Network Architectures
and Services
NET 2013-07-2

Dissertation

Secure Data Transmission in Wireless Sensor Networks

Corinna Schmitt

Network Architectures and Services
Department of Computer Science
Technische Universität München

TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Secure Data Transmission in Wireless Sensor Networks

Dipl.-Inf. (Bioinformatik) Corinna Schmitt

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. Johann Schlichter
Prüfer der Dissertation:	1.	Univ.-Prof. Dr.-Ing. Georg Carle
	2.	Univ.-Prof. Dr. Uwe Baumgarten

Die Dissertation wurde am 13.02.2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 03.06.2013 angenommen.

Cataloging-in-Publication Data
Corinna Schmitt
Secure Data Transmission in Wireless Sensor Networks
Dissertation, July 2013
Network Architectures and Services, Department of Computer Science
Technische Universität München

ISBN: 3-937201-36-X
ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
DOI: 10.2313/NET-2013-07-2
Network Architectures und Services NET 2013-07-2
Series Editor: Georg Carle, Technische Universität München, Germany

Abstract

Today a growing number of applications, in particular in the area of cyber-physical systems (CPS), depend on data collected by wireless sensor networks. The individual nodes of these networks have severe resource limitations, concerning storage, processing, and transmission capabilities, but also concerning energy available for communication functions. Supporting the required protocol functionality in combination with the goal of energy-saving operation typically results in the development of proprietary, highly specialized communication protocols for wireless sensor networks.

However, with Internet technology as the dominating communication paradigm for a wide range of application areas, it became an attractive goal to be able to use Internet protocols within wireless sensor networks. An important functionality within wireless sensor networks is data gathering and aggregation. An Internet protocol standardized within IETF that supports this functionality is the 'IP Flow Information Export' (IPFIX) protocol.

In this thesis, concepts for adapting the IPFIX protocol to the needs of wireless sensor networks have been investigated, resulting in the development of the protocol TinyIPFIX, which is an adaptation of the 'IP Flow Information Export' (IPFIX) protocol. The new protocol has been assessed in a representative use case involving a building application. TinyIPFIX has been extended by compression capabilities and by aggregation functionality. Furthermore, extensions to support secure data transmission have been developed, using the protocol 'Datagram Transport Layer Security' (DTLS). This solution ensures that data collected by sensor nodes is transmitted via secure channels to a global data sink, and that authorised access is ensured from a data sink to a wireless sensor network. For validation, a system has been realized that allows configuration of the networks components dynamically, and that supports visualization of the current network status and the collected data in real time.

Zusammenfassung

Eine wachsende Anzahl von Applikationen, insbesondere im Bereich von cyber-physischen Systemen (CPS), ist auf die Datensammlung mittels drahtlosen Sensornetzen angewiesen. Die eingesetzten Sensorknoten sind stark in ihren Ressourcen hinsichtlich Speicher, Verarbeitungs- und Übermittlungskapazitäten eingeschränkt. Zusätzlich ist die verwendete Hardware in der zur Verfügung stehenden Energiereserve für Kommunikationszwecke limitiert. Um die angeforderten Protokollfunktionalitäten in Kombination mit dem Ziel der Energieeinsparung zu unterstützen, müssen für drahtlose Sensornetze hoch spezialisierte Kommunikationsprotokolle entwickelt werden.

Auf Grund der heutigen Anbindung vieler Anwendungsbereiche an das Internet wird es für die Forschung immer interessanter, existierende Internetprotokolle für die Gegebenheiten in drahtlosen Sensornetzen anzupassen und dort zu etablieren. In drahtlosen Sensornetzen stellen die Datensammlung und die Datenaggregation die grundlegenden Aufgaben dar. Das sogenannte „IP Flow Information Export“ (IPFIX) Protokoll ist ein bei der Internet Engineering Task Force (IETF) standardisiertes Internetprotokoll, welches die von Sensornetzen geforderte Funktionalität unterstützt.

In dieser Arbeit werden Konzepte zur Anpassung des IPFIX Protokolls an die Bedürfnisse von drahtlosen Sensornetzen untersucht. Auf diesen Ergebnissen basierend, wurde das Protokoll TinyIPFIX entwickelt, welches eine Adaption des IPFIX Protokolls darstellt. Dieses neue Protokoll wurde evaluiert und in einem repräsentativen Anwendungsfall für drahtlose Sensornetze (in einem Gebäudeszenario) angewendet. TinyIPFIX wurde erweitert durch Kompressionsfähigkeiten und Aggregationsfunktionalitäten. Weiterhin wurden Erweiterungen integriert, die einen sicheren Datentransport unter Verwendung des „Datagram Transport Layer Security“ (DTLS) Protokolls unterstützen. Diese Lösung gewährleistet, dass durch Sensorknoten gesammelte Daten über einen gesicherten Kanal zur globalen Datensenke übermittelt werden können. Weiterhin wird durch diese Lösung ein autorisierter Zugriff von der Datensenke zum drahtlosen Sensornetz garantiert. Zur Validierung wurde ein System realisiert, welches eine dynamische Konfiguration der Netzwerkkomponenten erlaubt und gleichzeitig die Visualisierung des aktuellen Netzwerkstatus und der gesammelten Daten in Echtzeit unterstützt.

Acknowledgements

In order to develop a doctoral thesis, support of many people is required.

Prof. Georg Carle, who supervised this work, gave me the initial input upon which my ideas were developed, and allowed me to establish the wireless sensor network research again at his chair 'Network Architectures and Services'.

Apart from my colleagues and advisor, I would like to thank my students Thomas Kothmayr, Benjamin Ertl, and Andre Freitag for supporting me in the practical part and providing invaluable contribution to the success of this protocol implementations for wireless sensor networks.

Dr. Wen Hu and Alexandra Pischel read the whole thesis, and provided lots of valuable feedback concerning different aspects.

I also want to thank my sister Cornelia and my friend Alexander for always supporting me, even in busy times.

To conclude, I would like to thank my parents Brigitte and Dr. Hans-Bernd Schmitt for their continous support throughout the many years of my education, studies and research, for their wisdom and for their generosity. To them I owe it all.

Publications

Corinna Schmitt, Burkhard Stiller, Thomas Kothmayr, and Wen Hu
DTLS-based Security with two-way Authentication for IoT
Internet-Draft (work in progress), draft-schmitt-two-way-authentication-for-iot-00.txt
Internet Engineering Task Force (IETF)
July 2013

Thomas Kothmayr, Corinna Schmitt, Wen Hu, Michael Brünig, and Georg Carle
DTLS based Security and Two-Way Authentication for the Internet of Things
In Journal: ELSEVIER Special Issue on Security, Privacy and Trust Management in the Internet of Things era (SePriT)
doi: 10.1016/j.adhoc.2013.05.003
June 2013

Thomas Kothmayr, Corinna Schmitt, Wen Hu, Michael Brünig, and Georg Carle
A DTLS based End-to-End Security Architecture for the Internet of Things with Two-Way Authentication
In Proceedings of the 7th IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp)
Clearwater, USA, October 2012

Thomas Kothmayr, Wen Hu, Corinna Schmitt, Michael Brünig, and Georg Carle
Securing the Internet of Things with DTLS
In Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys), Poster Session
Seattle, USA, November 2011

Lothar Braun, Corinna Schmitt, Benoit Claise, and Georg Carle
Compressed ipfix for smart meters in constrained networks
Internet-Draft (work in progress), draft-braun-core-compressed-ipfix-03.txt
Internet Engineering Task Force (IETF)
September 2011

Alexander Klein, Lothar Braun, Corinna Schmitt, and Georg Carle
MAUS: A Multi-hop Autonomous Sensor Network for Monitoring Applications with Full IP-support
In Proceedings of the 9. GI/ITG KuVS Fachgespräch Sensornetze (FGSN)
Würzburg, Germany, September 2010

Corinna Schmitt, Lothar Braun, Thomas Kothmayr, and Georg Carle
Collecting Sensor Data using Compressed IPFIX
In Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Poster Session
Stockholm, Sweden, April 2010

Thomas Kothmayr, Corinna Schmitt, Lothar Braun, and Georg Carle
Gathering Sensor Data in Home Networks with IPFIX
In Proceedings of the 7th European Conference on Wireless Sensor Networks (EWSN)
Coimbra, Portugal, February 2010

Corinna Schmitt and Georg Carle
Applications for Wireless Sensor Networks
Chapter in Book: Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications
Antonopoulos N.; Exarchakos G.; Li M.; Liotta A. (Eds.)
Information Science Publishing
January 2010

Contents

1. Introduction

Some decades ago the Internet was connecting only thousands of computers. Due to its growth, the Internet now connects several billions of computers. Today the connection is not limited to computers any more; it also includes objects such as mobile devices (e.g. smart phones, PDAs). In 2005 the International Telecommunication Union (ITU) published the report 'ITU Internet Reports 2005: The Internet of Things' [1], which focused on the recent developments around the *Internet of Things* (IoT) at this time [1]. The report pointed out that the term *Internet of Things* is linked to Kevin Ashton (executive director Audio-ID Center) who first mentioned this term in 1999 during a presentation by Procter and Gamble dealing with radio frequency identification [1, 2]. The characteristics of devices, used in the *Internet of Things*, is manifold and can include the following ones as pointed out in the report by ITU: own Internet Protocol addresses, support of wireless communication, embedding into complex systems, involvement of sensor technology to collect information, and/or involvement of actuators to interact with the environment [1]. As can be seen, devices of the *Internet of Things* are of heterogeneous structure, so that the *Internet of Things* only requires the capability of IP communication support. The included devices can be assigned to different groups, among others in Web of Things (WoT) and smart dust [3, 4]. Devices of the group WoT are embedded components and are connected to other objects via Internet by using web standards, such as the Hyper Text Transfer Protocol (HTTP) [5]. Smart dust represents devices building a network of very small wireless microelectronic physical objects, such as radio frequency identification (RFID) [6]. The intersection of the two groups is represented by wireless sensor networks that build a network of low complexity sensors with constrained hardware and communicate wirelessly [5, 4].

In the research field of the *Internet of Things* one traditional focus is on routing, because independent of the application the main goal was to support transmissions from the data collection point (source) to the next hop towards the final destination e.g. data sink. It is assumed that an optimal route should be fast, cost-efficient, and reliable. The developed routing protocols are based on ideas known from ad-hoc, IP or Peer-to-Peer networks. Another long standing research question deals with security for the *Internet of Things*. Comparable strategies from the Internet or web service area are the 'Secure Sockets Layer' (SSL) or 'Transport Layer Security' (TLS) protocol for security support in both areas; HTTP is used for communication pur-

poses in the Internet and Simple Object Access Protocol (SOAP) for data exchange purposes between systems and for remote procedure calls (e.g. web services). Yet, no solutions, that are equivalently secure, exist for the *Internet of Things*, especially not for wireless sensor networks. This thesis investigates and solves, therefore, the security problems arising from wireless sensor networks only focusing on key mechanisms and data encryption rather than securing the transmission channel [7, 8, 9].

In this dissertation it is assumed that very constrained hardware communicates with each other as part of the *Internet of Things*. The devices used in wireless sensor networks represent a special class of this constrained hardware. They are very limited concerning energy, memory, and computational capacity. Their main goal is to collect information and transmit it into the direction of sinks. The sink serves as gateway that has a connection to the Internet. In most literature for wireless sensor network applications the term *gateway* is used for the *device complex* sink and server. In this case, the sink is the last component that communicates wirelessly with the network components and is wired to a server (e.g. PC) that has an Internet connection [10, 11]. The previously mentioned setup, called *device complex,* is required, because it is unusual that sensor nodes have a direct connection to the Internet. Servers are not resource-limited, and, therefore, can support required protocols and security mechanisms [10]. Due to the growing application field of wireless sensor networks, the functionality supported by the hardware grows, but resources are still limited and can restrict the support of the functions. Today it is common that research focuses on one application scenario developing a special solution. In order to transfer the developed solution to other application scenarios, researchers try to establish a generally applicable solution that can be adapted with less input to new requirements (e.g. hardware changes). A prominent field of application is the area in which different devices (e.g. sensor nodes, management units) collect information and exchange it in order to react depending on the analysis results.
An example is a building scenario, a representative of a cyber-physical system (CPS) [12], which provides the basis of this thesis. A cyber-physical system is a combination of physical elements and system's computational elements. In the case of the assumed building scenario in this dissertation different sensors collect environmental data (e.g. light, temperature, humidity) and transmit it to a management unit (e.g. climate control, energy control). This management unit processes the data by applying pre-defined knowledge (e.g. defined room temperature). Depending on the analysis result the management unit activates the corresponding components in order to perform a special action (e.g. to turn on/off heating/light). [13]
Other application examples for cyber-physical systems can be found among others in the area of health care, logistic or behavior monitoring. In general, the previously mentioned application scenarios are characterized by heterogeneous hardware usage, flexibility requests, and different network sizes. [10, 12]

1.1 Problem Statement

As previously mentioned, it is required to integrate wireless sensor networks into the *Internet of Things*. It is assumed that sensor devices have enough resources to support IP communication. An overview of IP solutions for sensor devices is presented, e.g. ZigBee, IPv6 over Low power Wireless Personal Area Network (6LoWPAN), and the Berkeley Low-power IP stack (BLIP) representing the underlying stack (cf. Section 3.1). Due to the limited resources of sensor hardware, especially energy, memory, and computational capacity, supported solutions must be developed in a

resource-efficient way.
In order to support interoperability of protocols and heterogeneity in the hardware, the developed solutions must be flexible and independent of the underlying stack (e.g. 6LoWPAN, BLIP), which is i.a. responsible for the network and transport functionalities. As a consequence the integration of protocols and algorithms on the application layer with focus on standardization is preferred as solution. The application layer is located separately above a complex (called *network stack*) that includes all underlying layers (e.g. transport, network, physical). The protocols and algorithms in the application layer communicate with the underlying layers via interfaces (cf. Section 3.1). For example, if a protocol using 6LoWPAN on the underlying layer is developed and ZigBee is now exchanged by BLIP, it is only required to modify the interfaces between the application layer and the underlying stack but not the application itself. Instead of developing new isolated solutions, this dissertation focuses on the analysis of existing standards, for example of IP networks, and the possibility for application and protocol transfer to constrained hardware, such as sensor nodes, and transfers advantageous characteristics (e.g. message format, energy efficiency) into one combined protocol - *TinyIPFIX* and its extensions.

In conclusion, this dissertation develops a secure, efficient and standardized communication solution for constrained devices in the *Internet of Things*. The solutions' objective is to be able to be scaled on different network sizes, to be flexible, and applicable on very constrained hardware.

1.2 Research Questions

The research questions answered in this dissertation focus on efficient data transmission in wireless sensor networks with support of a security solution. Security considerations are required, because of the integration of wireless sensor networks into the *Internet of Things* and because of the connection between data and personal/sensible information (e.g. GPS). In order to develop a resource-efficient solution the requirements of wireless sensor networks, which are presented in Chapter 2, are kept in mind. The developed solution should support different hardware platforms and allow integration of new features into a modular architecture.

1.2.1 Efficiency

Due to the constrained hardware used in wireless sensor networks, as presented in Chapter 2, resource efficiency is very important. The hardware has limited computational capacity, memory, and energy. In order to save energy, which is the most limited resource, data sensing and processing, especially transmission, must be optimized, because it requires most of the energy of a sensor node [14, 15, 16, 17]. It is the goal to transmit as much information as possible in one packet. As described in Section 3.1 the supported maximum transmission unit (MTU) of the used hardware's RF transceiver CC2420 is limited to 127 bytes [18]. The finally available MTU for individual data (e.g. measured sensor data) becomes more limited due to additional headers (e.g. TinyOS message header and overhead by IP communication as shown in Figure 5.4). Thus, an efficient transmission protocol must be used in order to achieve the goal to put as much data payload as possible into one packet. This reduces the number of transmissions of a sensor node and helps to save energy, which is the most limited resource.

This dissertation deals with the following research questions in the field of efficient data transmission:

(E1) Is the IP Flow Information Export (IPFIX) protocol a viable solution for transmission of sensor data in wireless sensor networks?

(E2) Is it possible to combine data pre-processing techniques (e.g. aggregation) with the IPFIX protocol within the network?

(E3) Can all sensor platforms perform TinyIPFIX and its extensions (compression, aggregation), as well as TinyDTLS?

Research question E1 is answered in Section 3.2 which focuses on the efficient transmission protocol 'IP Flow Information Export'. The IPFIX protocol is characterized by a push-protocol behavior and a template-based message design [19, 20]. Together with the characteristics of the data transmission protocols used in wireless sensor networks, as presented in Section 4.1.1, the main issue for efficient data transport is the separation of meta information and data, which is done by the IPFIX protocol. The IPFIX protocol is a known standard from IP networks and can be adapted to sensor network requirements with some modifications as introduced in Section 4.1.2. This section also presents compression techniques in order to get along with the additional overhead caused by additional headers (IPFIX message and Set header). In order to optimize the efficiency of data transmission in wireless sensor networks (cf. research question E2) aggregation techniques are integrated into the wireless sensor network, as introduced in Section 4.2. The term *aggregation* includes two functionalities. One functionality is *message aggregation* where two or more individual messages are compressed into one combined packet without any modification within each message itself. The second functionality deals with pre-processing and is called *data aggregation*. This means, on a special point (node) within the network messages are collected and special information (e.g. temperature value) is extracted from the payload. This extracted information is pre-processed (e.g. MIN, MAX, AVG calculation of temperature) and the calculation result is transmitted to the next hop in one message. The implementation of the developed data transmission protocol TinyIPFIX with its extensions (e.g. compression, aggregation) is studied in Sections 5.1 and 5.2, followed by the evaluation in Sections 6.1 and 6.2, which answers research question E3.

1.2.2 Security

Due to the integration of wireless sensor networks into the *Internet of Things* and the relation between data and private/sensible information a secure data transmission is very important. Today, applications of wireless sensor networks prefer to use the unreliable 'User Datagram Protocol' (UDP) instead of the reliable 'Transmission Control Protocol' (TCP). As described by Wagenknecht et al., todays' deployed wireless sensor networks use UDP for data transmission towards a sink and TCP is used for administrative functions (e.g. node configuration) [21]. If TCP would be used, the standard TLS protocol could be used. It requires that both communication endpoints support the same network layer. Another problem for using TLS over UDP is the missing authentication of packets if packet loss occurs which is possible when UDP is used.

In order to support secure data transmission between participating sensor nodes, this dissertation deals with the following research questions in the field of security:

(S1) Is it possible to secure data transmission in wireless sensor networks with known standards from IP networks?

(S2) Can DTLS be performed on severely resource constrained hardware as used in wireless sensor networks?

In order to evaluate existing standards that are relevant for wireless sensor networks an overview is given in Section 3.3 answering research question S1. The presented standards in Section 3.3 provide confidentiality, integrity and authorization support, as well as strategies against different attacks and defense strategies. Those standards and existing security solutions for wireless sensor networks influenced the design decisions for the developed solution in this dissertation presented in Section 4.3. Section 4.3.2 focuses on research question S2 in order to secure data communication in wireless sensor networks by using DTLS, which provides server authentication, confidentiality, and message integrity, as well as protection against replay attacks [22, 23]. A brief introduction to the implementation is given in Section 5.3, followed by a detailed evaluation in Section 6.3.

1.3 Structure of this Dissertation

The remainder of this dissertation uses the terms *sensor node* and *mote* as synonyms, as well as *building scenario*, *home scenario* and *office scenario*. It is organized as follows:

Chapter 2 discusses the background information related to wireless sensor networks which is required in order to get a feeling about how challenging the constrained hardware used in wireless sensor networks is. The chapter starts with the introduction of design principles in order to establish a high standard of quality of service (QoS), energy efficiency, and scalability of the performed functionalities (c.f. Section 2.1). It has to be kept in mind that design principles are always in global scope and their characteristic depends on the used hardware and the selected application. The next parts of this chapter focus on the characterization of sensor node hardware in order to give a feeling of their constraints. In general, the sensor node hardware is very limited in memory, computational capacity, and energy resources. Because of those constraints not every hardware can be used in every application and/or perform the same functionalities (e.g. aggregation, encryption), as well as energy saving methods become interesting in order to support long lifetime of the system. In Section 2.4 a brief overview of the most common operating systems in wireless sensor networks is given. The operating system TinyOS is driven by the academic side, whereas the operating system Contiki is driven by the industry. Both operating systems can be used for common sensor platforms (e.g. TelosB, OPAL), but at the same time not every operating system gets along with all platforms (e.g. TinyOS supports IRIS but Contiki does not). The choice of the operating system is influenced by the used hardware and the preferences of the developers of the wireless sensor network. For both operating systems the basic idea is the same: Keep protocols/algorithms as simple as possible in order to get along with the limited resources of the hardware. In addition, they try to support a modular structure in order to add/delete/exchange parts without deep impact to other protocol/algorithm parts. In order to support all kind of sensor platforms, it becomes interesting to transfer developed protocols between operating systems. In the case of TinyOS and Contiki a brief description is given what the requirements are. Finally,

this chapter closes with an introduction of today's relevant application scenarios in Section 2.5. The application groups, however, can be grouped differently. In this dissertation a grouping concerning mobility of sensor nodes and gateway is performed, followed by a handful of available applications for each group.

