "u": 1
    },
    {
      "n": "place",
      "v": "ETZ G Floor",
      "t": "s",
      "u": 1
    },
    {
      "n": "ID",
      "v": 101,
      "t": "i",
      "u": 0
    }
  ]
}
```

the according member are displayed on the pop-up window, but some parameters might be managed in a special way:

- `name` is interpreted as the name of the device,
- `place` is displayed as the current place of the sensor node and
- `type` is a code for a specific platform. For instance, type `1` means that the device is a MeshBean900.

D

Web Application

To provide a user-friendly interface for using the RESTful API, a Web application has been developed using GWT [40].

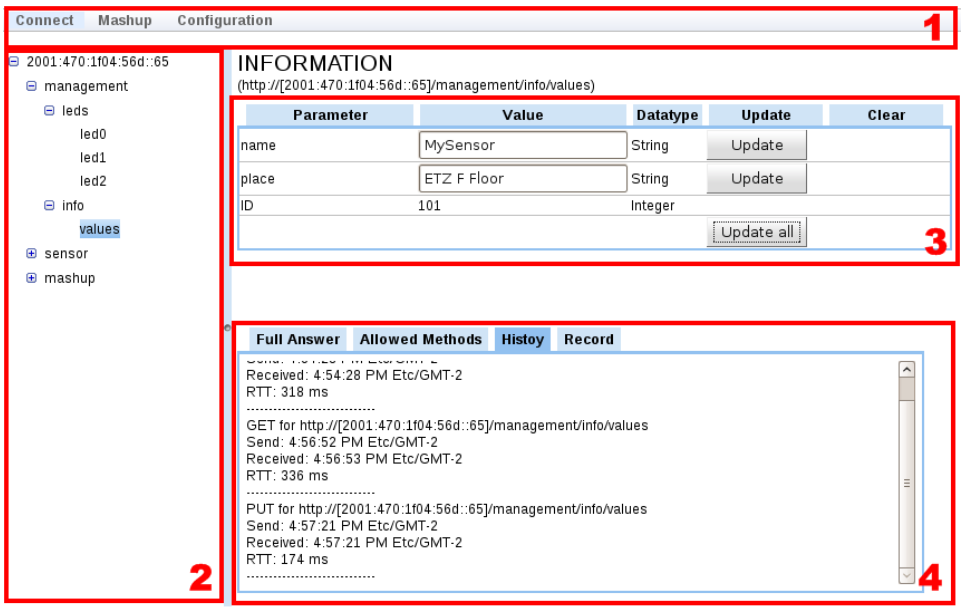


Figure D.1: Main screen of the Web application.

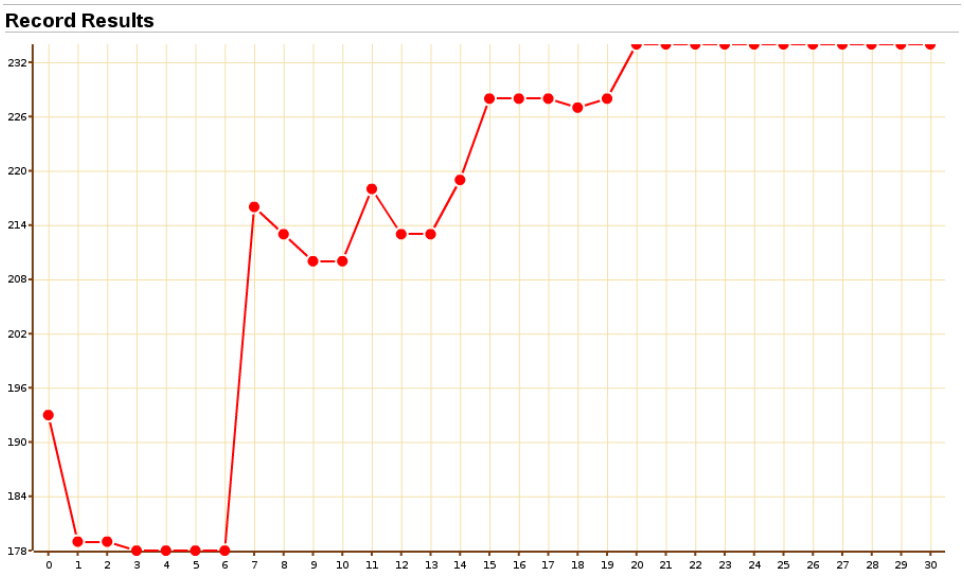


Figure D.2: Recording function of the Web application. The chart is automatically updated during recording.