Chapter 3 covers background information of related standards that are interesting for wireless sensor networks in order to support integration into the *Internet of Things*. The first step is to support IP communication with constrained hardware. Sections 3.1 briefly describes the history concerning stack development for IP communication on constrained hardware. The drawback of IP communication support is the added overhead by required IP headers. Thus, it becomes important to focus on efficient data transport functionality in order to transport as much relevant data (e.g. sensor data) as possible in a maximum transmission unit. In this dissertation the used RF transceiver CC2420 limits the maximum transmission unit to 127 bytes [18]. From IP networks the efficient data transmission protocol 'IP Flow Information Export' (IPFIX) comes to mind. It is known from monitoring tasks in IP networks and has an interesting message structure in order to reduce redundancy in the messages by separation of meta information and data in different messages. The IPFIX protocol will briefly be introduced in Section 3.2 pointing out its advantages and why it is interesting to adapt the protocol to the requirements of wireless sensor data. Section 3.3 focuses on related standards for security support. Due to the connection to the Internet, it is an attractive aim for attackers. The problem is that transmitted data can include sensitive data (e.g. personal information) that must be kept private. Thus, the network and the data must be secured. In order to find an appropriate method for this task this section defines terms such as security attack, security services, and security mechanisms. Further, this section introduces standard security solutions (e.g. cryptographic functions, public key infrastructure, (D)TLS protocol, trusted platform module) that influence the design decisions for the implemented security solution in this dissertation.

Chapter 4 requires the previously introduced background information about wireless sensor networks and related standards from IP networks in order to justify the design decision for the upcoming transfer of selected standards and required modifications for wireless sensor networks. For further purposes it is assumed from this point on that the used sensor data supports IP communication using the introduced BLIP stack. As introduced before the hardware components of the wireless sensor network used in this dissertation are very limited in memory, energy, and computational capacity. Therefore, the supported functionality of the individual components varies. First, the focus of this chapter is on efficient data transport realized by transferring the message structure of the IPFIX protocol to the requirements of sensor nodes (cf. Section 4.1). The result is the developed TinyIPFIX protocol, which separates the transmitted data into messages including only meta information or only data. Because of possible route updates caused by the BLIP protocol and the used unreliable transmission protocol UDP, meta information is repeated periodically in order to guarantee the translation of the collected sensor data at all points in the network. The IPFIX protocol adds additional overhead to the already limited message size, which reduces the original goal to send as much information (sensor data) as possible in one message. Thus, compression techniques for the IPFIX headers (IPFIX message and Set header) are designed which add a pre-header to the message that verifies the size of each field in the IPFIX header. As a result, the overhead caused by IPFIX support can be reduced by 85% and leaves

enough space for the original goal. In order to reduce the traffic within the network itself, an aggregation protocol 'TinyIPFIX-Aggregation' is designed, as specified in Section 4.2. It supports message aggregation and data aggregation. The latter allows pre-processing of data within the network. In order to support a secure data transmission within the sensor network, especially in direction to the gateway, a DTLS solution is designed (cf. Section 4.3). The DTLS standard is adapted to the requirements of the sensor node OPAL, which includes a trusted platform module for authentication purposes. Independent of the protocol transfers a graphical user interface (GUI) was designed in order to configure the wireless sensor network and to receive visual feedback about the system's status and the collected information. The graphical user interface is briefly characterized in Section 4.4, which is based on a virtual representation of the real sensor network.

As mentioned in Chapter 2 the resources of the used sensor hardware are limited, as well as the supported maximum transmission unit by the chosen operating system TinyOS. In order to establish a better understanding for the evaluation of the implemented solutions **Chapter 5** focuses on the implementation of TinyIPFIX and its extensions as well as the graphical user interface. During the explanation of the design decisions it was pointed out that the development of solutions also depends on the chosen application scenario. Therefore, a building scenario is assumed in this dissertation. With the help of this scenario different environmental conditions could be tested that influenced the performance of the implemented solution (e.g. wall structures, distances between sensor nodes). This scenario also provides an intuitive idea why secure data transmission is required, in order to ensure only allowed access to data, especially using authentication mechanisms. Concerning the developed graphical user interface this chapter focuses on the required workflows in order to support configuration functionality of the network components, the visualization of the network status, and different possibilities to import/export data.

After presenting detailed background information about the design decisions and the implementation the developed secure data transmission solution for wireless sensor networks presented in this dissertation is evaluated in **Chapter 6**. Due to the requirements of wireless sensor networks, it is a challenge to integrate them into the *Internet of Things* without loosing their main tasks (e.g. data collection and forwarding) out of scope. The developed solution - TinyIPFIX and its extensions - deals with an efficient data transport possibility, containing a template-based message design and support of pre-processing (aggregation) of data within the network itself, and the establishment of a secure data transmission possibility (cf. Sections 6.1 to 6.3). In this chapter the presented solutions are evaluated concerning their requirements for memory, transmission efficiency, and energy consumption. Further, a comparison of the established solution with related standards is presented. It will be pointed out that not every sensor platform is able to support all functionalities, but all support IP communication. This regulation is mainly based on the required memory of the solutions (e.g. performing aggregation) or because the solution requests special hardware (e.g. trusted platform module). In order to support an user-friendly environment to manage the wireless sensor network a graphical user interface is developed. The graphical user interface is running on the server side, so that no regulations concerning memory, computational capacity or energy exist. Thus, the evaluation of the graphical user interface is a proof of concept showing examples for hardware configuration, feedback of network status, and data import/export (cf. Section 6.4). Chapter 6 is concluded with compliance proof of

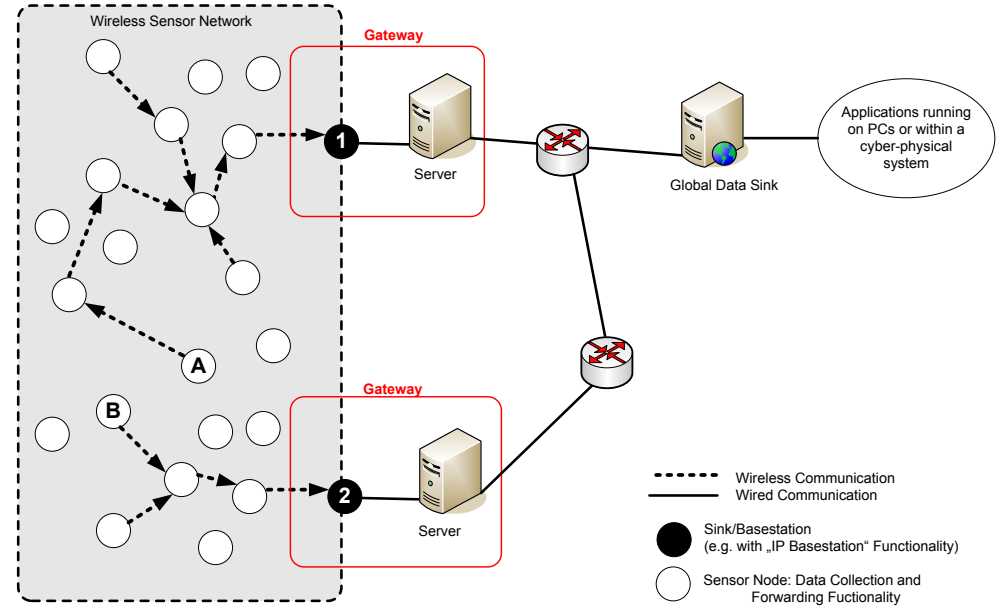
the requirements for a cyber-physical system focusing on standardization, resource efficiency, flexibility, and usability of the developed solutions in this dissertation.

Finally, this dissertation is concluded in **Chapter 7**. It briefly summarizes the faced challenges for the developed solutions in order to integrate a wireless sensor network into the *Internet of Things* with a building scenario as application example. Furthermore, the advantages of the developed TinyIPFIX protocol with its extensions is resumed. Next, the mentioned research questions in Section 1.2, focusing on efficiency and security, are answered in order to summarize the research impact of this dissertation. Last but not least, an outlook is given for further investigations in order to improve the established wireless sensor network with all its components and features.

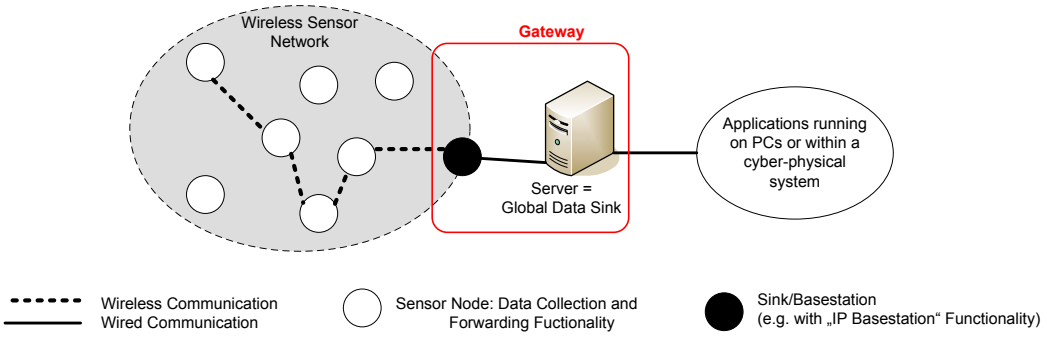
2. Fundamentals of Wireless Sensor Networks

In general, a wireless sensor network consists of different sensor nodes (white) from different vendors which collect individual data and transport it to one or more sink(s) (black) as illustrated in Figure 2.1a (analog to references [24, 10]). The number of participating nodes depends on the network size and may cause a long communication link to the sink with several hops in between. Depending on the placement of the sensor nodes, some nodes are not able to communicate with the sink directly. To overcome this, the network must support routing functionality in order to route data packets to the sink in an efficient way. Therefore, the routing algorithms take the current network status into account in order to be able to calculate the optimal route towards the sink. For example, depending on the location of sensor node *A* it would be expected that the routing algorithm would prefer to forward the collected data to sink 2 and further to the global data sink. Instead, the underlying routing algorithm (e.g. BLIP) decides to route the data over five hops to sink 1 using direct addressing of the sink by its individual IP address. From sink 1 the data is forwarded using LAN connections to the global data sink. In both cases the data would arrive the global data sink but the required time might be different.
The sink is a sensor node with gateway functionality (e.g. IP Basestation[1]). It forwards the wireless received data to a wired infrastructure, such as a server or PC, which is responsible for the further handling of the data (e.g. interpretation and analysis of the data, forwarding to the global data sink). The terms *sink* and *base station* are equivalents and together with a server connection a *gateway* is formed. Various functions are performed on the server such as data storage, data pre-processing, visualization or node configuration. In general, the gateway has a connection to other components of the cyber-physical system and provides the collected data to the components for different application purposes (e.g. online analysis, entity management). [24, 10]

[1] 'IPBaseStation is a modification of the generic BaseStation which ships with tinyOS-2.x. It alters the serial protocol to pass 802.15.4 frames instead of Serial.h packets. It also adds an out-of-band configuration protocol which allows a driver running over the serial port to reboot the mote, and to set the device address, channel, and retransmission parameters. These changes are useful when one wishes to use a mote attached to a computer as an 802.15.4 interface rather then an actual mote. The actual queuing logic for copying packets is mostly unchanged, and it continues to make use of serial ACKs.' [25]



(a) General illustration of a wireless sensor network



(b) Wireless sensor network special case

Figure 2.1: Components of a wireless sensor network including communication links

Figure 2.1b illustrates a special case of a wireless sensor network, which is mainly used in todays' application fields (cf. Section 2.5). In comparison to Figure 2.1a the wireless sensor network has one sink. The performed routing algorithm addresses all packets in the wireless sensor network to one sink. The server in the gateway complex is equivalent with the global data sink.

One of the first sensor nodes for wireless sensor networks in research - called smart dust - was developed by Jason Hill and David Culler at the University of California, Berkeley, USA [26, 27]. The task was simple: build small devices that are able to collect and transmit different data. Data was collected by different sensors (e.g. light sensor, microphone, motion sensor) and was transmitted over a number of hops to a sink via a wireless connection. The sink was part of the server's wired network, on which various analysis applications ran. [26, 27]
Based on the ideas and developments of the group around Jason Hill and David Culler the first wireless sensor network was deployed as part of the Golden Gate Bridge Project, a structural health monitoring (SHM) approach [28]. For this application the sensor nodes of type MICA were developed. Other application areas can

be found in health, military, building scenarios, and early-warning systems, e.g. for tsunamis and earthquakes. Those varied application areas are characterized briefly in Section 2.5.
During the last ten years, the developments in the field of sensor networks have been primarily in data transmission, cost-benefit ratio, and energy consumption. The miniaturization of devices allowed enhanced requirements such as more sensors per unit, lighter weight units, and the ability to recharge. Due to limited space on used hardware, resources must be placed and used meaningfully. It is, therefore, important to be aware of design principles for sensor networks (cf. Section 2.1) and to choose the appropriate operating system for the hardware used accordingly (cf. Section 2.4).

2.1 Design Principles

The operators of wireless sensor network set a high standard to good support of quality of service, energy efficiency, and scalability of the performed functionalities. In order to accomplish the operators' requests, design principles are important as described by Holger Karl et al. in reference [10]. The design principles introduced below, are differently weighted according to the chosen application scenario, the performed functionalities, the network size, and the resources of the hardware used.

The *network organization* is a very important issue, because it influences the performance of the sensor network. In today's applications sensor devices are organized in a distributed way combined with self-organization protocols. Different sensor nodes in the network perform different functionalities, such as data collecting, preprocessing of data within the network (e.g. aggregation, compression) or forwarding it towards the next hop. Depending on the deployment of the sensor nodes a self-organization mechanism must be performed in order to support a quick data transmission towards the sink. Representatives of self-organization protocols are neighbor-discovery algorithms and routing protocols that both support the establishment of a stable communication network. [10]
If the wireless sensor network is organized in a distributed way, the design principle of *in-network processing* becomes interesting, because resources of sensor nodes can be saved. In this case, selected sensor nodes in the network perform special algorithms, such as aggregation or compression, in order to pre-process data within the network. In general, the selected sensor nodes have more resources for calculation and buffer purposes. The advantage of in-network processing is the reduction of the amount of data in the wireless sensor network and the reduction of energy consumption due to the reduction of transmissions. [10]
In order to transmit recorded data in a wireless sensor network in an efficient and resource saving way towards the sink the design principle of *accuracy* becomes a research issue. If a wireless sensor network is working accurately, it takes the current network status into account in order to optimize routes. For example, if a sensor node has less energy resources, routes of the sensor network are updated in a way, which avoid the exhausted sensor. The goal is to ensure a long lifetime for this overused node. [10]
In wireless sensor networks it is popular to address data instead of the source (e.g. address of the node that collected the data). This strategy is based on the idea of *data centricity* which allows reconstruction of the data's source out of communication relationships (e.g. neighbors and link quality). Data centricity can be combined with database approaches in order to obtain interactions between sensor

nodes collecting data (called publishers) and devices requesting data (called subscribers; e.g. sensors with aggregation functionality, users with mobile devices) in a wireless sensor network. [10]
Today, different application scenarios exist, which include sensor devices with different capabilities for different functionalities within a wireless sensor network (e.g. aggregation, compression, encryption). In order to support the *heterogeneity* of system components (e.g. not every sensor node has the same sensors included), TinyOS has a *component-based architecture* [29]. This component-based structure allows dynamic adaptation of the protocols (e.g. activation or deactivation) for each component depending on whether it is needed or not (cf. Figure C.1). For example, one sensor node collects light and temperature data, whereas another node collects humidity and acoustic data. For the transmission of both data sets protocols are required, which adapt automatically to the required settings. In general, the exploitation of heterogeneity is combined with cross-layer optimization strategies, which allow the integration of new features with little modification to the already existing protocol stack. A drawback of cross-layer optimization can be a performance reduction of the network due to feedback loops. [10]

2.2 Hardware Specification of Sensor Devices

The standard equipment of a sensor node consists of a microcontroller, a memory unit, a communication device (radio), a power supply, and one or more sensors or actuators [11, 10]. Depending on the hardware design other components might be included such as LEDs, power switches, external power connectors, external RF connectors or expansion connectors. The main task of sensor nodes is to collect data and forward it. Depending on the application some sensor nodes can perform additional tasks, such as packet aggregation or data pre-processing, within the wireless sensor network. [11, 10]
The most relevant component of a sensor node is the microcontroller. It has several responsibilities such as data collection from sensors or actuators, data processing, decision management, and flow control. The memory is subdivided into a random access memory (RAM) and the read-only memory (ROM), which is sometimes an electrically erasable programmable ROM (EEPROM) or a flash memory. In contrast to ROM, RAM looses its content by power loss. Thus, it is used for buffering purposes, especially for data that can be changed while a program runs. The executing code is stored in the ROM in order to avoid reprogramming after power loss. The node, therefore, must only be programmed once and can be reused several times until a code update is required. [11, 10]

The communication device is responsible for the communication ability of the sensor node as described in detail in reference [10]. Depending on the application scenario the number of sensors and actuators differ. The power supply must be dimensioned according to the hardware, the application scenario, and the targeted lifetime of the system.

The hardware itself is limited regarding power resources, memory, and computational capacities [30]. Figure 2.2 shows platforms highlighting typical approaches, such as Berkeley Motes (e.g. MICA2dot, MICA2, IRIS, TelosB), and new approaches, such as OPAL. Another challenge is the size of hardware which ranges from coin to matchbox sized. This makes the placement of sensors next to other equipment such as microcontroller, conductor, and all other technical equipment especially challenging. [31, 32, 33]

	Smart Dust	MICA2dot	MICA2	IRIS	TelosB	OPAL
Chip	AT90LS2343	ATMega128L		ATMega1281	TPR2400CA	Atmel Cortex SAM3U4E
Program Flash Memory	8 kB	128 kB		128 kB	48 kB	256 kB
Measurement (Serial) Flash	128 B	512 kB		512 kB	1024 kB	n.n.
RAM	512 B	n.n.		8 kB	10 kB	52 kB
Configuration EEPROM	128 B	4 kB		4 kB	16 kB	n.n.
Power Source	Coin (CR2354)	Coin (CR2354)	2 AA		USB 2 AA	microUSB B 3 AA
Processor Current Draw	Active: 2.4 mA Sleep: <0.001 mA	Active: 8 mA Sleep: <0.015 mA	Active: 8 mA Sleep: <0.015 mA	Active: 8 mA Sleep: 0.008 mA	Active: 1.8 mA Sleep: 0.051 mA	Active: 30 mA Sleep: 0.0025 mA
RF Transceiver Current Draw	n.n.	Receive: 8-10 mA Sleep: < 0.001 mA	Receive: 10 mA Sleep: < 0.001 mA	Receive: 16 mA	Receive: 23 mA Idle: 0.021 mA Sleep: 0.001 mA	Receive: 16 mA
Size [mm]	25 x 25	25 x 6	58 x 32 x 7		65 x 31 x 5	60 x 50 x 10
Weight [g]	n.n	3	18		23	40
Sensors & Features	Temperature, Light, Acoustic	Light, Acoustic, 2-Axis Accel.	Light, Temperature, GPS, Humidity, Acoustic actuator, Acoustic, Barometric pressure, Seismic, Magnetometer		Light, Humidity, Temperature	Trusted Platform Module (TPM)
Manufacturer	Crossbow Inc.	Crossbow Inc.	Crossbow Inc.	Crossbow Inc.	Advantic Sistemas Y Servicios S.L., Crossbow Inc.	CSIRO

Figure 2.2: Hardware specification of common sensor node platforms

For the MICA2, MICA2dot and IRIS platform developers decided to work with an add-on sensor board, which is connected via an UART-connection to the main board. A big advantage of this technique is the flexibility of the sensor boards. Different boards can be equipped with different combinations of sensor types such as brightness, temperature, microphone, pressure, humidity or GPS. Depending on the application the sensor board must be exchanged, but the basic hardware stays the same. [31]
The OPAL platform also offers an UART-connection opportunity to add a sensor board. Additionally, this platform has an integrated Trusted Platform Module (TPM) chip on the main board which offers security functionality. The TPM technology is known from notebooks, smart cards, and other security sensitive devices in order to authenticate the user by his hardware. The TPM securely stores information about the host system (e.g. notebook, sensor node), such as cryptographically keys, certificates or passwords. The task of this security principal is to prevent fraudulent use of hardware and stored data. [33, 34]
With the TelosB nodes the strategy was different. Here developers decided to develop a sensor node platform with on-board sensors, so that the sensors are located on the same board as the processing and communication components. As a consequence, a new sensor node must be developed for applications that do not fit into the standardized sensor equipment. [32, 31]

The energy resources of sensor platforms vary as shown in Figure 2.2. In general, two AA batteries supply Berkeley Motes. For the coin sized platforms - MICA2dot and Smart Dust - the energy is limited by a coined cell. The TelosB

node can also be charged via a USB connection. The OPAL node can either be supplied by a three AA battery pack or by a non-exhaustible USB connection to a power source (e.g. PC, electric socket). In all cases, with exception to the USB support, the energy resources for the introduced sensor platforms are very limited. Thus, research focuses on integration of energy saving methods into the established wireless sensor networks as briefly introduced in Section 2.3. [31, 32, 33]

2.3 Energy Saving Methods

As illustrated in Figure 2.2 power resources of sensor platforms are very limited. Depending on the application battery power might be exhausted very soon (e.g. calculation, transmission). Usually, the most energy consuming procedure is the transmission of data to the next hop towards a sink followed by decoding and processing operations directly on the node. In order to ensure a longer lifetime of the wireless sensor network energy saving techniques should be integrated into the system.
One idea is to reduce energy consumption by implementing different modes of activity for the nodes as described in reference [35]: full active, idle, and sleep. Most power is consumed when the node is fully active, which means everything requires full power for listening, sending, and data collecting. Sensor nodes in the sleep mode consume very little energy - microjoule instead of millijoule [35]. Normally, nodes are programmed with internal clocks that wake them up in predefined intervals in order to perform, e.g. data collection followed by sending data directly to the next hop afterwards, before they fall into sleep mode again. The idle mode is a mode in between those two modes. Here the node actively listens to the surrounding traffic for beacons that let it know when to wake up and when to perform operations. [35]

Another idea to reduce energy consumption is software based and focuses on message size and network traffic. It has been proven that messages with smaller size consume less transmission energy than bigger messages [16, 17]. Therefore, one strategy is the reduction of overhead. In the context of sensor networks overhead is the meta information connected to each measurement, which is anytime the same for the same sensor node. Thus, splitting of sensor data packages into a message that includes meta information, and a message with the measured values is the strategy of choice. The message including meta information is sent out to all network components when the sensor node boots. After this announcement the sensor node only transmits messages, including the measured data, and refers to the before announced meta information. Also, each packet should produce low overhead in order to offer more space for individual payload, including relevant data. This approach can be realized by performing compression techniques on the messages components (e.g. headers). Another strategy is the reduction of traffic within the network itself in order to save energy. In this case either message aggregation or pre-processing of data can be performed on nodes within the network. The simple idea behind this approach is that sometimes not all data is required in order to perform a specific action. For example, only the average room temperature is required to manage the cooling system in a room. [16, 17]

A complete different method to raise the system's life time is to charge the power resources by using solar panels or environmental inputs such as vibration or temperature [36].

2.4 Operating System for Sensor Devices

Today the operating systems TinyOS and Contiki are very popular for wireless sensor networks which are briefly characterized in Sections 2.4.1 and 2.4.2 [37]. A few years ago researchers started to develop new operating systems based on Unix (e.g. MantisOS, LiteOS), but those approaches are not as popular as TinyOS and Contiki and do not support the hardware mentioned in Figure 2.2, yet [37, 38]. Thus, they are not addressed in this section. Finally, a brief look on code porting possibility between TinyOS and Contiki is presented, which is driven by industry partners and the ongoing development in the *Internet of Things* [39].

2.4.1 Characteristics of TinyOS

TinyOS is a research driven operating system, which is used for hardware with limited resources such as Berkeley Motes (e.g. MICA2, MICA2dot, IRIS, TelosB) that are used in the deployed wireless sensor network in this dissertation and were introduced in Section 2.2. TinyOS is an open source project [25]. It was developed by David Culler and Jason Hill at the University of California, Berkeley, USA in 2000 especially for wireless sensor networks based on the requirements of Berkeley Motes [26].
TinyOS is a component-based operating system and has an efficient multi-threading engine, which is composed of a two-level-scheduler and realizes the computer-time-spreading for threads. The application code consists of a `Makefile` including compiling commands, module files including configuration information, and one or more configuration files including the required information of the interfaces used and the component wiring [25].

The following two sources of concurrency are necessary prerequisites in order to understand the execution order of the two-level-scheduler in TinyOS [29]:

- *Tasks*, which contain current network routing and data preparation, typically take longer to finish, because hardware events have higher priority.
- *Events* on the other hand must be handled immediately, so that long duration blockades caused by current applications and data loss are prevented.

It is important to take the consumption of power into account to guarantee an efficient way of processing parallel data streams. The components link the sensors, the processing-, and communication-units that result in a wiring graph of components (cf. Figures C.1 and C.2). They contain a *Command Handler* and an *Event Handler*, so that an immediate reaction to a change in state is possible.
Components of a higher level communicate with the ones on lower levels via the *Command Handler*. After completion of a command on lower levels, independently of success or failure, the *Command Handler* answers with a signal (e.g. return value or flag) to the higher level in order to indicate the run to completion. A *Command* cannot cause *Events*. For example, the command `init()` is called in order to initialize a component for the first time, the command `start()` activates the start of the component, and finally the component is stopped with the command `stop()`. The *Event Handler* deals with *Events*, which can be caused directly or indirectly by hardware-interrupts (e.g. `fired()` in order to signal that an interval has passed). *Events* can even cause *Events* of higher levels. Frames contain the data memory of a component. Within a component *Tasks* are running, which have been started

by commands, *Events*, or other *Tasks* from the same component. Typical for *Tasks* are the atomic structure and the so-called *run-completion semantic*, i.e. the *Task* cannot be stopped and stops automatically when finished. [29, 25]

The advantage of the two-level-scheduler is that TinyOS only requires one memory stack, because memory allocation happens during program compilation process. This knowledge allows a classification of the components into three types [25]:

- Hardware abstraction components: Mapping of physical hardware onto a components model. One of the components is the Radio-Frequency-Module (RFM), which operates the pins of the RFM sender/receiver, and informs other components of successful transmission via *Events*.
- Synthetic hardware components: Mapping of ongoing hardware, e.g. the radio-byte-hardware, which sends data to the RFM, and gives a signal when a whole byte has been sent.
- High level software components: They contain the application-logic. They control components, and calculate the routing, and other data information.

A drawback of TinyOS is the missing support of simultaneous execution of several applications. Due to the Event-based programming, an operational capacity during applications with a high degree of parallelism can be reached. The modular architecture allows the user to add or remove different components without much overhead.

The configuration of the components is realized by a special programming language called nesC which is a derivate from the programming language C [40, 29]. It was specially designed for the requirements of TinyOS. The basic concepts are as follows [40, 29]:

- Construction and composition are separated.
- Component behavior is specified in terms of set of interfaces; interfaces are bidirectional.
- Components are statically linked to each other via their interfaces.
- NesC is designed under the expectation that code will be generates by whole-program compilers.
- The concurrency model of nesC is based on run-to-completion *Tasks* and interrupt handlers which may interrupt *Tasks* and each other.

2.4.2 Characteristics of Contiki

In contrast to TinyOS the operating system Contiki is an industry driven operating system. In 2004 the development of Contiki by Adam Dunkels at the Swedish Institute of Computer Science was driven by the request of a light weight operating system (2 kB RAM and 40 kB ROM) for embedded systems that later on was adjusted to the requirements of wireless sensor networks [41]. The technical term 'embedded system' summarizes small and limited hardware that is included in big systems (e.g. control system in a washing machine). Those embedded systems must function despite limited storage and computational capacities in the same way as

components of a wireless sensor network. A representative for Contiki usage is the ScatterWeb project [42].
The most important advantage of Contiki in comparison to TinyOS is the IP-communication (IPv6, IPv4), which is supported from the beginning. In Contiki the storage allocation happens during compiling. In contrast, the storage allocation for TinyOS is specified by the programmer, which means it is hard coded. A Contiki system consists of the kernel, the libraries, the program loader, and a set of processes, which can be an application or a service. The difference between an application and a service lies in the flexibility of the adaptations. In general, a service is programmed in a highly flexible manner so that different programs or applications can use it. In contrast the application is only developed for one special goal, e.g. health or building monitoring (cf. Section 2.5). In order to be flexible, Contiki supports different hardware such as Tmote Sky, JCreate, TelosB,MicaZ, Scatterweb plattforms MSB and ESB [41, 39, 30]. The programming language is Java. The process and the drivers can be replaced without interrupting a running system. If the processes need to communicate with each other they go through the kernel and communicate directly with the required hardware. The kernel itself is only responsible for event handling and outsources the remaining tasks to libraries that are linked if needed. In comparison to TinyOS the program code of Contiki consolidates all relevant packet imports and the application code itself in one file. [41, 39]

2.4.3 TinyOS code porting Contiki

In research TinyOS is the operating system of choice. The big advantages are the modular structure, the ongoing development, and the established community. A disadvantage is the fact that TinyOS requires special hardware such as Berkeley Motes that narrows the application field. But if an implementation runs under TinyOS with little complexity, the code and hardware can be transferred to Contiki. In order to realize the code porting, developed modules under TinyOS need to be subdivided into application and service processes in Contiki. The underlying wiring of components in TinyOS needs to be replaced by services interfaces in Contiki. If the code of TinyOS is adapted to the code requirements of Contiki, everything runs under the other operating system. For example, the TelosB nodes represent a platform, which can be programmed with TinyOS and Contiki, if the required drivers are supported. [39, 25]

2.5 Characteristics of Application Scenarios

The scaling of a wireless sensor network and its supported features depend on the application scenario, which ranges from environmental monitoring tasks (e.g. influences to building structures or user comfort), logistic, and localization tasks to health care areas. Those scenarios can be divided into indoor, outdoor, or mixed scenarios [43]. Alternatively, the applications can be divided depending on the mobility of components [11]. It is a more specific grouping as performed in this section. In every scenario different challenges must be faced such as weather conditions, architecture structures, as well as environmental factors. Depending on those challenges the chosen hardware must be dimensioned and designed. The interactions between network components are periodical measurements, event detection, and tracking. Before establishing a wireless sensor network developers must be aware of design principles, hardware constraints, and operating system possibilities as discussed in the previous Sections 2.1 to 2.4.

The currently established wireless sensor networks can be classified into the following three groups, where grouping depends on the node and sink location or status [11]:

1. Group: Static sink and static nodes (cf. Section 2.5.1)
2. Group: Static sink and mobile nodes (cf. Section 2.5.2)
3. Group: Mobile sink and mobile nodes (cf. Section 2.5.3)

Finally all the application groups have in common that networks consist of one sink and one or more sensor nodes (cf. Figure 2.1b). Those sensor nodes are equipped with application-dependent hardware and periodically collect values. Those values are transmitted to the sink automatically, if it is in communication range. Depending on the network structure this data transmission happens over one or more hops where sensor nodes can work as intermediates. [11]

2.5.1 Group 1 - Static Sink and Static Nodes

If the sink and the sensor nodes are static and the location is known, the established wireless sensor network is included in the first group. Those networks are usually used if everything is known and the monitored area or situation is fixed. In general, those deployed networks can be found in the area of structural health monitoring (SHM) setups observing building structures. It is used to analyse the changes in the structure of the building, which can affect the performance and the architectural stability of the construction.

The first recorded project is the Golden Gate Project in San Francisco, USA [44]. The aim of this project is to observe the ambient vibration of the same named bridge, because the bridge is located in an area where earthquakes and sharp sea winds often occur. The bridge is a frequently used traffic connection. The construction, therefore, needs to be stable and resist strong vibrations. Due to the construction of the bridge, it is essential to measure two directions perpendicular to the bridge's span - one up-down and one across the span. Two types of accelerometers are used in the project which have various measurement ranges to account for all possible vibration modes. The first type has a range from -2G to 2G for big movements (e.g. earthquakes) and the other has a narrow range from -0.1G to 0.1G to sample ambient vibrations (e.g. car traffic, wind). The bridge is 6450 ft long in total, the towers are 500 ft high and the bridge piers are 246 ft in parallel away from each other. The distance between the two bridge towers is 4200 ft. [44]
Project leaders decided to work with over 50 nodes and used Berkeley-Motes, which were the technology of choice at that time. The location of the sensor nodes is fixed as well as the sink. Collected measurements are transmitted via hops towards the sink where the longest hop distance is 280 ft long. The established network works with IEEE 802.15.4 at 2.4 GHz and supports bi-directional communication. The bi-directional communication was established in order to update the measurement intervals of the nodes depending on the analysis results of the data. For example, a higher sample rate is valuable during emergency situations. Long lifetime of the system is ensured, because sensor nodes transmit data only a third of the time and otherwise are in idle state combined with solar panels to recharge the batteries. [44]

Such an established wireless sensor network with fixed locations of sensor nodes and sinks (perhaps with modified hardware) can be found in different constructions,

which are located at critical environments such as earthquake regions. Another approach with the same setup situation can be found in emergency alert systems with fixed locations [45]. The most popular examples are the tsunami warning systems established in the pacific sea area [46, 47] and the earthquake warning system in Europe [48]. Those systems are combined with an alarm system via mobile devices. The sensor nodes are located in a fixed position in the sea in measurement buoys or next to known regions for earthquakes, such as volcanoes or seismotectonic active regions. They measure the movement of water, sea ground or earth. The data is transmitted to a global data sink located in a research center at the land surface where data is analysed. Depending on data changes researchers can predict a tsunami or an earthquake. In this case they start the early warning system, which is integrated in online applications (apps) available for mobile devices, and allow the people to evacuate as early as possible. [45, 46, 47, 48]

Other examples for group 1 networks can be found in medical and building (home or office) applications. In the medical field the technical term is wireless body area networks (WBAN) [49]. In this case, different sensors are located on the human body, such as movement sensors or cardioplegia sensors, to observe the movement pattern or heart beat of a person. The person either wears the sink, for example on the belt, or it is located next to the person such as in the neonatal monitoring case. The body sensors must bridge a short distance to the data sink, so that it can be done in one hop compared to the other approaches included in this application group. The sink either stores the data and needs to be connected to the internet manually to transmit the data to a medical analysis center (= global data sink) or is equipped with a permanently active internet connection and transmits data directly e.g. via a mobile phone. Such a medical emergency application was implemented by the CodeBlue project published in reference [50].

2.5.2 Group 2 - Static Sink and Mobile Nodes

The second group of wireless sensor networks has a slightly modified setup compared to the previously described grouping in Section 2.5.1. The sink is still static, but the sensor nodes are mobile. Those networks can be found in military, logistics, and building applications.

Networks in military scenarios deal with monitoring militants activities on roads or in villages, intrusion detection, and, in general, are used for protection purposes [51, 52]. Usually a headquarter has a fixed location and should be protected. In order to ensure protection, the surroundings must be observed. The sink is located in the headquarters. Flying objects can be equipped with different sensors such as GPS and infrared sensors in order to observe the region. Those aerial drones are unpiloted and fly over the region of interest in order to collect data. This data is usually transmitted via a satellite connection to the ground-based global data sink where the tactical companies analyse collected data and plan the attack strategies for military groups. [51, 52]

In logistic scenarios the sink is generally fixed in the transport vehicle and the sensor nodes are on the transport material such as packets or boxes. Here the sensors are equipped with a GPS unit, with a radio-frequency identification (RFID) unit or vibration sensors [53]. One of the challenges of this setup is the question of self-organization. Boxes might be loaded randomly and must organize a working infrastructure by themselves in order to ensure a working network to transmit data

to a sink. If a box is moved, its location and communication links need to be updated, which is the main reason for putting the application in group 2. [53]

The home or office application scenario is a variable scenario. Depending on the possible scenario it can either fit in group 1 or group 2. In every scenario the data sink is fixed. If sensors like temperature, humidity, gas or brightness for monitoring tasks of the environment are located on known positions, the network can be classified in group 1. If the sensors, such as GPS, are used to locate persons in the building, the scenario belongs to group 2. The project AutHoNe deals with those scenarios and is used as the basic application for the presented work in this doctoral thesis [13]. One part of the AutHoNe project was to measure environmental data and transmit it to a gateway. On the gateway the sensor data is compared to predefined knowledge. Depending on the analysis' result an action is planned and advised to the corresponding actor (e.g. climate management unit). [13]

2.5.3 Group 3 - Mobile Sink and Mobile Nodes

Wireless sensor networks dealing with wildlife monitoring tasks are representatives of a network of the third group [54, 55]. The characteristics of this group are the mobility of sensor nodes and sink. Here the hardware needs to be equipped with a huge memory unit, because data needs to be stored until a sink is available.

The ZebraNet project developed by the University of Princeton (USA) is one example of such a network type [55]. In this project zebras wear a tracking collar that weighs 1090g. This collar consists of a GPS chip, CPU, 640 kB flash memory, short-range radio unit, long-range radio unit with packet modem, Lithium-Ion batteries, and solar cell array [56]. The main characteristic of this sensor system is the long-term animal tracking even over long distances (e.g. > 100 km). This arrangement allows researchers to follow and find those zebras throughout their habitat in order to collect information. The project is located at the Mpala Research Center in central Kenya. Biologists observe Plains Zebra (*Eciton burchelli*) which live in tight-knit uni-male, multi-female breeding groups. Only males need to wear collars to collect enough information about the whole group, because males cause the the direction of movement. Collected data is stored locally on the collars until a sink is available. A ranger with the necessary equipment serves as mobile sink. The ranger drives to well known places, such as water holes, and waits until zebras cross the range of the sink. The sink announces itself permanently to the environment. If a collar detects this signal, it directly transmits all its data automatically. In order to ensure that all collected data from all collars reach the sink even if they are not in range, collars transmit some data to other reachable nodes in their surroundings. It might happen that those nodes have earlier contact to the sink and then transmit also the out of range nodes' data. A similar approach was done with Iberian lynx as reported in reference [54]. [55, 56, 54]

In 2012 the 'UvA Bird Tracking System' started with tracking birds' movement [57]. The project was developed at the University of Amsterdam. In order to observe such small animals, lightweight hardware must be developed. As described on the project homepage the hardware utilized includes a tri-axial accelerometer and a GPS logger. In order to support a long lifetime, solar and rechargeable batteries power the hardware. A bi-directional communication between the base station and the mobile bird hardware was established in order to allow updates of the system. The sink is represented by mobile ground stations, which further transmit the data to the labs (= global data sink) using satellite connections. Received data is

automatically processed and visualized in the developed virtual lab. The global goal of this project is to gain a better understanding about migration and navigation of birds. In addition, the project develops software in order to gain access to recorded measurements outside the laboratory, to share data with other research groups, and to visualize the data mapped on global maps in order to follow the travel distances of the birds. [57]

2.6 Summary and Findings

This chapter gave a brief introduction to wireless sensor networks, which will be used as required background information for the upcoming chapters. First, this chapter introduced general design principles that must be kept in mind in order to establish a wireless sensor network. Followed by a characterization of commonly used hardware in todays' sensor networks. Pointing out the constrains of the hardware concerning memory, energy, and computational capacity. In order to save resources general used energy saving methods were introduced, such as compression and aggregation. Additionally, the two main used operating systems for wireless sensor networks - TinyOS and Contiki - were briefly characterized. TinyOS is mainly driven by academic/research decisions and Contiki by industry constraints. It was also shown what is required to port protocols between both operating systems in order to use different hardware, because not every hardware is supported by both operating systems. In order to conclude the background information about sensor networks different application scenarios were introduced and grouped by the mobility of the network components. Todays' deployed wireless sensor networks have in common that they transmit data to one global data sink, perhaps via a number of intermediate hops (e.g. other nodes, routers or local data sinks / gateways). In summary, this chapter demonstrated what constraints different sensor hardware have, where sensor networks are available today, and what features should be supported in order to optimize the behavior of wireless sensor networks.