Its main screen is outlined in Fig. D.1. It can be divided into four areas (also marked in Fig. D.1):

1. The *Menu Bar* contains the following buttons:
   - Connect: allows a user to connect to a specific sensor node,
   - Mashup: starts the mashup application and
   - Configuration: provides several options to control the behavior of the Web application.
2. The *Connection Bar* displays all sensor nodes to which the user is connected. By clicking on a device address, all collections are shown. The same applies to a collection while by clicking on a member, its parameters are retrieved and displayed on the right.
3. The *Parameter Table* outlines all parameters of the member and allows the modification of these parameters.
4. The *Detail Area* contains four tabs: the full JSON answer, a list with the supported methods of the member, a list with all requests sent to this member and a tab for recording. The latter allows the selection of a parameter to be recorded and a chart of the recorded values can be displayed (see Fig. D.2).

Bibliography

[1] "ITU Internet Reports 2005: The Internet of Things," International Telecommunication Union (ITU), Nov. 2005.

[2] V. Stirbu, "Towards a RESTful Plug and Play Experience in the Web of Things," Aug. 2008, pp. 512–517.

[3] "ZigBee Alliance," Jun. 2009. [Online]. Available: http://www.zigbee.org/

[4] "HART Communication Foundation (HCF)," Jun. 2009. [Online]. Available: http://www.hartcomm.org/

[5] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking (MobiCom)*, New York, NY, USA, 1999, pp. 263–270.

[6] *Das Internet der Dinge*. Springer Berlin Heidelberg, 2005. [Online]. Available: http://www.springerlink.com/content/978-3-540-24003-7

[7] A. Dunkels, "Full TCP/IP for 8 Bit Architectures," in *Proceedings of the First ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys 2003)*, San Francisco, May 2003. [Online]. Available: http://www.sics.se/~adam/mobisys2003.pdf

[8] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944 (Proposed Standard), Internet Engineering Task Force, Sep. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4944.txt

[9] M. Harvan and J. Schönwälder, "A 6lowpan Implementation for TinyOS 2.0," in *6th GI/ITG KuVS Fachgespräch "Wireless Sensor Networks"*, Aachen, Jul. 2007.

[10] S. Dawson-Haggerty, "Berkeley IP Information," Mar. 2009. [Online]. Available: http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip

[11] X. Jiang, S. Dawson-Haggerty, P. Dutta, and D. Culler, "Design and Implementation of a High-Fidelity AC Metering Network," in *Proceedings of 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, San Francisco, USA, Apr. 2009.

[12] A. Dunkels, "IP Networking and Web Server for the TelosB/Tmote Sky," Tutorial, Jun. 2008. [Online]. Available: http://www.sics.se/contiki/tutorials/tutorial-ip-networking-and-web-server-for-the-tmote-sky.html

[13] D. Guinard and V. Trifa, "Towards the Web of Things: Web Mashups for Embedded Devices," in *Proceedings of WWW (International World Wide Web Conferences), Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009)*, Madrid, Spain, Apr. 2009.

[14] D. Guinard, V. Trifa, T. Pham, and O. Liechti, "Towards Physical Mashups in the Web of Things," in *Proceedings of INSS 2009 (IEEE Sixth International Conference on Networked Sensing Systems)*, Pittsburgh, USA, Jun. 2009.

[15] "TinyOS," 2009. [Online]. Available: http://www.tinyos.net/

[16] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An Operating System for Sensor Networks," in *Ambient Intelligence*, 2005, pp. 115–148.

[17] *ZigBit 700/800/900 MHz Wireless Modules*, Atmel, Apr. 2009.

[18] *AT86RF212 Datasheet*, Atmel, Jun. 2008.

[19] "IEEE Std 802.15.4-2006," IEEE Computer Society, Sep. 2006.

[20] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460 (Draft Standard), Internet Engineering Task Force, Dec. 1998, updated by RFC 5095. [Online]. Available: http://www.ietf.org/rfc/rfc2460.txt

[21] W. Stallings, "IPv6: The New Internet Protocol," *IEEE Communications Magazine*, vol. 34, no. 7, pp. 96–108, Jul. 1996.