3. Related Standards for Wireless Sensor Networks

Today, a number of implemented use cases for the *Internet of Things* (IoT) and wireless sensor networks exist, as described in the previous Section 2.5 and in reference [58]. In most of the scenarios (e.g. Project UvA Bird Tracking System, building, logistic or military applications) it is preferable to make collected data globally accessible to authorized users and to authorized data processing units (e.g. analysis server, visualization tool) through the Internet. In general, much of the collected data in these scenarios, such as location and personal IDs, is sensitive information. User's privacy can be violated if unimportant data (e.g. energy consumption, light, sound) is monitored somehow, so that it can be directly linked to the user. As pointed out by the following statement given by the market research firm Gartner Inc., it is essential to think about security solutions [59]:

> *'The Internet of Things concept will take more than 10 years to reach the Plateau of Productivity - mainly due to security challenges, privacy policies, data and wireless standards, and the realization that the Internet of Things requires the build-out of a topology of services, applications and a connecting infrastructure.'* [59]

The solution for security problems is to integrate end-to-end security into the *Internet of Things*. This means, that communication between client (e.g. sensor node) and server (e.g. gateway) is isolated from the outside and not readily ascertainable to anybody else, that data cannot be modified during transmission, as well as the communication channel. Protecting data once it leaves the scope of the local network is not enough. An end-to-end security solution provides security even if the underlying network infrastructure is not under control of the user. This case occurs if data is transported to a remote data center (gateway). If the gateway is a stationary installation, it can provide security functionality to the higher-level network, but the gateway is still a high-value target for attackers. If an attacker gains access to a gateway, access to all information is possible. Examples for scenarios where user has limited control can be found in the logistic area, where no gateway to the provider's network with user's control exists, or in building scenarios, where the users share the common infrastructure for metering and climate-control purposes,

but try to keep their device's data private from other network members. If a protocol such as DTLS is used, security responsibility is shifted from the gateway to the communication participants. The DTLS protocol is located between the transport and application layer, which excuses the infrastructure (e.g. gateway) from supporting security mechanisms. Thus, the gateway's attractiveness for the attacker is reduced. Another possibility is the usage of IPsec which is located on the network layer [60]. IPsec offers two packet formats for cryptographically protected data: (1) the IP authentication header provides integrity protection and authentication [61] and (2) the IP encapsulating security payload which also supports confidentiality protection through encryption [62]. [59, 60, 61, 62]

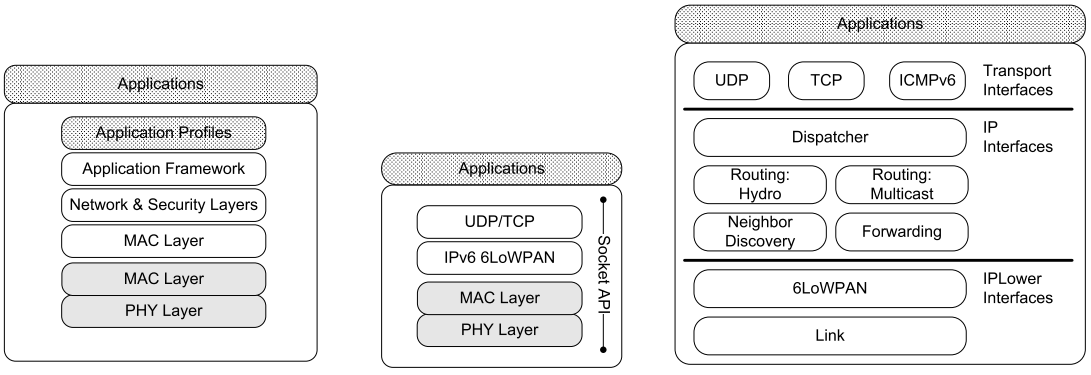
The request for integrating constrained systems (e.g. wireless sensor networks) into the *Internet of Things* was motivated by Jonathan Hui and David Culler in their article 'IP is Dead, Long lives IP for Wireless Sensor Networks' [63]. The authors pointed out that a connection to the *Internet of Things* can only be successful if the hardware of wireless sensor networks can co-exist with IP communication. But at the same time, the authors requested resource saving solutions in order to support the basic functionality of data collection, pre-processing and forwarding without restrictions. They developed a new stack for the constrained hardware which could be interoperable with existing standards from the IP networks (e.g. TCP/IP stack), and do not restrict the previously mentioned basic functionalities. The problem with standards supporting IP communication is their complexity, which makes it impossible to transfer them to constrained hardware with limited resources as described in Section 2.2. Researchers, therefore, decided to develop new resource-adapted stacks as described in Section 3.1. [63]

After bringing IP communication to such constrained hardware, researchers focused on optimizing data transmission within such networks. Usually, transmitted messages in wireless sensor networks include data itself and its meta information. If IP communication is supported additional information is placed into the individual payload, so that space in the maximum transmission unit is more limited (cf. Section 5.1.1). But the goal of transmission is still the same: Put as much data as possible in one message. With this aim in mind this dissertation accepts the challenge to optimize the data transport in wireless sensor networks by using protocol standards from IP networks. Additionally, the developed solution should be integrated into the application layer in order to remain compatible with possible stack exchanges below (e.g. use BLIP instead of 6LoWPAN as described in Section 3.1).
In IP networks the 'Internet Protocol Flow Information Export' protocol is used to transmit data. The protocol itself offers a packet design for efficient usage of the maximum transmission unit by reducing redundant information, such as meta information that is the same in each data packet from one sensor node. Section 3.2 will introduce the IPFIX protocol and point out in detail what functionality or characteristic is interesting for wireless sensor networks. The required modifications (e.g. IPFIX message header compression) for the integration into wireless sensor networks are described in Section 4.1. Next, the aggregation support within the network using TinyIPFIX as message structure is described in Section 4.2. Information about the implementations are given in Sections 5.1 to 5.2, and a detailed evaluation in Chapter 6, assuming a building scenario as application example.

When the wireless sensor network is integrated into the *Internet of Things* and data transport is optimized, the last challenging task is security support. In this dissertation the idea for solving this task is the usage of security standards known from applied cryptography. Therefore, Section 3.3 gives an overview of security tasks and techniques. The section starts with the definition of security attacks, security services, and security mechanisms. Followed by a brief description of standardized security solutions such as cryptographic functions, public key infrastructure, (D)TLS protocol, and trusted hardware component by using a trusted platform module. In Section 4.3 existing security solutions for wireless sensor networks are described and it is motivated why this dissertation prefers to support the DTLS protocol. Information about the implementation is given in Section 5.3, followed by an evaluation in Section 6.3.

3.1 IP Communication Support

Today, Internet access is possible nearly everywhere, which calls for global communication solutions between participating entities. Depending on the applications access to an established network, such as a wireless sensor network, should also be possible via Internet access. This approach can be realized by bringing IP communication to those devices. The TCP/IP stack offers this opportunity, but it is too complex for the limited hardware resources of sensor devices. In 2002 the **ZigBee stack** was developed by the ZigBee-Alliance based on IEEE 802.15.4 and supports IP communication for resource limited hardware as described in reference [64]. Figure 3.1a shows the supported ZigBee stack developed by the ZigBee-Alliance. The stack includes the physical and MAC layer from IEEE 802.15.4. The physical layer supports three frequency ranges: 868 MHz, 933 MHz, and 2,4 GHz. Energy efficiency is gained by functionalities on the MAC layer. On top of those two layers the ZigBee part was added, including an extra MAC layer, a network and security layer, followed by an application framework and application profiles. The extra MAC layer includes updates for the underlying IEEE 802.15.4 MAC layer in order to extend basic functionalities. Applications are located on top of the ZigBee stack. The ZigBee stack requires 8 kB RAM, produces 8-16 bytes of network overhead, and only mesh functionality in networks of a maximum size of 65,000 components is supported as Mulligan analysed in reference [65]. Those constraints together with the limited hardware and the request for direct IP addressing were reasons to implement new stacks for IP communication on limited hardware. [59, 64, 65]



(a) ZigBee stack (b) 6LoWPAN nano stack (c) BLIP stack

Figure 3.1: Stack comparison

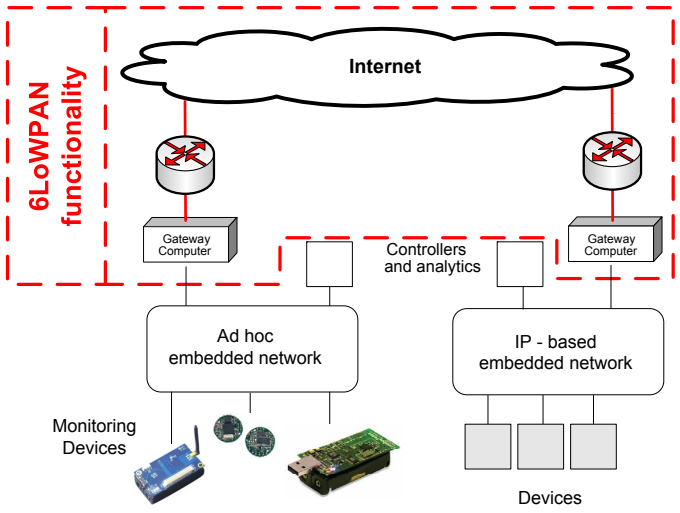


Figure 3.2: Communication interoperability via Internet using 6LoWPAN

In 2005 the Internet Engineering Task Force (IETF) working group 6LoWPAN was founded [66] dealing with research questions for supporting IPv6 in wireless sensor networks. Harvan et al. developed the **'IPv6 over Low power Wireless Personal Area Networks' (6LoWPAN)** implementation in 2007 [67]. Based on this implementation the IETF started the standardization process with the RFC 4919 specifying the problem statement, assumptions, and goals for IPv6 power Low-Power Wireless Personal Area Networks [68]. Detailed protocol specification were published in the RFC 4944 [69]. The idea of the implementation was to bring IPv6 features over UDP/TCP on sensor nodes regardless of the underlying physical layer. It allows communication via the Internet which results in interoperability as illustrated in Figure 3.2. The red dashed lines show the 6LoWPAN functionality, which covers the communication between two stand-alone networks via IPv6. [67, 68, 69]
The 6LoWPAN nano stack, as shown in Figure 3.1b, consists of a Socket API and applications, which are addressed via socket interfaces. The following components build the Socket API: The physical layer (PHY) of the 6LoWPAN stack is based on IEEE 802.15.4 and supports basic communication capabilities on radio. The next layer is compliant to the MAC layer of IEEE 802.15.4. It supports a contention-based channel access method of unslotted CSMA/CA for data transmissions. In comparison to the ISO/OSI layer model an adaptation layer in 6LoWPAN - called IPv6 6LoWPAN - replaces the network layer. This layer is the main component of 6LoWPAN, which supports compression of TCP/UDP and IP headers. The normal TCP/IP header dimensions are too large for transmission in IEEE 802.15.4 networks. A maximum transmission unit of 1280 bytes is required for IPv6, so that the adaptation layer supports packet fragmentation and reassembling techniques. Among other tasks the layer is responsible for routing, neighbor discovery, and multicast support. UDP and TCP are transport protocols, which are supported by the next layer representing the transport layer in the ISO/OSI layer model. [67, 68, 69]
As described in the RFC 4919, 6LoWPAN works with a small packet size which operates with a maximum transmission unit of 127 bytes on the physical layer 102 bytes individual payload on the media access control (MAC) layer [68]. If

6LoWPAN is used, only 2-11 bytes of overhead occur depending on used compression [65]. The protocol supports 16-bit and 64-bit address space, different bandwidth, and different network topology. Concerning security issues 6LoWPAN offers AES-128 encryption and authentication. Finally, the 6LoWPAN nano stack only requires 4 kB RAM compared to ZigBee and supports network sizes up to 2^{64} components [65]. On the lower layers the functionalities of IEEE 802.15.4 are included followed by a special developed IPv6 6LoWPAN and UDP/ICMP layer. Individual applications (e.g. security operations) are located on top of the nano stack. [65, 68, 69]
Concerning security issues it was pointed out in RFC 4919 that applications using 6LoWPAN require confidentiality and integrity protection, which can both be provided at different layers (e.g. application, transport, network, link) [68]. It was recommended to use link layer security, because most IEEE 802.15.4 devices support the *Advanced Encryption Standard* (AES) on the link-layer. AES is a block cipher operating on blocks of fixed length and is adaptable to different modes of operation, as described in detail in Section 3.3.1.1. For network layer security the RFC 4919 distinguishes between end-to-end security (e.g. IPsec) and security limited to the wireless portion of the network (e.g. secure gateway + IPsec tunnel) (cf. Section 4.3.1.3). [68]

Due to the advantages of 6LoWPAN compared to the ZigBee approach the development continued. Researchers at the University of Berkeley are responsible for TinyOS, which became more and more popular as the operating system of choice in academic research. In 2010 researchers decided to implement a 6LoWPAN version compatible with TinyOS requirements - called **Berkeley Low-power IP stack (BLIP)** [70, 71]. This implementation supports different node platforms (e.g. IRIS and TelosB) with the RF transceiver CC2420 from Texas Instrument. The RF transceiver CC2420 is an IEEE 802.15.4 compliant radio transceiver, which supports a maximum packet length of 127 bytes including all headers [18]. The BLIP stack consists of the following parts (bottom to top) as illustrated in Figure 3.1c [72]:

- IPLower Interfaces: In this part the link layer support is included with a 6LoWPAN layer on top. The 6LoWPAN component compresses headers and breaks large packets into multiple link-layer fragments.
- IP Interfaces: Those interface represent network functionalities including support for IPv6neighbor discovery, forwarding, routing (default selection, point-to-point) and dispatcher functionalities.
- Transport Interfaces support standard Internet transport protocols (e.g. UDP, TCP).
- Applications include all user protocols and algorithms (e.g. the developed TinyIPFIX in this dissertation).

IPv6 itself requests a maximum transmission unit of 1280 bytes as specified in RFC 2460 [73]. But the today used IEEE 802.15.4 compliant RF transceiver CC2420 only supports a packet size of 127 bytes [18]. In order to support IPv6 communication, fragmentation must be supported. This fragmentation is realized by the included 6LoWPAN layer in the IPLower interfaces of the BLIP stack [72]. In the

installation packets of BLIP a tunnel driver is included to realize the node connection to the gateway/computer, which performs the role of the router as shown in Figure 3.2). As specified in RFC 4944 addressing, stateless auto configuration and header compression features are implemented [69]. Concerning security issues BLIP supports the same security as 6LoWPAN, because it is an implementation of 6LoWPAN for the operating system TinyOS [72].

From this point on it is assumed that a wireless sensor network supports IP communication and might be connected to the Internet. This setup makes it possible to think about adaptation of IP network standards to wireless sensor networks (cf. Section 3.2), but it also brings up the question of security, because the collected data might be sensitive and restricted in access (cf. Section 3.3).

3.2 Internet Protocol Flow Information Export Protocol

If a network has a connection to the Internet or consists of more than two parties, it is helpful to monitor the occurring traffic within. This monitoring is essential if the system contains sensitive data in order to support a secure system and to recognize abnormal behavior as soon as possible. For monitoring purposes in IP networks the IETF further developed the Netflow protocol from Cisco System, which resulted in the IPFIX protocol standardized in RFC 5101 [19]. The IPFIX protocol was developed in order to standardize the exchange of network monitoring information. This means, the protocol deals with the transmission of IP flow information between different instances in the network as described in the RFC 5101.
The protocol is located on the application layer and works among others over UDP, TCP, and the 'Stream Control Transmission Protocol' (SCTP). IPFIX itself is a *push-protocol* which periodically sends out collected data by itself. This behavior makes IPFIX an attractive choice for wireless sensor networks, because they often rely on collection traffic. The underlying traffic pattern means that information flows from many source nodes to only a few information sinks, such as the gateway nodes in wireless sensor networks. Another welcome characteristic of the protocol is the template-based design. An *Exporter* periodically transmits its data to one or more *Collectors* by using IPFIX messages. The first message an Exporter transmits to its neighbors when entering the network is a so called *Template Record* (cf. left part of Figure 3.3 [20]). The Template Record consists of meta information about the upcoming *Data Records* (cf. right part of Figure 3.3 [20]). The receiving neighbors for decoding purposes regarding received Data Records from this special Exporter store those Template Records. After the Template Record is announced the Exporter can start with transmission of the Data Records, which refer to the corresponding Template Record. Due to the highly dynamic nature of network structures, Template Records are repeated periodically to ensure a successful decoding. [20, 19]
As a result of the separation, packets are smaller, because the Data Records only transmit values and not the data together with its meta information. The coherence between Template Record and Data Record is illustrated in Figure 3.3 [20]. The different Records and the corresponding headers build a so called *Set*, which is transmitted in total. If it is requested by the monitoring task, the IPFIX protocol also offers the opportunity to combine different Data Records referring to the same Template Record in one Set. [20]

The IPFIX protocol itself is very flexible [20]. It allows data sending of every type if needed. The only requirement is to specify the data by an individual data triple

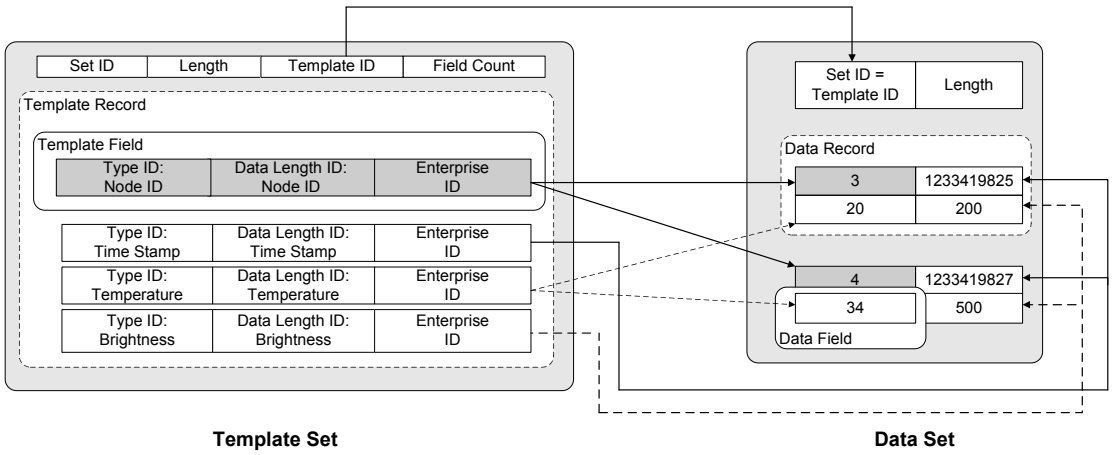


Figure 3.3: Components of the IPFIX protocol showing decoding of the data

in the *Template Field* which consists of the following three fields: `Type ID`, `Data Length ID`, and `Enterprise ID`. In the case of exchanging non-standardized data, the vendor must use Type IDs located above ID 32767. Type IDs below are reserved for traffic measurement data types. A Type ID specifies the type of data while the Enterprise ID denotes the organization which issued the Type ID. In case of new types the vendor needs to register the new chosen Enterprise ID with the Internet Assigned Numbers Authority (IANA) [74]. Via this serial number each vendor can be identified uniquely. A vendor can be a special hardware value such as a sensor. The Data Length ID includes the field size of the value, for example, two bytes. [20]

Due to the previously mentioned characteristics, the IPFIX protocol is of interest for constrained networks, especially the used data format. Thus, a modified version of the IPFIX protocol, only including the flexibility and the template-based data format, was implemented in this dissertation. The resulting protocol was called *TinyIPFIX* where sensor nodes act as Exporters and transmit their measurement data using IPFIX. Before implementing IPFIX on very constrained hardware developers must familiarize themselves with the standard implementation known from common IP networks (e.g. message structure), especially with the required IPFIX headers. An IPFIX message itself consists of an *IPFIX Message Header* and an *IPFIX Set Header* followed by one or more Templates or Data Records. [20]
Figure 3.4 shows the structure and sizes of the IPFIX headers, which cause an overhead of 20 bytes in total, whereas IPFIX Message Header spends 16 bytes and Set Header four bytes [20]. The first field `Version Number` with two bytes specifies the version of the utilized IPFIX protocol followed by a two bytes long `Length` field given the total length of the IPFIX message including all headers and payload. The four bytes long `Export Time` field is a time stamp to verify when the packet was constructed for transmission. The following `Sequence Number` and the `Observation Domain ID` (each four bytes long) are two more IDs in the IPFIX Message Header which are essential for putting the packets into the correct order at monitoring or analysis point. This information is needed if the packets are received in a mixed order caused by different chosen routes as the packets find their way to the sink. The Set Header consists of a `Template ID` and `Length` field each two bytes long. The first field specifies an ID, which is essential for decoding tasks between Template and Data Records. The second field specifies the total length of the following fields which build the Template or Data Record as shown in Figure 3.3. [20]

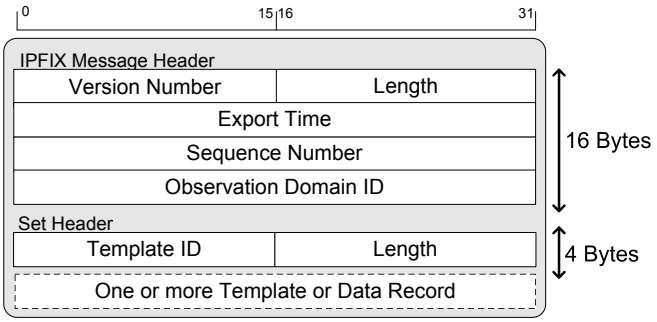


Figure 3.4: Structure of an IPFIX Message [bits]

The additional 20 bytes overhead caused by the IPFIX message header and the Set header reduces the available payload for sensor data in a message (cf. Section 5.1.1). In the experiments performed for this dissertation the RF transceiver CC2420 is used and the maximum transmission unit on the MAC layer, therefore, is limited to 102 bytes [18] where 12 bytes are required for the TinyOS packet structure and 43 bytes for IP support by 6LoWPAN in compressed format with long addresses, as specified in RFC 4944 [69] support. The required payload space for IP communication can be reduced to six bytes, when using BLIP implementation in compressed format with short address support [75]. If the IPFIX message header is used, the resulting payload size of 84 bytes is reduced to 64 bytes (cf. Figure 5.4). A detailed description of the final packet structure is provided in Section 5.1, where the implementation of the TinyIPFIX protocol and its extensions is described. The global goal in wireless sensor networks is it to transmit as much data as possible in one individual payload. Therefore, the overhead caused by new headers must be reduced. The design decisions for an implementation of IPFIX for wireless sensor networks are described in Section 4.1.2, followed by a detailed description of the used message format under the operating system TinyOS 2.x in Section 5.1.

3.3 Security Issues

Today the term *security* summarizes concepts that are used to secure a system or communication between devices. In order to achieve an equal understanding of this concept, efforts to organize this research field have been undertaken. This section gives an overview of concepts, which are required in order to secure a network. Used terms follow the security terminology published in the RFC 2828 [76]. In general, the two communication partners are called *Alice* and *Bob*, and the attacker is called *Eve*. Those terms will be used throughout this section in order to describe attacks and defense strategies.

Before a successful defense strategy can be developed for a system a user must think of the following three concepts [77, 78]:

1. Security attack,
2. Security service, and
3. Security mechanisms.

Under the term **security attack** all strategies are summarized where an attacker tries to compromise security of a system in order to gain access to stored information

or to processed functionalities by the system (e.g. transmission, processing) [77, 78]. In literature it is assumed that capabilities of an attacker are based on the Dolev-Yao model [79]. This model assumes that an attacker can receive messages from other members of a network. The attacker can overhear and intercept all messages sent in the network. The attacker is a valid member of the network and can send messages to all other members of the network. In addition, the attacker can send messages to other network members while using a false identity. In general, attacks can be distinguished between active and passive attacks, which are briefly characterized in the following. [77, 78, 79]
In active attacks the attacks performed by *Eve* aim on the system's performance in order to affect the operations of the system. Such active attacks are masquerade, replay and denial of service attacks, and message modifications. During a masquerade attack where *Eve* masquerades herself as a person *Alice* knows (e.g. *Bob*). *Eve* uses her fake identity to collect information about *Alice* and her used message encryption in order to gain knowledge about *Alice*'s secured communication. In comparison to replay attacks, *Eve* plays an active role in the communication to collect information. Before *Eve* can perform replay attacks she has a passive role where she passively captures data of the communication between *Alice* and *Bob*. Passively captured data is retransmitted by *Eve* in order to gain access to the system or files of one of the communication participants. Normally, replayed messages are not modified in comparison to attacks performing message modifications. In case of message modifications, *Eve* captures a message send from *Alice* to *Bob* (e.g. 'Meeting starts at 9 a.m. in the computer science lecture room.'), modifies the message content (e.g. 'Meeting starts at 9 a.m. in the seminar room in the physics department.'), and sends this modified message to *Bob*. As a consequence *Bob* will wait in the wrong room for *Alice*. The previously mentioned three active attack types focus on influencing message flow between *Alice* and *Bob* with more or less input. The forth active attack - the denial of service attack - has a deeper impact on the communication. This attack focuses on disturbing the service provided by the system or communication via the network. Different ways exist to perform this attack, but in general the system or network is overloaded with traffic/messages in order to occupy resources. This attack can cause a crash and finally a shut down of the whole communication system. [77, 78]
Passive attacks, in comparison to active attacks, are harder to detect, because the attacker does not modify any data. Representatives of passive attacks are the release of message content or traffic analysis. In the first case, *Eve* passively eavesdrops on the communication between *Alice* and *Bob*. The aim of *Eve* is to learn the message content passed between the two communication parties. Here *Eve* also reads the content of the messages. In comparison, if *Eve* performs a traffic analysis as a passive attack, she does not read the content but analyses the pattern of the message exchange between *Alice* and *Bob*. Hereby, *Eve* can collect information about the identity of the persons, message frequency, and message length, among other information. *Eve* uses the collected information to infer the communication nature of *Alice* and *Bob*, e.g. by modifying the location or identity of the communication hosts. This kind of attack can be performed by *Eve* with less overhead if the communication between *Alice* and *Bob* is not secured (e.g. message encryption). [77, 78]

Security services are integrated into a system in order to support security of data or information processing by performing one or more security mechanisms. Security services can be grouped into five categories that represent security goals

that should be addressed when designing a security solution for a system or network: (1) data confidentiality, (2) data integrity, (3) authentication, (4) access control, and (5) availability. [77, 78]
The first group deals with data confidentiality. The aim is to protect data from being read by an attacker. This group also includes protection mechanisms against traffic analysis with the goal to disturb the attacker's possibility to observe the communication's source or destination and the communication's characteristics. The second group deals with data integrity in order to ensure that the message send out is received by the destination unmodified. Hereby, research distinguishes between connection-oriented and connection-less services. In the connection-oriented service data integrity support includes the guarantee that message are not modified, not replayed, not duplicated, and not omitted. In contrast, the connection-less services only give a guarantee that the message is unmodified. Services supporting data integrity can also include recovery mechanisms that can detect a violation and automatically restore the prior state. The third group of services deals with authentication and guarantees that the communicating entity (e.g. *Alice*, *Bob*) is the one, who it claims to be. Authentication can be guaranteed either for each message or for the whole message exchange. In the latter, it is assumed that the participating entities authenticate each other before the message exchange starts and that it is impossible for an attacker (e.g. *Eve*) to successfully masquerade as one of the communication partners. The fourth group of services works in tight cooperation with authentication and focuses on access control. The services limit the access to host systems or applications. The entities can only get access if they are authenticated and have the required rights. [77, 78] The last group of security services deals with availability that offer defense support against denial of service attacks with disrupting the communication ability of a system or network. [77, 78]

Security mechanisms are processes that are designed to prevent attacks or to detect attacks if they happened and to recover the system from attacks [77, 78]. As specified by the ITU in the recommendation X.800[2] security mechanisms are incorporated into different protocol layers. The security mechanism *Access Control* supports mechanisms to enforce access rights to resources. Data integrity is provided by a variety of mechanisms that are applied either on a data unit or a message flow. In order to ensure the identity of an entity throughout information exchange, the authentication exchange is the mechanism of choice. The encryption mechanism uses an algorithm to transform plaintext into an unintelligible ciphertext and the other way round. In order to perform encryption, special algorithms and encryption keys are required. Digital signatures refer to data attached to a message. The digital signature can be used by the receiver to prove the source and integrity of the message. If attacks on routing occur (e.g. link congestion), the security mechanism *Routing Control* can be used for defense purposes. This mechanism selects other routes in order to avoid undesirable nodes. [77, 78]
Further security mechanisms and strategies that inspired the design decision for the implemented security solution in this dissertation are introduced in the following Sections 3.3.1 to 3.3.3. The upcoming security mechanisms and strategies are standardized solutions that are used for networks independent of the constraints of wireless sensor networks.

[2]Security Architecture for Open Systems Interconnection For CCIT Applications - Recommendation X.800: `http://www.itu.int/rec/T-REC-X.800/`

3.3.1 Security Protocols

As mentioned in the beginning of Section 3.3, different security mechanisms exist. This section focuses on security protocols, based on cryptographic functions and public key infrastructure. Presented background information refers to the book 'Network Security: Private Communication in a Public World' written by Charlie Kaufman et al. [78], as well as the book 'Protocols for Authentication and Key Establishment (Information Security and Cryptography)' written by Colin Boyd and Anish Mathuria [77]. A detailed security analysis of existing cryptographically algorithms and their transfer possibilities to constrained hardware platforms were done within the scope of the bachelor thesis by Christian Liedl [80] and the master thesis by Thomas Kothmayr [81].

3.3.1.1 Cryptographic Functions

In order to secure a system, cryptographic functions are the basic component. This section introduces the symmetric and public key cryptography, as well as message hash functions. Those security mechanisms only provide computational security, because the performance of these security mechanisms are expensive concerning time and computational capacities. In order to verify the strength of a cryptographic function, security is measured in bits. Assuming a 128-bit key length, it can be said that the message has 128 bits of security if no attack is known that is better than exhaustive search of the key space. In this case, an attacker has to perform 2^{127} decryption operations in order to find the required key. Due to today's high performing computers an attacker can solve the problem even faster. In 2007 the National Institute of Standards and Technology (NIST) published a security analysis report pointing out that 80 bits of security be phased out as of 2010 and that 112 bits of security will have a lifetime until 2030. If the previously mentioned example of 128 bits of security is performed, NIST recommended that it will be secure beyond the year 2030 [82]. [78, 77, 80, 81, 82]

The first security mechanism introduced in this section is the **symmetric cryptography**. In this mechanism the same key is used for decryption and encryption of a message. For example, *Alice* wants to send a message to *Bob* using symmetric cryptography. *Alice* performs an encryption algorithm where her plaintext is encrypted with the symmetric key. As a result of this operation she receives a cipher text, which is transmitted to *Bob*. *Bob* knows the symmetric key *Alice* used and performs a decryption algorithm in order to decrypt *Alice*'s message and receives the plaintext. [78, 80, 81]
When using symmetric cryptography it is required that the used algorithm is strong [78]. Which means an attacker should not be able to decrypt the cipher text or to recalculate the key if the attacker gains access to a number of plaintext and resulting cipher text combinations. It is also required that the key exchange between *Alice* and *Bob* was performed through a secure channel. If the attacker gains access to the key, the upcoming communication between *Alice* and *Bob* becomes unsecure. Thus, keeping the key secret is the most important requirement for a secure communication. The research community assumed that the key exchange is secure and therefore focused on the development of strong algorithms. The algorithms can be divided into two classes. The first class uses stream ciphers, whereas the second class uses block ciphers. [78, 80, 81]
As the name of the first class discloses, stream ciphers operate on data streams [78]. The result is that the cipher text is as long as the initial plaintext. The operation

consists of two parts: First, a long (pseudo-)random string is generated based on the used key. Second, the resulting string is combined with plaintext by a exclusive disjunction (XOR) operation. For an attacker it is impossible to determine the plaintext, because of using a one-time pattern for the (pseudo-)random string the resulting cipher text does not include enough information to recalculate the plaintext even if the attacker gains access to several cipher texts. A drawback of stream ciphers is the fact that if a block is lost, it is an unrecoverable error. If an unreliable transport protocol is used in the network, e.g. in wireless sensor networks, the usage of block cipher in the encryption algorithm is not applicable. [78, 80, 81]
In comparison to stream ciphers the block ciphers divide the plaintext into fixed sized blocks and apply permutations and substitutions on these blocks. As a result the cipher text size is always a multiple of the block size. The mentioned drawbacks of stream cipher concerning unrecoverable errors caused by block lost can also occur depending on the chosen mode of operations. [78, 80, 81]
Today, one of the most popular block ciphers is the 'Advances Encryption Standard' (AES) cipher [78, 77]. It replaced the 'Data Encryption Standard' (DES) cipher, because DES has a poor performance and offers only a 56 bit security, which is not secure anymore as pointed out by NIST [82]. AES offers three different key lengths: 128 bits, 192 bits, and 256 bits. The choice of the used key length depends on the available resources of the hardware and of the security request by the user. In the case of sensor networks limiting factors are the available resources, as briefly described in Section 2.2. In 2007 Großschädl et al. showed that AES performed well on sensor network hardware [83]. AES supports a block length of 16 bytes. In the case of bigger block length different handlings for encryption are possible. The 'Electronic Code Book' (ECB) block cipher mode uses the resulting block of cipher text produced by the encryption algorithm without any further modifications. When encrypting the same block of data independent of its location in the stream, the result will be the same cipher text. If this cipher mode is used, an attacker may be able to collect information and later on be successful. Instead, a solution is preferable where the resulting cipher text looks random. The 'Cipher-Block-Chaining' (CBC) mode makes this possible. The CBC mode uses the preceding cipher block in order to alter the result for the current cipher block. Connecting the plaintext block with the preceding cipher text block with the XOR function does this. Assuming that the output of each cipher text block is random, the performed CBC mode is comparable to the usage of a random number in order to alter each input block and prevent them from repeating themselves. When the process begins only one cipher text block is available, so that the XORing with the preceding block is impossible. In this case an initialization vector, a random number, is used instead. For a successful decryption the user needs to know this initialization vector and the key. When CBC is performed, an attacker can disturb the decryption if the cipher text blocks are changed or rearranged. A defense strategy to mitigate the influence of this attack is the usage of message integrity mechanisms. This means a second cryptographic round is processed over the source data as described in the book written by Kaufman et al. [78]. Protocols such as the 'Transport Layer Security' (TLS) protocol still use the CBC mode. For integrity checks TLS uses also 'Hashed Message Authenticity Codes' (HMACs). [83, 78, 82, 80, 81]

The second security mechanism deals with the problem, how to exchange a key over an insecure network. The solution for this task is the **public key cryptography**, which assumes a pair of related quantities (public and private) associated with each party [78]. In comparison to the public quantity the private one is never published

and stays secret. Different public key algorithms that vary in their supported functionalities and characterized in reference [77] exist.
The most noted algorithm is the RSA algorithm named after its inventors Rivest, Shamir and Adleman [77]. Before using RSA the spadework must be done by generating a public and private key. In order to generate the key pair the following steps must be performed whereas the steps (1)-(3) are used to calculate the public key $\langle e, n \rangle$ and the step (4) for the private key $\langle d, n \rangle$ [77, 81]:

1. Choose to large numbers p and q.
2. Calculate $n = p * q$.
3. Choose a number e that is relatively prime to the Euler function $\phi(n) = (p-1) * (q-1)$.
4. Choose a number d out of the range $\{0, .., n-1\}$, whereas $(e * d)\ mod\ \phi(n) = 1$.

If a message $m < n$ is encrypted with a public key, the cipher text c is calculated by the formula $c = m^e\ mod\ n$. The private key is required for performing the decryption with the formula $m = c^d\ mod\ n$. A similar calculation way is performed if a signature is used. The owner of the private key computes the signature s with the formula $s = m^d\ mod\ n$. The formula $m = s^e\ mod\ n$ is used if the signature must be verified, but it requests the knowledge of the corresponding public key. [77, 81]
Kleinjung et al. analysed the security offered by the RSA algorithm [84]. They concluded that it is not secure anymore to use a 768-bit modulus for RSA and that a 1024-bit modulus will become insecure in the next decades. As recommended by the NIST report, using 2048-bit RSA keys will be secure until 2030 [82]. Assuming a network consisting of resource constrained hardware, such as used in wireless sensor networks, using 1024-bit or 2048-bit RSA keys requires a lot of memory, calculation capacity, and energy for performing the RSA algorithm. For completeness' sake it must be mentioned that researchers from the mathematic field believe that elliptic curves cryptography can be an alternative to RSA. In this dissertation the focus is on RSA, because in the planned application scenario a hardware platform using a trusted platform module is integrated which currently only supports RSA. Thus, it is referred to literature (e.g. reference [77]) for further information about elliptic curve cryptography. [84]
In order to rule the application of asymmetric cryptography, Public-Key Cryptography Standards (PKCS) were developed. The most important PKCS rules the procedure how something is encrypted with RSA, because RSA can be misused, because an attacker is able to guess the content of the RSA encrypted message. An example is given in the book written by Kaufman et al. [78]. The PKCS#1 issues the encryption with correct message formatting which prevents the aforementioned problem with the misuse of RSA. In version 1.5 of PKCS#1 a padding field is inserted that consists of a minimum of eight random nonzero bytes. A detailed description can be found in RFC 2313 [85]. The padding was replaced in version 2 of PKCS#1 by the optimal asymmetric encryption padding (OAEP), because an attack was discovered on protocols such as SSL and TLS relying on RSA PKCS#1 padding. In this attack an eavesdropper was able to recover a key that was encrypted with RSA from a SSL server in around one million messages. The OAEP adds an element of randomness to the message, which makes the deterministic RSA encoding more probabilistic. In addition, OAEP prevents partial decryption of the

cipher text or other information leakage. A detailed description can be found in reference [86] and a security analysis under RSA in reference [87]. [78, 85, 86, 87]

The third security mechanism introduced in this section is the usage of **hash functions** [78]. The task of a hash function is to transform a message m of arbitrary length into a hash *h(m)* with a fixed length. The hash should have the following characteristics [88]: Independent of m, it is computationally easy to calculate *h(m)*. For a given $h(m_1)$ it is computationally infeasible to find another $m_2 \neq m_1$, where $h(m_1) = h(m_2)$. This assumption is known under the term *hash collision*. In addition, a hash function h is called a strong hash function if it is computationally infeasible to find a hash collision for any two $m_1 \neq m_2$. With this characteristic a single bit change in the input should flip approximately half of the bits in the output. Therefore, it is not possible to predict the influence of input changes on the output and an attacker has to observe a large number of messages in order to get an idea of the used *h(m)*. Assuming a hash function with a 128 bit output (e.g. MD5) the attacker requires 2^{64} different messages to find two messages with the same hash value with 50% probability. With today's computers calculation can be performed easily, so that a MD5 is no longer secure. An alternative is the SHA-1 algorithm standardized by NIST, which produces a 160 bit hash value [82]. At the moment SHA-1 is assumed to be secure by offering 80 bits of security. [82, 78, 80, 81]
In order to perform a hash function on input data several rounds of the algorithm are applied. In each round several operations such as bit shifting, binary operations or addition with a constant value are performed. It is not defined how many rounds are performed. The goal is to produce a function that fulfills the aforementioned characteristics of a strong hash and at the same time can be executed quickly with the used software and on the used hardware. [78, 80, 81]
In this dissertation hash functions are an interesting possibility to offer message authenticity and digital signatures. In comparison to the asymmetric cryptography where a complete message is signed with the RSA private key, only the constant length hash value of the message is signed. The result is a faster performance even for longer messages and is also applicable for short messages. Additionally, hash functions ensure message integrity. This can be guaranteed if the underlying hash function is secure. It can than be proven that a 'Hashed Message Authenticity Code' (HMAC) has the following two properties [88]: First, it is impossible to find two inputs that have the same output. Second, an attacker cannot compute the proper hash function without knowing the used key.

3.3.1.2 Public Key Infrastructure

As mentioned in the beginning of Section 3.3 the Internet Security Glossary gives different definitions for security terms. For the term *Public Key Infrastructure* (PKI) the following definition is presented:

> *'A system of Certificate Authorities (and, optionally, [...] other supporting servers and agents) that perform some set of certificate management, archive management, key management, and token management functions for a community of users in an application of asymmetric cryptography.'* [76]

In addition, it defines where trust is derived from during exchange of public keys. The IETF working group Public-Key Infrastructure X.509[3] focused on this topic.

[3]IETF working group Public-Key Infrastructure X.509: `http://datatracker.ietf.org/wg/pkix/`

They defined a complex model for deploying a certificate-based architecture to the Internet dealing with the design of X.509 certificates. The X.509 certificate consists of the following components: A public key of the entities and the identification along with technical information (e.g. certification validation period).
The identity of an entity is established via a common name of the entity (e.g. server's web address, individual name). A certificate authority (CA) signs this certificate with its private key. Two questions arise in this context: First, who can function as a certificate authority? Second, how can the user ensure that the received certificate is not signed by a malicious entity? Kaufman et al. described several solutions for these questions focusing on establishing and extending trust in their book [78]. In the Internet several commercial certificate authorities exist (e.g. Microsoft, Telekom) that are trusted by web browsers. Users are able to add their own trusted authorities to the list (e.g. company, university). Those trustworthy certificate authorities can issue certificates for intermediate certificate authorities that issue a certificate to an end entity (e.g. staff member of company). If an entity receives a certificate, which was signed by an intermediate certificate authority, it does not know that the entity can check its trustworthiness by using its certificate. The entity tries to trace back the certificate to its original certificate authority and can than decide if it is trustworthy or not. This technique is known under the term *chain of trust*. [78]
The performed functionality of the security mechanism PKI also includes publishing of certificates in repositories, revoking certificates, and registering of entities in order to sign certificates. For further information about this functions it is referred to RFC 5280 dealing with Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile [89] and to the IETF working group Public-Key Infrastructure X.509.