[22] S. Bradner and A. Mankin, "The Recommendation for the IP Next Generation Protocol," RFC 1752 (Proposed Standard), Internet Engineering Task Force, Jan. 1995. [Online]. Available: http://www.ietf.org/rfc/rfc1752.txt

[23] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," RFC 4291 (Draft Standard), Internet Engineering Task Force, Feb. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4291.txt

[24] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)," RFC 3315 (Proposed Standard), Internet Engineering Task Force, Jul. 2003, updated by RFCs 4361, 5494. [Online]. Available: http://www.ietf.org/rfc/rfc3315.txt

[25] S. Thomson and T. Narten, "IPv6 Stateless Address Autoconfiguration," RFC 2462 (Draft Standard), Internet Engineering Task Force, Dec. 1998, obsoleted by RFC 4862. [Online]. Available: http://www.ietf.org/rfc/rfc2462.txt

[26] A. Conta, S. Deering, and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC 4443 (Draft Standard), Internet Engineering Task Force, Mar. 2006, updated by RFC 4884. [Online]. Available: http://www.ietf.org/rfc/rfc4443.txt

[27] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861 (Draft Standard), Internet Engineering Task Force, Sep. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4861.txt

[28] J. W. Hui and D. E. Culler, "IP is Dead, Long Live IP for Wireless Sensor Networks," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*. New York, NY, USA: ACM, 2008, pp. 15–28.

[29] A. Dunkels and J. Vasseur, "IP for Smart Objects," Internet Protocol for Smart Objects (IPSO) Alliance, White Paper #1, Sep. 2008.

[30] "IPSO Alliance: Promoting the use of IP for Smart Objects," 2009. [Online]. Available: http://www.ipso-alliance.org

[31] N. Kushalnagar, G. Montenegro, and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals," RFC 4919 (Informational), Internet Engineering Task Force, Aug. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4919.txt

[32] J. Hui, D. Culler, and S. Chakrabarti, "6LoWPAN: Incorporating IEEE 802.15.4 into the IP Architecture," Internet Protocol for Smart Objects (IPSO) Alliance, White Paper #3, Jan. 2009.

[33] "Routing Over Low Power and Lossy Networks (ROLL) Working Group," Internet Engineering Task Force (IETF), Apr. 2009. [Online]. Available: http://www.ietf.org/html.charters/roll-charter.html

[34] S. Dawson-Haggerty, "b6lowpan Stack Walkthrough," Presentation, Oct. 2008.

[35] "SourceForge.net Repositiory - TinyOS 2.0," Jun. 2009. [Online]. Available: http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x/

[36] *Chipcon AS SmartRF CC2420 Preliminary Datasheet*, Chipcon, Jun. 2004.

[37] "Twitter: What are you doing?" Jun. 2009. [Online]. Available: http://twitter.com/

[38] R. T. Fielding and R. N. Taylor, "Principled Design of the Modern Web Architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, 2002.

[39] D. Crockford, "The application/json Media Type for JavaScript Object Notation (JSON)," RFC 4627 (Informational), Internet Engineering Task Force, Jul. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4627.txt

[40] "Google Web Toolkit," Jun. 2009. [Online]. Available: http://code.google.com/webtoolkit/

[41] "pachube :: Connecting Environments, Patching the Planet," Jun. 2009. [Online]. Available: http://www.pachube.com/

[42] "National Data Buoy Center," Jun. 2009. [Online]. Available: http://www.ndbc.noaa.gov/obs.shtml

[43] "Google Maps API," Jun. 2009. [Online]. Available: http://code.google.com/apis/maps/

[44] N. Matsumoto, "TCP blip," Mar. 2009. [Online]. Available: http://smote.cs.berkeley.edu:8000/tracenv/browser/code/tcpblip

[45] C. Aoun and E. Davies, "Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status," RFC 4966 (Informational), Internet Engineering Task Force, Jul. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4966.txt

[46] J. Hagino and K. Yamamoto, "An IPv6-to-IPv4 Transport Relay Translator," RFC 3142 (Informational), Internet Engineering Task Force, Jun. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3142.txt