3.3.2 (D)TLS Protocol

Due to the rise of IP communication, as described in Section 3.1, the request for secure data transmission occurred. Protocols using TCP on the transport layer preferred the 'Transport Layer Security' (TLS) protocol. It is described in detail in the book 'SSL and TLS Essentials. Securing the Web' written by Thomas Stephen [90]. Because of the inclusion of UDP on the transport layer the TLS protocol showed a drawback. Which means, if a packet loss occurred, the following packets could not be authenticated anymore. In order to solve this problem, the 'Datagram Transport Layer Security' (DTLS) protocol was developed and standardized in RFC 4347 [91]. Developers of the DTLS protocol modified the existing TLS protocol in the areas essential in order to support unreliable transport. [23]
For further understanding a brief introduction to the TLS protocol is given, followed by the design decisions and functionality of the DTLS protocol. In 2004 Nagendra Modadugu and Eric Rescorla published a paper concerning the modification request of the TLS protocol in order to support secure data transmission via the unreliable transport protocol UDP [23]. This reference will be the information base for the characterization of the (D)TLS protocol.

The **TLS protocol** was developed for reliable transport (e.g. TCP) [22]. The protocol provides a secured communication channel for application protocols in order to support server authentication, confidentiality, and message integrity of the communication channel. As an option TLS also offers the authentication of clients if requested. For authentication purposes TLS uses public key based digital signatures supported by certificates. The server's certificate contains the server's domain. The

server has two possibilities to authenticate itself: Either authentication by performing a decryption of a secret encrypted by the server's public key or authentication by signing a short-dated public key. [22, 23]
In comparison to the server authentication the client authenticates itself by signing a random challenge. The certificates of clients contain random identities. In order to establish a secure communication channel, a handshake protocol is performed between the two communication entities. For the TLS handshake a reliable transport protocol is assumed and it is requested to follow the message order. The TLS handshake consists of the following message exchanges [23]:

1. The client sends a `ClientHello` message to the server in order to initiate the TLS handshake. This message includes several information, such as supported TLS version, supported algorithms and compression techniques, and a random nonce.
2. The server answers with three messages in a row:
   - The `ServerHello` message is send to the client in order to announce the server's choice of TLS version, algorithm selection, and a random nonce.
   - The `Certificate` message contains the server's certificate chain.
   - The `ServerHelloDone` message is send to the client in order to announce the completion of message transfer.
3. The client chooses a random `PreMasterSecret`. This secret is used as the basis for each side's keying material. The client sends the following messages to the server in a row:
   - The `ClientKeyExchange` message includes the encryption result of the `PreMasterSecret` under the server's RSA public key.
   - The `ChangeCipherSpec` message is send in order to announce the changing to the newly negotiated protection suite.
   - The `Finish` message contains the Message Authentication Code (MAC) of the previous handshake message. This message is encrypted with the new protection suite (e.g.: AES + SHA-1).
4. The server finishes the TLS handshake by sending two messages in a row:
   - The `ChangeCipherSuite` message is send to the client in order to confirm the change to the protection suite.
   - The `Finish` message concludes the TLS handshake by the server.

Today different applications (e.g. Voice over Internet Protocol (VoIP)) prefer unreliable transport over UDP for performance reasons. Yet, the introduced TLS protocol requires a reliable transport protocol such as TCP. In order to offer the TLS security functionality for unreliable transport protocols, a solution (especially for handling packet loss during the handshake) was essential. The solution was the Datagram TLS protocol, which offers the same security features as TLS only for unreliable packet transport. The DTLS solution is, therefore, also applicable for wireless sensor networks, which prefer UDP on the transport layer. [23]

In comparison to TLS minor, but important, modifications are essential for the **DTLS protocol** in order to transfer the functionality of TLS to unreliable connections such as UDP [23]. The TLS protocol works over TCP and requests reliable,

in-order packet delivery, and replay protection. Those features are not supported if a datagram transport is used. Therefore, this section deals with the requested modifications in order to offer the equivalent security features of TLS for DTLS. [23] Developers of DTLS assume that DTLS records fit within a single datagram in order to avoid fragmentation handling [23]. The advantage of this request is that the DTLS layer does not need to buffer partial records, so that memory is saved and space for other operations (e.g. performing of security mechanisms) is offered. [23]
If fragmentation is supported, problems occur if fragmentation parts are lost. In addition, the question occurred how long the fragments must be buffered before discarding them. The last two mentioned challenges do not occur if only a signal datagram is used. DTLS records can extend the maximum transmission unit. It is no problem, because IP fragmentation is performed and the IP re-assembly is transparently handled by the kernel as pointed out by reference [23]. As a result of the previously mentioned assumption and the resulting advantages the DTLS record format is expanded with a `epoch` and `sequence number` field compared to the TLS record format. The `epoch` numbers protect record payload by verifying the used cipher state. These numbers also prevent ambiguities in case of renegotiation of the session (see example in reference [23]). The `epoch` field is sized to 16 bits. The `sequence numbers` protect against replay attacks, but in comparison to TLS they must be mentioned in DTLS. These numbers are incremented by one for each record. In case of renegotiation of the session, the cipher state can change and the sequence number is reset to zero. The `sequence number` field is sized to 48 bits. The functionality of replay attack protection is performed using replay window mechanisms. Packets with sequence numbers not matching this window are discarded. The DTLS protocol requires other cipher suites than TLS 1.0, basically because of the unreliable connection support and the possible not in-order receiving of records. In TLS version 1.1 the cipher suite AES-128 in CBC mode with SHA-1 and HMACs is proposed as briefly characterized in Section 3.3.1.1. This cipher suite can be used for DTLS and requires an explicit initialization vector for the first encryption operation. For authentication purposes during the handshake RSA can be used as performed in the example for this dissertation. [23]
The performed DTLS handshake is nearly the same as introduced in the previously mentioned characterization of TLS [23]. Two modifications are required for the DTLS handshake in order to prevent denial of service attacks, because of the unreliable transport protocol is used. Those attacks can be prevented by performing a cookie exchange technique before a proper handshake is performed. Here the server provides a cookie that must be replayed by the client. Therewith, the client proves that he is able to receive packets at its announced IP address to the server. The aforementioned TLS handshake is extended by the following messages that are performed before the proper TLS handshake [23]:

1. The client sends a `ClientHello` message to the server including a cookie. The cookie can be an empty cookie or potentially one from a prior exchange.

2. The server has two possibilities to react:

   - If the server is unable to verify the received cookie, the server sends a `HelloVerifyRequest` message to the client in order to check its liveness of the DTLS client. This `HelloVerifyRequest` message includes a cookie (e.g. keyed hash of the clients IP address).

- If the server can handle handshake latency (e.g. accepting session IDs of the client) than the server skips the `HelloVerifyRequest` message and sends the `ServerHello` message directly. Then the TLS handshake starts directly.

3. If the client received an `HelloVerifyRequest` message by the server it responds with a `ClientHello` message analog to the client's starting point in the TLS handshake.

During the handshake the client and the server agree on the key exchange algorithm based on the used cipher suite. The cipher suite itself is a set of encryption and hashing algorithms. If X.509 certificates are used in order to establish an identity, it is required to use negotiated signature algorithms (e.g. RSA) for signing purposes. DTLS prefers to use RSA. An alternative could bethe usage of pre-shared-keys. The resulting key material protects the upcoming message exchange with symmetric cryptography. The key material consists of keys for block cipher, an initialization vector for CBC mode, and a key for the hashed message authentication code (HMAC). Each communication direction has its own key material whereas it is derived from a combination of two keyed HMACs based on the MD5 and SHA-1 hash functions. In order to produce enough key material each HMAC function is applied recursively and its output is concatenated. The finally used master secret is constructed by XORing the outputs from the HMAC chains. [23]
As mentioned in the message exchange prior to the TLS handshake a session ID can be used by the client in order to resume a DTLS session. This possibility allows reducing the communication overhead if an old session is resumed. A detailed description of the reuse of session IDs is presented references [23, 81]. The DTLS handshake requires a reliability mechanisms for a successful performance due to the underlying unreliable transport protocol. Therefore, DTLS includes a simple mechanism. The sender transmits its current set of messages (e.g. message set (2) of TLS handshake consists of three messages); then waits until an answer is received in the handshake from the destination. The destination will only answer the sender if it has received all previous handshake messages correctly. If an interrupt occurs, the handshake has to be performed again. [23, 81]

3.3.3 Trusted Hardware Component

Today's notebooks offer the opportunity to use an integrated *Trusted Platform Module* (TPM) in order to secure a system and its data. A detailed description of the Trusted Computing strategy can be found in the book 'A Practical Guide to Trusted Computing' written by David Challener et al. [92]. The TPM chip is located on the hardware's main board of (e.g. notebook). It supports RSA key generation, built-in RSA encryption and decryption, and secure key storage. The aim of the development of such a chip in cooperation with the software on the system was to increase the reliability and trustworthiness of a system. [92, 81]

As published by the Trusted Computing Group[4] a Trusted Platform Module consists of the following four main components: The cryptographic processor, a microcontroller, is responsible for the RSA operation performance, the SHA-1 hashing, and the random number generation on the chip. Available memory is separated into a persistent storage and a versatile memory. In the persistent storage the firmware

[4]Trusted Computing Group: `http://www.trustedcomputinggroup.org/`

of the trusted computing module, the endorsement key, and the storage root key are securely stored, because they cannot be passed on outside the TPM. On the outside only derivates of the persistent stored keys are used. In comparison to the persistent storage the versatile memory includes the storage itself, the attestation identity keys, and the platform configuration registers. The last main component of a trusted platform module is the component responsible for the secure input and output from the trusted platform module. [92, 81]
The main work is done by the TPM microcontroller. Together with the RSA functionality different keys can be stored. Each key depends on the machine configuration. This machine configuration is build up on the hardware and the software of the machine during booting, which build a chain of trust. The result is the *Storage Root Key* from which different storage keys can be derived. The storage root key is stored in the secure key storage together with information about the system's integrity. In order to encrypt data of the system, the derived storage keys are used which are stored on the notebook's hardware. If arbitrary data is encrypted with such a storage key, they are sealed to the unique status of the system. Access requests are only successful if the TPM check is positive otherwise they fail. [92, 81]

Especially, the previously mentioned opportunity to store the root storage key in persistent storage, requires an attacker to spend considerably more effort and time to extract the stored keys compared to a flash storage. Thus, it is considered that the trusted platform module is tamper resistant. This behavior makes the trusted platform module interesting for wireless sensor network research, because it is assumed that an attacker wants to gain physical access to the deployed sensor nodes. If a trusted platform module is included in the sensor hardware and is used for providing security, it can be used to uniquely identify hardware in the deployed sensor network. The highest type of security can be gained with the integration of such trusted platform modules in encryption procedures at the moment. At the moment the only available sensor node platform including a TPM chip is the OPAL node produced by CSIRO [33] and is integrated in the deployed wireless sensor network in this dissertation.

3.4 Summary and Findings

As described at the beginning of Chapter 3, IP communication is a requirement for the successful integration of wireless sensor networks into the *Internet of Things* [1]. Therefore, different approaches exist to bring IP communication on constrained hardware such as sensor nodes. Section 3.1 gave an overview of three commonly used approaches - ZigBee, 6LoWPAN, and BLIP. All approaches have in common that they use less resources to scale well on different network sizes and to allow stack exchanges below the application layer with less modifications. Concerning security issues it was pointed out that 6LoWPAN supports link and network security in order to support confidentiality and integrity protection [68]. Due to the fact that BLIP is an implementation of 6LoWPAN for the operating system TinyOS, it offers the same security standard as specified in RFC 4919. In order to optimize data transmission within the network itself, the IPFIX protocol was analysed in Section 3.2. Due to its push-protocol character and its template-based design, the IPFIX protocol is interesting for sensor networks. In addition, it can be adapted to different data types (e.g. flow data, sensor data). Today, security concerns (e.g. Who knows or gets access to what kind of information?) in the population grow which requires a secure transmission solution independent of data types. In Section 3.3

security risks are briefly characterized together with defense strategies. The defense strategies focus on security protocols and trusted hardware components dealing with cryptography and existing protocols in order to establish a transport layer security solution, such as (D)TLS.
The characterized solutions in this chapter and the background information on wireless sensor networks will inspire the design decisions for the developed solutions in this dissertation, which are presented in the following Chapters 4 to 5.

Regarding the mentioned research questions on efficiency the following contribution has been made in this chapter:

(E1) Is the IP Flow Information Export (IPFIX) protocol a viable solution for transmission of sensor data in wireless sensor networks?

> Yes. As assumed in Chapter 3 IP communication is supported by constrained hardware and it became interesting to transfer standardized protocols to constrained hardware as used in wireless sensor networks. The IPFIX protocol was introduced in Section 3.2. The message structure of the IPFIX protocol is very interesting for wireless sensor networks, because it separates meta information and data in different messages. It works over UDP, which is the preferred transport protocol for sensor networks. IPFIX is flexible concerning its message structure and reduces retransmissions of known information (e.g. meta information). A drawback is the additional overhead of 20 bytes caused by the IPFIX headers, which reduces free space in the payload of each message. But this drawback can be solved as described in Chapter 4.

Regarding the mentioned research questions on security the following contribution has been made in this chapter:

(S1) Is it possible to secure data transmission in wireless sensor networks with known standards from IP networks?

> Yes. Concerning security Section 3.3 gave a brief overview of solutions used in IP networks in order to secure data transmissions within a network. In order to answer research question S1, selected security solutions focused on security protocols (cryptographic functions, public key infrastructures), (D)TLS protocol, and trusted hardware component using a trusted platform module. Throughout this section it became obvious that not every solution is interesting and contributing to wireless sensor networks, because of limited resources, especially in memory and computational capacities. But the given overview influenced the design decision for the realized security solution in this dissertation and is presented in Chapter 4.

4. Design Decisions and Specifications

As described in Chapter 2 and 3, wireless sensor networks are specialized IP networks where the components are resource limited regarding power supply, memory, and computational capacity. The application range is manifold and usually measured and transmitted information includes sensitive data, so that only authorized entities are allowed access. This fact becomes very important, because today's established wireless sensor networks usually work over IP due to the integration into the *Internet of Things* [1]. As pointed throughout Chapters 3.1 to 3.3 different solutions in common IP networks exist, which are very interesting for sensor networks in order to achieve their requirements for IP communication and efficient data transmission support within the network together with secure data transmission between different entities. But those solutions are too bulky for the limited hardware. The solution for this challenge is a restriction of algorithms to address computational and memory limitations and compression (e.g. headers) to reduce network overhead.

In the upcoming sections of this chapter interesting protocols for wireless sensor networks dealing with data transmission (cf. Section 4.1), in-network aggregation (cf. Section 4.2), and security issues (cf. Section 4.3) are presented. Selected protocols inspired the design of the protocols developed during this doctoral thesis. This chapter includes the required adaptations of the protocols for the constraints of the wireless sensor network's hardware. The final implementation is described in detail in Chapter 5, followed by the evaluation and comparison to related protocols in Chapter 6. In Section 4.4 it is motivated why a special graphical user interface was developed and how it visualizes the deployed wireless sensor network. A proof of concept of the graphical user interface functionalities is shown in the evaluation chapter.

4.1 Efficient Data Transmission

A possible way to save resources is the improvement of data transmission. In a wireless sensor network, data is transmitted together with its meta information resulting in relatively big messages; sometimes including redundant information (e.g. data source). Those messages need more resources, especially energy, for transmissions due to long periods of full radio activity. In order to solve this problem an efficient data transmission protocol was developed in this dissertation with

focus on reducing redundant information (e.g. meta information). Related works in wireless sensor networks dealing with this topic are COAP and sMAP, briefly characterized in Section 4.1.1 as inspiration for the development of the TinyIPFIX protocol. The IPFIX protocol known from common IP networks is also interesting, but currently does not fit the limited resources. Therefore, an adaptation is essential, which results in the implemented TinyIPFIX protocol developed during this doctoral thesis. TinyIPFIX will be presented in Section 4.1.2.

4.1.1 Related Protocols

Chapter 3 motivated the usage of standards in wireless sensor networks. It was pointed out that the main task for the components is data collection, followed by transmission towards the sink. The standard push-protocol IPFIX is an appropriate candidate for efficient data transmission, because it has a template-based design [20]. Data values and meta information are sent in separate messages. Another advantage is the high flexibility of the protocol gained by minimal configuration requirements (e.g. new templates based on sensor's meta information) when new vendors are integrated. Those characteristics can also be found by the XML based COAP and the JSON based sMAP implementation. Both approaches are inspired by the 'Representational State Transfer' (REST) architecture developed for web services [93, 94]. Sections 4.1.1.1 to 4.1.1.2 briefly describe protocols, which inspired the design decisions for the developed transmission protocol TinyIPFIX in this dissertation. A comparison to these protocols will be presented during the evaluation in Section 6.1.4.

4.1.1.1 Constrained Application Protocol

Today's development of networks with machine-to-machine applications such as building automation has increased in lock step with web service applications. The working group Shelby et al. developed the *Constrained Application Protocol* (COAP), which is undergoing the standardization process by the IETF at the moment [95, 96]. This protocol is an implementation of the REST architecture for constrained hardware used in wireless sensor networks. COAP itself works over UDP with a simple retransmission mechanism. Retransmissions are reliable due to unique message IDs (URI) for each request. COAP requests are equivalent to HTTP requests using XML. It offers functionalities for interactions between applications and end-points following the REST Method/Response model [93]. Resource discovery is possible and different basic web concepts, such as URIs and content-types, are supported. Those functionalities allow COAP an interaction with HTTP causing less overhead fitting constraint hardware requirements and offers compression possibilities. COAP supports four message types which are indicated in the header field T: `Confirmable message` (CON), `Non-Confirmable message` (NON), `Acknowledgment message` (ACK), and `Reset message` (RST). [93, 95, 96]
The COAP header is four bytes long and includes a `version` field (Ver), `type` field (T), `option counter` field (OC), `code` field (code), and the `Message ID` field, followed by `options` and individual payload, which are both optional as illustrated in Figure 4.1. Version field determines the COAP version used. As marked with dashed lines the fields `Options` and `Payload` are optional and their amount is indicated in the OC field. The code field indicates if a message is a request, a response or if it is empty. More details of messages and code types can be found in reference [96].

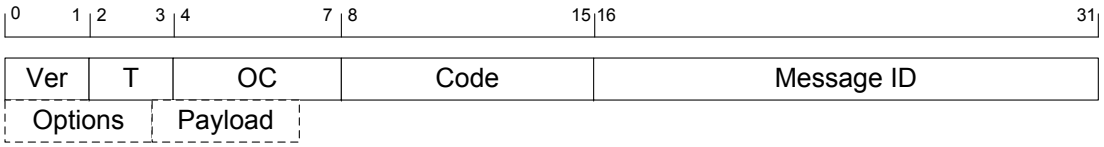
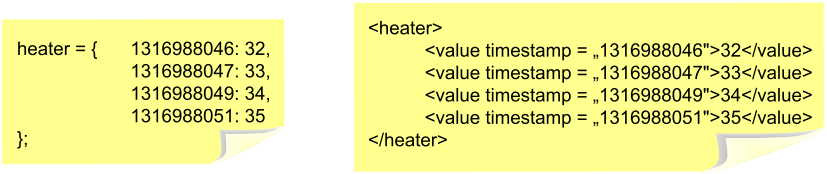


Figure 4.1: COAP message format [bits]

A sample COAP message flow between two clients (A and B) and a server, requesting different data, looks as follows: Client A requests a temperature value and client B a light value. Both clients send out a message of type CON with individual Message IDs, GET request and individual Tokens. In the case of client B, the request cannot be fulfilled and causes the server to answer directly. The server responds with an ACK message referring the specific Message ID, Token, and the search result. In the case of client A, the server needs some time to look up the answer. In order to prevent retransmission of the request by client A, the server answers with an ACK only. After the server looked up the answer for the request, it composes the appropriate message including the reference token, the content, and a new Message ID. Client A responds to the server's message with an ACK and Message ID. [96]

4.1.1.2 Simple Measurement and Actuation Profile Protocol

The wide variety of physical information (e.g. water, weather, occupancy, environment) and the call for efficient data exchange motivated Dawson-Haggerty et al. to develop the *Simple Measurement and Actuation Profile* (sMAP) protocol in 2010 [97]. The design of sMAP is inspired by REST web services. REST defines a set of architectural principles - known as RESTful, which a web service must be aware of. In RESTful architectures each device provides a special RESTful web service. HTTP is used for communication where standardized URLs are used. sMAP uses the ideas of RESTful web services which allow direct publishing of data. sMAP itself is located between producers of physical information and consumers known as applications (e.g. storage, debugging, virtualization, authentication). [97]



(a) JSON format (b) XML format

Figure 4.2: Message format comparison

In comparison to COAP, sMAP uses the *JavaScript Object Notation* (JSON) format for object exchange. JSON utilizes a common programming language pattern which simplifies the implementation. Figure 4.2 illustrates a structure comparison between JSON and XML messages. The JSON message shown requires only 72 characters in comparison to the XML message with 177 characters. This short example shows the size advantage of the JSON format.

In addition, sMAP uses an *embedded binary HTTP* (EBHTTP) version to match sensor hardware requirements. Supported methods are GET, POST, DELETE, and PUT. In comparison to HTTP, EBHHTP produces minimal transport and

space overhead, delivers without an acknowledgment mechanism, and eliminates unused headers. Those devices can have several resources. Web services with sMAP scale with millions of clients which makes it very interesting for wireless sensor networks. [97]

4.1.2 Design Decisions for the TinyIPFIX Protocol

The introduced protocols COAP and sMAP showed that the compression of meta-data in sensor networks help to obtain the transmission efficiency. They also prefer support of HTTP, TCP/IP, and web services in order to publish collected data in the Internet in order to give authorized persons or organizations access to it. Both protocols scale well in networks with hundreds of entities. In comparison to those protocols the standard IPFIX prefers the reduction of network overhead by separation of data and meta information into different messages. This solution is practical in networks with redundant information as it is used in sensor networks where each message includes data and meta information. Meta information is often redundant and, therefore, it is assumed in the wireless sensor network in this dissertation that for each sensor node its meta information stays the same over long period of time; regardlessly of sensing intervals [20]. An update of meta information can only happen if other sensor combinations are activated on the sensor node. For example, the sensor board MTS300 for sensor node IRIS offers the sensor combinations `<light, sound>` or `<temperature, sound>` [31]. Another example of a meta information update is the change of aggregation functionality as described in Section 4.2.2.
Today it becomes interesting to use well-known and widely-used network standards as motivated throughout Chapter 3, which calls for an adaptation of those protocols to the new requirements of wireless sensor networks. Due to the previously inspired design decisions by the application protocols COAP and sMAP and the requirements occurred due to IP communication support the favored standard should support data separation and work with TCP/IP. As introduced in Section 3.2, the flow protocol IPFIX is an interesting protocol for a wireless sensor network due to its push-protocol characteristic, template-based design, flexibility, and high efficiency. Before it can be used in wireless sensor networks the protocol has to be modified corresponding to the resource constraints such as limited message size as realized within the scope of the bachelor thesis by Thomas Kothmayr [98].

In the wireless sensor network experiments performed in this doctoral thesis the maximum transmission unit is limited to 127 bytes due to the RF transceiver CC2420 of the hardware [18] and the used BLIP stack as described in Section 3.1. Here 12 bytes are spent for the TinyOS message header (cf. Figure 5.3) and the rest for individual payload [25]. Normally, the payload consists of measured data values, its meta information, and some information about the source node such as ID and a local time stamp. In the developed application protocol *TinyIPFIX*, the template-based design of the IPFIX protocol is used, which means that the individual payload in the TinyOS message consists of an IPFIX message [20]. The IPFIX message can either consist of a Template Record or a Data Record together with required IPFIX headers (cf. Figure 3.3). The Template Record includes the meta information of the data and the Data Record includes the data values itself. [98]

The design space of the developed TinyIPFIX protocol consists of four areas that need to be adapted to fit the requirements and constraints of a cyber-physical system [12]: standardization, resource efficiency, usability, and flexibility.

The first area is called *standardization*. Sensor devices measure data, which have a specific format and must be represented accordingly. This representation has to be general and universal, meaning that a protocol should be able to uniquely distinguish each measurement type. A measurement type is defined as a reading from a specific model of a sensor, which carries information about data type, accuracy, precision, and conversion to scientific units, rather than a dangerously simple description such as temperature. In the case of TinyIPFIX, the sensor measurement type is identified by an individual Type ID and Enterprise Number (EID), which are registered with the Internet Assigned Number Authority (IANA) [74]. This ensures adaptability to other platforms or new measurement types. Since an TinyIPFIX Template only carries syntactical meta data for the measurements sent in an TinyIPFIX Data packet, the semantics for that data still needs to be published. If the Enterprise and Type IDs have been labeled globally unique, a public repository for this semantic data, presented as XML markup, becomes feasible.
The second area focuses on *resource efficiency*, because resources of sensor nodes are limited in terms of power, memory space, and computational capacities (cf. Figure 2.2). TinyIPFIX is evaluated with regard to its memory requirements and energy consumption. Additionally, a TinyIPFIX-Aggregation framework is implemented, which offers in-network aggregation mechanisms for data pre-processing. By leveraging in-network aggregation, additional energy savings can be achieved through transmission reduction.
Third, the development needs to deal with *usability*. The benefits of using IPv6 in sensor networks were detailed in a previous work presented by Jonathan Hui and David Culler [63]. For the presented implementation in this doctoral thesis it was chosen to send TinyIPFIX packets via the BLIP implementation [99], which supports IPv6 and UDP. With this setup the established wireless sensor network can smoothly be integrated into an existing IP-based network infrastructure. Such an infrastructure can be the system developed in the project AutHoNe[5], which provides different functionalities such as knowledge sharing and remote access, as well as an autonomic mechanism to integrate new devices into building networks [13]. Another example are online visualization tools, such as COSM, which require internet access for a live upload of data.
The forth area focuses on *flexibility*. Implemented solutions, such as TinyIPFIX, must be flexible in order to support different application areas and vendors at the same time. This performance request is proven by runs of numerous real world tests of TinyIPFIX in building application scenarios on different vendor hardware as well as in a large wireless sensor network deployment on the Harvard Sensor Network Testbed (Motelab) [100].

Before implementing the TinyIPFIX protocol on sensor nodes some adaptations are necessary. Type ID and the Enterprise ID identify sensor measurement data. In order to enhance interoperability, they need to be standardized. For today's home networks, typical environmental data such as temperature, brightness, and humidity can be measured by sensor nodes. Therefore, standard Type IDs can be issued. Until now, Enterprise IDs for sensor data did not exist, and, therefore, they had to be chosen and registered by IANA [74]. An example for the used hardware (e.g. IRIS, TelosB, OPAL) in the deployed wireless sensor network is shown in Table 4.1. The example shows that the Enterprise ID is independent of hardware

[5] Autonomic Home Networking Project partly funded by the German Federal Ministry of Education and Research under grant agreement no. 01BN070[2-5].

platform and platform vendor, but depends on sensor and vendor of the technical unit. [20, 98]

Hardware platform	Platform vendor	Sensor	Vendor technical unit	Enterprise ID	Type ID
TelosB	Advantic Sys.	Temperature	Sensiron SHT11	3841	33025
TelosB	Advantic Sys.	Humidity	Sensiron SHT11	3841	33026
TelosB	Advantic Sys.	Light	Hamamatsu S1087	3845	33025
IRIS	Crossbow Inc.	Temperature	Panasonic ERT-J1VR103J	3843	32771
IRIS	Crossbow Inc.	Light	TAOS TSL2550	3846	33282

Table 4.1: Exemplary information for IANA registration

Following registration, the new Enterprise ID can be used to identify sensor node measurement data. At that point, new Type IDs can also be used to describe typical sensor data, like temperature or brightness values. Semantics and type length need to be included in the ID standardization in order to ensure interoperability. The next generation of sensor nodes will have the ability to measure other types of data (e.g. gas concentration), which will result in new IDs. However, the common base for transmitting data is still IPFIX, which also supports the interoperability between devices due to the fact that it is not influenced by hardware specifications. [20, 98]

When essential modifications, such as header compression as described in Section 4.1.3, are done, sensor nodes act as Exporters and transmit their measurement data using TinyIPFIX in a wireless sensor network. When a sensor node boots up, it has to announce its Template Record before sending its measurement data to the Collector (cf. Section 3.2). This has to be done only once, because a Collector buffers the Template Record to decode Data Records. Depending on the chosen application the network structure may change. In order to ensure a successful decoding in this case, the sensor nodes repeat their Template Records periodically. Data Records do not have to contain anything but the measurement data, because all meta information has already been sent in the Template Records. A short header containing the number of transmitted values and the referenced Template ID only accompanies Data Records. This Template ID verifies the Template Record used. Several Data Records (e.g. light and temperature) can be put into a single message until the maximum transmission unit is reached. All records within a packet, which can be decoded with a single template, form a Data Set as shown in Figure 3.3. [20, 98]

As described in Section 3.2 the original size of the IPFIX header is 20 bytes. In order to transmit as much data as possible in one message, it is essential to reduce overhead caused by additional headers. Therefore, the IPFIX message header must be reduced to the absolute minimum size necessary for TinyIPFIX. Compression techniques to accomplish this goal are presented in Section 4.1.3. [20, 98]

4.1.3 Specification of Header Compression Techniques

Three compression techniques for IPFIX headers (IPFIX message and Set header) were developed and implemented in this dissertation. The defensive compression approach is an initial run to reduce overhead, followed by a modified compression

version (cf. Section 4.1.3.2). Finally an aggressive compression approach is introduced in Section 4.1.3.3, which reduces overhead of the additional headers (IPFIX message header, Set header) by 85%. The compression techniques are described in the upcoming sections related to references [101, 98, 20].

4.1.3.1 Defensive Compression

The first technique is based on the idea of introducing a pre-header that specifies different field sizes separately, which results in a pre-header size of two bytes as shown in Figure 4.3 [101, 20, 98].

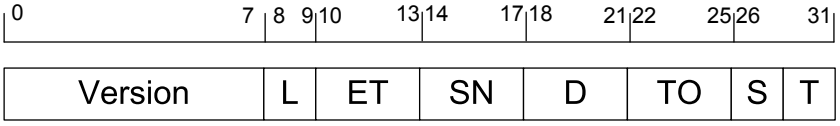


Figure 4.3: TinyIPFIX pre-header: defensive header compression [bits]

The `Version` field is shortened down to five bits. This is possible, because the IPFIX protocol used today is version 10. The size defined here allows a IPFIX protocol support up to version 31. The field `Length` (L) is one bit long which results in two possible values. The field L identifies the length in the following header where zero means one byte, and one means two bytes. The same size coding adjusted to two bytes is given to the fields `Export Time` (ET), `Sequence Number` (SN) and `Observation Domain ID` (D), which can stand for a maximum field size of four bytes in the following header. The `Template Offset` (TO) field is two bits long. If $TO = 0$, the decoder node should use the last received Template Record for the decoding purposes of an incoming Data Record. If $TO = 1$, the previous Template Record is used. If $TO = 2$, the pre-previous Template Record is used. In case the TO is given for a Data Message, two bytes for the SetID can be saved. If $TO = 3$, the field is ignored and a proper statement of the Template ID is expected in the header. The `Single Set Flag` (S) field is one bit long and indicates if the TinyIPFIX message consists of one single TinyIPFIX set. If only one set is expected, the explicit set length statement in the header can be omitted, because it can be recalculated from the total message length. The pre-header is concluded by the one bit long `Template Set Flag` (T) field. If $T = 1$, the first set in the IPFIX message is a Template Set with $SetID = 2$, which results in an omission of two bytes for Set ID definition. [101, 20, 98]

The worst case scenario results in a full sized pre-header with two bytes, because all fields have their maximum field size and nothing can be omitted. In total, the header for TinyIPFIX has the size of 22 bytes (two bytes pre-header plus 16 bytes IPFIX message header plus four bytes IPFIX Set header). In the best case scenario, all header fields can be fitted to one byte and the Set Header can be omitted as shown in Figure 4.4a. [20, 98, 101]
It is obvious that message `Length` and `Observation Domain ID` can be shortened to one byte. If more compression is required to fit the message in 102 bytes, it is sometimes possible to find and remove duplicate values in the original message. Usually, the field named `Observation Domain ID` refers to Node ID with a single byte, which allows up to 256 nodes. Given that data rates are usually low in wireless sensor networks, even if the counter `Sequence Number` rolls over after 255 messages, there is no detrimental impact. Thus, this field can be shortened to one byte. Depending on the application it is essential to transmit a timestamp to make reordering of the

messages at the sink and the analysis of other nodes possible. In general, sending a starting timestamp solves this problem. This allows following timestamps from the same node to be recalculated based on the known measurement and sending time intervals. The same strategy allows reduction of the `Export Time` field size to one byte as well. In the best case, the type of field recalculation strategies makes it possible to reduce the IPFIX header size from 20 to six bytes in the defensive compression approach. [20, 101, 98]

4.1.3.2 Modified Defensive Compression

Figure 4.4b shows more modifications in the pre-header, which reduce the overall IPFIX headers from 20 bytes down to three bytes by omitting fields reserved for `Export Time` (ET), `Observation Domain` (D), `Set ID`, and `Set Length` [101, 98]. In comparison to Figure 4.3 the `Version` field in the first step is deleted. The fields L, ET, S, and D have the same functionality and dimension as before. The field `Template Offset` is dropped followed by the field `Single Set Flag` as before. The field `Template Set Flag` is replaced by a new field called `SetID Lookup Index`, which is only present if the `Single Set Flag` is set. Normally, IPFIX Exporter provides only a limited number of Templates, which makes sizing of this field to a two byte size sufficient. The `SetID Lookup index` therefore tries to fit the most commonly used values for the SetID into one byte. Its value is interpreted as follows [98]:

- `SetID Lookup Index` = 0: SetID lookup is ignored, and a proper SetID definition follows after the IPFIX message header.
- `SetID Lookup Index` = 1: SetID = 2 is given as two, thus message contains a Template Set.
- `SetID Lookup Index` = {2..63}: reserved.
- `SetID Lookup Index` = {64..255}: SetID is given as 192 plus value when converting to normal IPFIX. The message contains a Data Set referencing the according template.

From now on the resulting compressed TinyIPFIX header is referred to as the modified defensive approach.

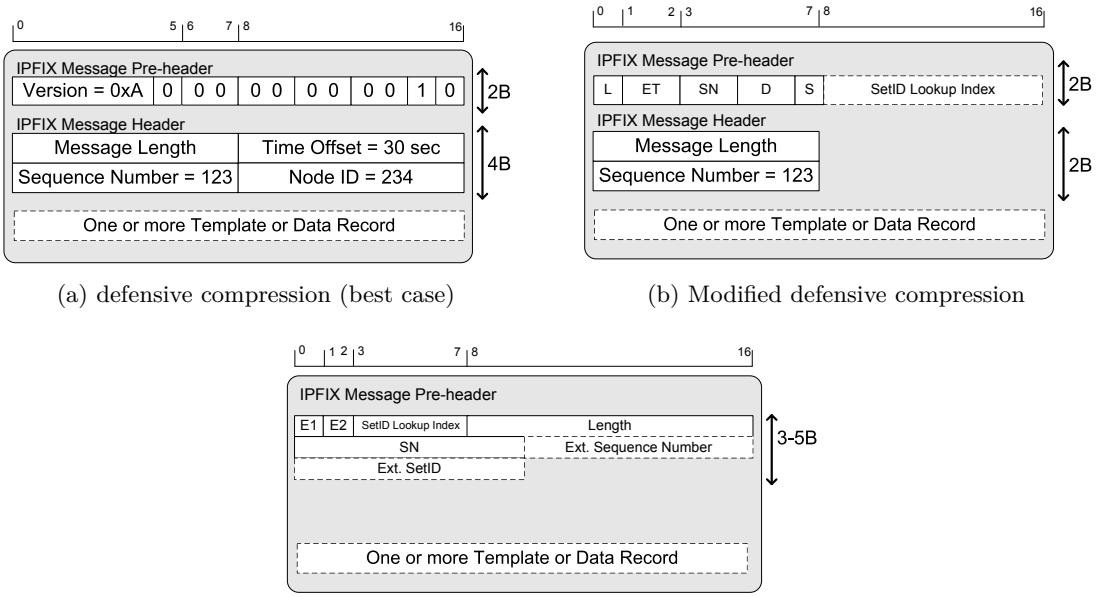
4.1.3.3 Aggressive Compression

The third more aggressive compression approach reduces overhead once more by one or two bytes as shown in Figure 4.4c [101, 98]. It starts by limiting the capabilities of IPFIX to those required in wireless sensor networks. The IPFIX packet length is limited to 1,024 bytes, which exceeds the maximum transmission unit of 127 bytes defined by the IEEE 802.15.4 standard used by the RF transceiver CC2420 [18] and, therefore, requires packet fragmentation [69]. In this activated header compression approach, only one set of templates or data is transmitted. [101, 98, 69]

Figure 4.4c shows the pre-header of the aggressive approach where the `SetID Lookup Index` field is moved even further forward within the header and shortened to four bits. It acts as a lookup field for the SetIDs. The bits marked `E1` and `E2` control the presence of the field `Ext.SetID` and the length of the field `Ext.SequenceNumber` respectively. If $E1 \neq 0$ and $E2 \neq 0$, those optional fields are expected. Due to the

low sampling rate in typical wireless sensor networks, the `Sequence Number` field has been shortened to one byte. Since TinyIPFIX packets are always transported via a network protocol, which specifies the source of the packet, the `Observation Domain` can be equated with the source of a TinyIPFIX packet and the field can be dropped from the header. The specification of a 32-bit time stamp in seconds would require the time synchronization across a wireless sensor network and produces too much overhead. Thus, the `Export Time` field is dropped here. A detailed decoding example is given in Section 5.1.1. [101, 98]

For common wireless sensor network settings as used in this dissertation, the typical header size of TinyIPFIX headers is three bytes in total if the aggressive approach is performed. Thus, a compression level of 85% is achieved as compared to the standard IPFIX message and Set header shown in Figure 3.4. In Section 5.1.1 the resulting message structure under TinyOS assuming a MTU of 102 bytes is illustrated in Figure 5.4.



(a) defensive compression (best case)

(b) Modified defensive compression

(c) Aggressive compression

Figure 4.4: TinyIPFIX message structure comparison [bits]

4.2 In-network Aggregation

Section 2.3 introduced the need for energy saving methods (e.g. activity modes, software based methods) due to limited resources of sensor nodes. This section introduces in-network aggregation as a software solution to save resources, especially energy. Different aggregation types exist, which all result in transmission reduction. Aggregation performance costs energy for computation, but it reduces message flow. In the field of in-network aggregation two main approaches exist today: (1) message aggregation and (2) data aggregation. The latter is also called *data pre-processing* in literature. [102]

The first approach is a combination of two or more messages into a new aggregate message without data pre-processing. The second approach evaluates data transmitted from one sensor node to another and computes aggregate messages with this data based on value type requests and aggregate functions. In general,

this can be performed centrally at the sink. Depending on the chosen application (e.g. only average values are requested) it can be an advantage to perform data aggregation within the sensor node network iteself. As a consequence, the number of transmissions to the next hop is reduced and less energy is consumed. In order to achieve in-network aggregation, sensor nodes with required resources (e.g. computation capacity, memory) must be available in the network.

In this dissertation an in-network aggregation technique called TinyIPFIX-Aggregation is presented, which was implemented within the scope of the bachelor thesis by Benjamin Ertl [102]. The remainder of this section briefly characterizes different aggregation algorithms in Section 4.2.1 as inspiration for the development of TinyIPFIX-Aggregation framework. TinyIPFIX-Aggregation will be compared to related work during the evaluation in Section 6.2.4. Finally, design decisions will be discussed with respect to the TinyIPFIX-Aggregation framework implementation in Section 4.2.2.

4.2.1 Related Aggregation Techniques

In this section the in-network aggregation techniques TAG, AIDA, and SIA are briefly characterized. Those techniques are representatives for a large number of aggregation techniques developed for wireless sensor networks. The hardware of the used sensor node defines the applicable aggregation technique, because each technique requires a different amount of resources. The techniques presented in this section inspired the development of the TinyIPFIX-Aggregation framework.

4.2.1.1 Tiny AGgregation Service

In 2002 Madden et al. presented a *Tiny AGregation* (TAG) service for ad-hoc sensor networks [103]. This approach is based on in-network data pre-processing, because applications often depend more on data aggregations rather than raw sensor data. Because the structured query language (SQL) offers an intuitive way to formulate aggregation queries, TAG uses a syntax similar to SQL. This solution allows the user to express simple, declarative queries and have them distributed and executed efficiently in low-power networks. Today ad-hoc network devices can identify each other and route data without prior knowledge or assumptions about the network topology. Thus, it does not matter if the network topology changes. In the eyes of researchers, aggregation is a central task and should be provided as a core service by the system's software. [103]

The aggregation queries are intelligently distributed and executed in a time and power-efficient manner. In conclusion, the idea of TAG is to process aggregates in the network by processing data as it flows through sensors, discarding irrelevant data, and combining relevant readings into more compact records when possible. Users post aggregation queries from a powered, storage-rich base station. Operators that implement the query are distributed over the network by piggybacking on the existing ad hoc networking protocol. Sensors route data back towards the user through a routing tree rooted at the base station. As data flows up this tree, it is aggregated according to an aggregation function and value-based partitioning specified in the query. [103]

Figure 4.5 shows the setup of a wireless sensor network consisting of six nodes where each node records values for temperature and light [103]. The measurements are forwarded towards the root (here: $NodeID = 1$). The expected data flow is

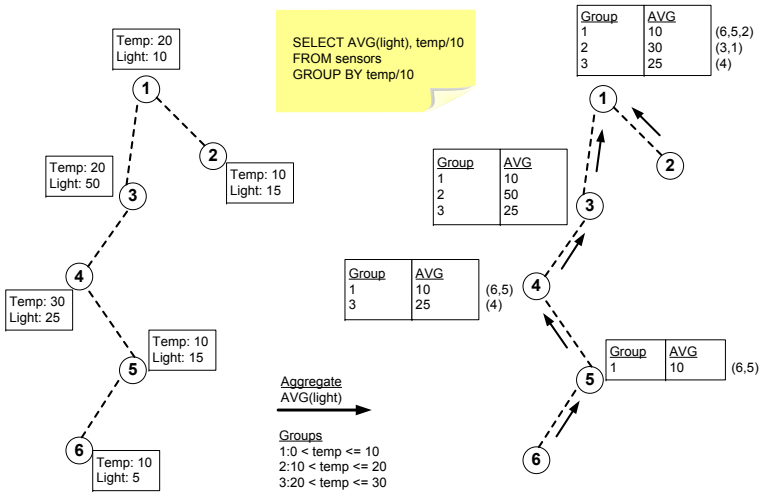


Figure 4.5: Wireless sensor network performing TAG

shown in the left part where no pre-processing of data is performed. In contrast the right part shows the same network with grouped aggregate applied to the setup as performed by TAG. The nodes with $ID = \{1, 3, 4, 5\}$ show the tables running in the background on each node when TAG is performed. The tables include information about the group affiliation (column 1) and the AVG value (column 2). Next to those tables parenthesized expression can be found including node IDs that contribute to the average. [103]

As mentioned before, TAG needs an ad-hoc routing algorithm. TAG must be able to deliver query requests to all nodes in a network. TAG must also be able to provide one or more routes from every node to the sink, where aggregation data is being collected. Those routes must ensure that at least one copy of the message arrives at the sink. [103]

4.2.1.2 Adaptive Application-Independent Data Aggregation

He et al. developed an aggregation mechanism called *Adaptive Application-Independent Data Aggregation* (AIDA) for wireless sensor networks [104]. AIDA isolates aggregation decisions into a module and is a member of a group of novel aggregation algorithms. This module - *AIDA* - is located between the network and the data-link layer in a common stack. The existing MAC and network-layer protocols in the wireless sensor network can be used without any modifications. [104]

In comparison to other aggregation mechanisms, the AIDA approach differs by the following behavior: An isolation of aggregation decisions from applications is performed by integrating an intermediate layer, which can perform adaptive aggregation. The component is generalized. It can be used for a wide range of applications and data types without causing rewriting operations for support reasons. Timely delivery and protocol overhead is taken into account to adjust strategies for aggregation depending on network traffic and equipment requirements. The upper layer is allowed to decide how the information is compressed in order to make a lossless aggregation performance possible for AIDA. As a result of those behavior strategies, AIDA has the ability to concatenate network units into an aggregate. Therefore, AIDA uses a novel adaptive feedback scheme to schedule the delivery of those aggregates to the MAC layer for transmission. The goal of the AIDA approach is to maximize utilization of the communication channel with energy savings coming as

an ancillary benefit. With significant costs incurred from channel contention, packet header overhead, and data padding for fixed sized packets this approach abates. Such costs vary depending on data aggregation degree at forwarding nodes in accordance with current local traffic patterns. [104]

The following design decisions, among others, were important for the AIDA development. Depending on the application, it is essential to properly aggregate named data from a common source; one must associate both: location and time in order to ensure that information is not lost or inappropriately merged. Any aggregation performed must, therefore, be time and direction sensitive in order to ensure that data received at the requester remains meaningful. In general, more aggregation is always better. But traffic levels vary, if aggregation performance varies as communication and throughput are optimized. Therefore, AIDA utilizes feedback control based on network traffic conditions when making aggregation decisions to adaptively optimize bandwidth while minimizing system energy consumption, which is underexploited by previous aggregation schemes. Based on the design decisions AIDA performs aggregation transparent to other components, which allows AIDA to function independently of other protocols. In general, packets are aggregated through network unit concatenation. The resulting overhead is eliminated by the aggregation component, which combines different network units into a single outgoing AIDA payload. [104]
The final implemented architecture is based on the idea of separating the functions into two components: *AIDA Aggregation Function Unit* (AFU) and *AIDA Aggregation Control Unit* (ACU). The AIDA AFU aggregates and disaggregates network packets, so called units. In comparison the AIDA ACU manages the control timer settings and fine tunes the specified aggregation degree. It is responsible for the decision of how many packets to aggregate and when to invoke such aggregation. It is a feedback-based adaptive component, which makes online decisions based on local network conditions. The AIDA ACU is responsible for the available aggregation schemes [104]. In mode `No Aggregation` a normal network stack is used, because packets are passed directly to the MAC protocol. If the mode `Fixed Scheme` (FIX) is performed, each AIDA payload consists of a fixed number of aggregated network units, which means degree of aggregation (DoA) is equal to N_{FIX}. The resulting packet will be passed to the MAC layer. In the AIDA output queue monitoring case, mode `On-Demand Scheme` is performed. Here it is ensured that there is always an AIDA payload resident for MAC layer dequeuing and transmission. The last aggregation mode - `Dynamic Feedback Scheme (DYN)` - is a combination of the mode *On-Demand Scheme* and *Fixed Scheme (FIX)*. In the *DYN* mode the degree of aggregation (N_{DYN}) threshold is adjusted dynamically corresponding to the traffic conditions in the network. [104]
In AIDA, different transmission possibilities are supported as well. If only one network unit is ready, unicast is used. If all network units have the same destination, AIDA aggregates them with the same destination specified and performs manycast. Multicast is performed if network units have different destinations, be aggregated and use the MAC broadcast address as the destination. Each receiver determines individually which units are needed. The fourth possibility is broadcast where all aggregated network units are broadcast messages. [104]
The packet format is modified in the way that AIDA adds meta information to a packet in the form of a header. In this header the aggregation format is specified. It is placed in front of all aggregated network units and is included in the AIDA

data units, which are passed to the MAC Layer for transmission. A detailed AIDA header analysis can be found in reference [104].

For evaluation purposes it can be observed that AIDA itself would cause header overhead. Due to aggregation of several network units into one MAC payload, the overhead is reduced. Resource saving can be gained by aggregating multiple upper layer payloads into a single transmission. The actual savings concerning transmission costs depend on the chosen transmission type.

4.2.1.3 Secure Information Aggregation in Sensor Networks

The *Secure Information Aggregation* (SIA) technique was developed by Przydatek et al. in 2003 [105]. The idea of this approach is to have so called aggregator nodes in the network. Collected raw data from the sensors will be processed locally and the aggregator will transmit the result to the remote user. Research has to face the possibility of physical tampering as well as limited resources in such a network. Security functionality is also necessary in the network. SIA achieves all goals, because it performs aggregation and validation. The approach is known as aggregate-commit-prove. [105]

The SIA configuration consists of several nodes deployed around a home server. The sensors collect data and transmit them to a home server, which is responsible for a statistical analysis. Commonly, network nodes lack enough resources to transmit data over long distances. Sinks, therefore, often exist as an intermediary between home server and sensor nodes. In general, sinks have more resources than nodes, which collect data. Therefore, they are good candidates to perform aggregation functionality. In SIA the aggregator forwards all data to the home server and authenticates the information from each sensor. The home server then can verify everything and can perform an analysis. [105]
The SIA approach aggregate-commit-prove consists of the following steps: First of all the aggregates, such as MAX, MIN, and AVG, of sensor nodes' raw data are calculated by aggregators. In the next step, the aggregators reply to the home server. The reply consists of the aggregation result and a commitment to the data collection. The correctness of results is essential. Therefore, the home server and the aggregator perform effective interactive proofs. Forward secure authentication is performed in SIA in order to ensure authentication. In this technique each sensor node shares an individual key with its home server. This key is updated at the beginning of each time interval using a one-way function. The updated key is used to compute the MAC on sensing data during that time interval. [105]

4.2.2 Design Decisions for TinyIPFIX-Aggregation

One technique for saving energy and computational capacities is in-network aggregation as motivated in Section 2.3 and published in reference [106]. Aggregation calls for so called *aggregator nodes* in a wireless sensor network. In general, those aggregator nodes have more resources and are located at selected positions within the network as assumed by the previously introduced aggregation techniques in Section 4.2.1. Depending on their aggregation technique and algorithm structure they require large resources of memory, computational capacity, and energy. For example, TAG requires memory for hosting the tables on each node. Whereas AIDA requires memory for the AIDA unit itself, which expands the network stack due to its plugin between network and MAC layer. In comparison SIA requires computational capacity in order to perform its security authentication procedures. All

known aggregation techniques have the reduction of resource usage to a minimum in common in order to perform on limited devices. [105, 104, 103, 106]

One possibility of aggregation is to bundle a number of individual messages into one message without any modifications (e.g. pre-processing). Depending on the application this approach is combined with pre-processing techniques within the network, where an aggregation function (e.g. MAX, MIN, AVG) is executed on the data (e.g. temperature). This global idea was transferred to the developed TinyIPFIX protocol in order to build the extension *TinyIPFIX-Aggregation framework*. The following two aggregation techniques are implemented in the TinyIPFIX-Aggregation framework related to reference [102]:

1. *Message Aggregation*: Aggregation of several data messages in one packet.
   - Type A: Data Records refer to same Template.
   - Type B: Data Records refer to different Templates.
2. *Data Aggregation*: Data pre-processing within the transmission way to the gateway using aggregation functions.

The first aggregation type, referred to as *message aggregation* in the following, can easily be performed, because message transmission is dimensioned to 127 bytes whereas 102 bytes can be used for individual payload (e.g. TinyIPFIX messages) [18]. In the case of TinyIPFIX this mode can be subdivided into two sub-cases. In the first possibility - *Type A* - several Data Records, which refer to the same Template are transmitted in one message as shown in the right part of Figure 3.3 where two Data Records are transmitted in one packet. The second possibility - *Type B* - is the combination of two different Data Records in one message, which refer to different Template Sets. In this case the combined Data Set must refer to all needed Template Sets for decoding purposes. If the setting illustrated in Figure 4.6 is assumed and the transmitted messages between the aggregator nodes (node ID = 3,6,0) towards the sink are observed, the number of transmissions is reduced by one between each aggregator compared to a common setting without aggregation (cf. Figure 5.1). [102]

The second type of aggregation, called *data aggregation*, uses aggregation functions such as $f = max\{a, b\}$ or $f = avg\{a, b\}$ or $f = min\{a, b\}$. The chosen aggregation function depends on the application. For example, if only the maximum temperature in a room is interesting as shown in Figure 4.6 in the upper left room, the aggregation function $f = max\{a, b\}$ can be used. In this case the number of transmitted messages to the gateway node (node ID = 0) can be reduced by one in total. [102]

An advantage of aggregation is energy saving due to minimizing transmission amount as illustrated by the previously mentioned examples. But simultaneously a disadvantage occurs, as proven by Krishnamachar et al. in reference [106]. If aggregation is performed within the network, data is not directly forwarded to the next hop, which results in some transmission delay due to calculation time for aggregation result and energy consumption for processing. The delay depends on the aggregation mechanism, which must be performed. Another kind of delay occurs if requested values for calculation are delayed. In this case, the calculation process is interrupted and enforces a delay in communication towards the next hop. [102, 106]

4.2.2.1 Specification of TinyIPFIX-Aggregation

Due to application requirements and limited resources of components in wireless sensor networks, aggregation techniques are an attractive add-on to optimize the performance of a network and save resources at the same time. The TinyIPFIX-Aggregation framework offers the user two modes of aggregation with the previously described characteristics: (1) message aggregation and (2) data aggregation. Both techniques work with the TinyIPFIX message format as illustrated in a building scenario in Figure 4.6. [102]

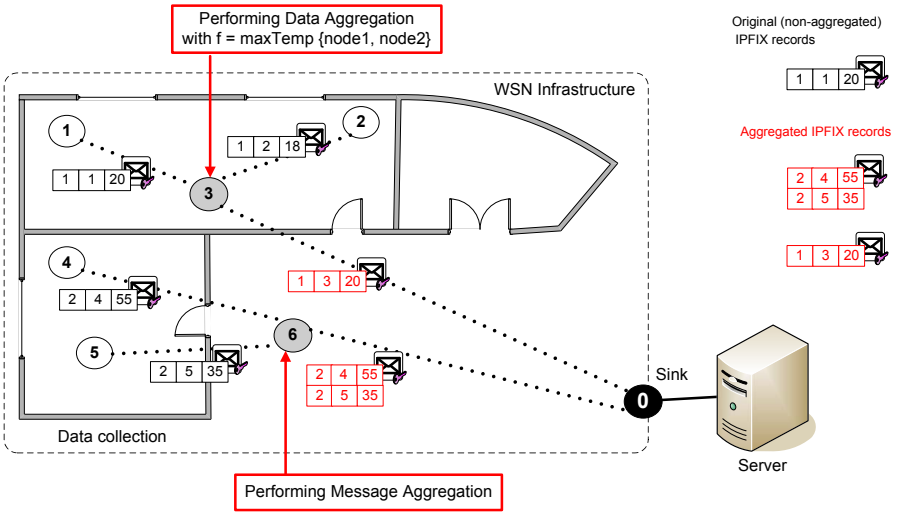


Figure 4.6: Simplified message flow in home scenario performing aggregation

The functionality of the protocol for message aggregation is shown in the lower room in Figure 4.6. TinyIPFIX messages, up to a certain amount, are aggregated into newly generated appropriate TinyIPFIX messages. The degree of aggregation is defined as the amount of TinyIPFIX messages aggregated in one TinyIPFIX message. The degree of aggregation depends on the aggregator's available memory as well as the number of sensor nodes in communication range of the aggregator, the acceptable message delay, and application constraints. In the case of message aggregation, information about the source of Template and Data Sets is essential for reconstructing data. Data is not allowed to be lost during the aggregation process. This requirement is addressed in message aggregation performed in *algorithm A*. The underlying decision and operation tree of algorithm A is summarized in Figure B.1 in the upper part and consists of the following steps [102]:

In the first step of **algorithm A**, the underlying TinyIPFIX protocol requires the announcement of related templates, which are buffered by the aggregator node to a maximum amount is equal to the degree of aggregation. In addition, incoming Data Sets are buffered before they are allocated to the corresponding Template Sets, which were stored during the first step. In the case of an unknown Template Set the Data Set is dropped, because the interpretation is impossible. [102]
If the maximum amount of buffered Template Sets is reached, the aggregator node announces the aggregated Template Set to the network as the third step of the algorithm. This announcement can happen before the aggregated Data Set is ready for transmission, which minimizes transmission delays. After the Template Set announcement the aggregated Data Set is prepared. [102]
In the following step updated Data Sets are allocated to their buffered Template

Sets and sent after the degree of aggregation has been reached. This step will be repeated periodically whenever aggregates are ready for transmission and the corresponding Template Set is known in the network. When receiving a new Template Set by the aggregator node, buffered Template Sets are updated and the procedure starts with step three again. [102]

In contrast, the upper room in Figure 4.6 shows the execution of data aggregation, which is also known in literature as data pre-processing. On the first look the resulted messages transmitted by the aggregators are shorter than in the case of message aggregation. The idea of this implementation is that the aggregator computes aggregates on the received sensor readings from the sensor nodes by applying aggregate functions on specific values, such as MIN, MAX or AVG. Because bidirectional communication is provided, aggregate functions and sensor reading types can additionally be selected and changed during operation. Selecting the aggregate function and value type via UDP-Shell commands does this. The underlying *algorithm B* for data aggregation consists of the following steps and is summarized in the lower part of Figure B.1 [102]:

In **algorithm B** data pre-processing aggregation is driven by the user request for specific sensor readings. The user can define if the aggregation function MAX or MIN or AVG should be performed as well as the degree of aggregation as marked with the bold blue dashed box in Figure B.1. As a consequence of pre-knowledge the recipient of the aggregated data already has the knowledge of the meaning of the expected data. Thus, the announcement of the modified Template by the aggregator node can be omitted. As in the previous algorithm, the aggregator buffers all incoming Template Sets from sensor nodes until the limit of the buffer or the degree of aggregation is reached. [102]
In the next step, incoming Data Sets are buffered. If enough Data Sets are buffered, the previous user defined aggregation function is selected and performed on the stored Data Sets. Before the computation can start the sensor's readings must be converted from the sensors' encoding of the measurement to an universal format. The aggregator, therefore, holds a lookup table for the data values based on the previously announced Template Sets. With the help of this table the aggregator can identify the desired value type of the measurement and computes the aggregate function on all buffered values of the same type. [102]
Finally, a newly generated Data Set for the demanded value type incorporating the aggregated values is generated and transmitted to the next receiver. Depending on the network structure the next receiver can be another aggregator node or a node with forwarding functionality without modification or the sink of the network. It might happen that additional information is added to the aggregated values in the aggregation Data Set. In this case it is essential that the receiver can decode the information properly. [102]

4.2.2.2 Specifiation of Neighbor Discovery Strategies

For completeness' sake it must be mentioned that depending on the application scenario of a wireless sensor network a neighbor discovery strategy is required before performing any specific algorithm (e.g. aggregation). In order to support the developed protocols in this doctoral thesis in different applications a neighbor discovery algorithm is included, which is based on common ideas from IP networks briefly described in the following.

Neighbor discovery is an important process for networks without a fixed link structure and is standardized in RFC 4861 [107]. Neighbor discovery protocols are re-

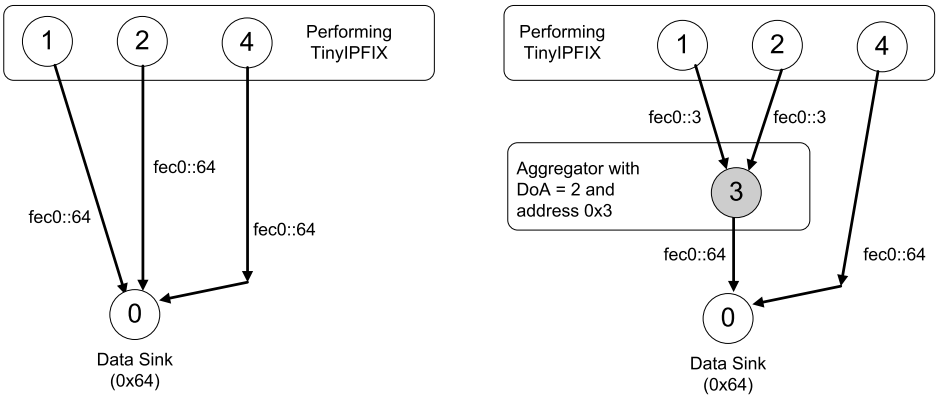
sponsible for link management within a network using IPv6 communication with multicast service. As described in Chapter 3 wireless sensor networks are integrated into the *Internet of Things*, and, therefore, require neighbor discovery in order to find a routing topology towards the gateway. Additionally, neighbor discovery is essential for a successful aggregation execution in order to process data within a network. Depending on the application scenario developers distinguish between two neighbor discovery techniques: (1) static/predefined neighborhood and (2) dynamic neighborhood. [102, 107]
In the first type the communication links are static and predefined, which means the position and location of each sensor node is known with respect to all others in a network. Those scenarios can be found in monitoring scenarios with fixed locations (e.g. home scenarios, structural / health / environmental monitoring). In this case, the context analysis is based on fixed pre-knowledge. Here the neighbor discovery is not very dynamic and updates happen only when nodes enter or leave the network. [102, 107]
If sensor nodes are deployed free, which means without a fixed network structure, a neighbor discovery phase is essential, which characterizes the second type of neighbor discovery techniques. Those situations can be found in emergency cases such as fire fighting or military scenarios, i.e. the sensor nodes are thrown out of a helicopter and must organize a working infrastructure on their own. The first performed operation after landing is the neighbor discovery phase in order to establish a working network to transmit collected data to a mobile sink. Although various neighbor discovery techniques exist, all of them have the following steps in common [107]:

1. The sensor node broadcasts a beacon with its individual information (e.g. ID, location, time).
2. Receiving nodes answer with their personal information, and add the transmitting node to their neighbor table.
3. This update of the neighbor table is done also by the sender.

In general, information of link quality is also added to the neighbor table in order to optimize routing, which implies a high dynamic in the network. Variations in link quality can cause the table entries to be updated, i.e. as nodes enter or leave the network. If nodes have less memory resources as characterized in Section 2.2, the neighbor table is limited to those neighbors with the best link qualities [108].

The TinyIPFIX-Aggregation framework introduced in Section 4.2.2.1 supports a combination of both neighbor discovery types, called (1) *static communication* and (2) *dynamic aggregation node discovery* [102].
The static communication is used if the wireless sensor network structure and communication links are known before deployment. In this case only minor updates are expected. The communications link between a sensor node and its aggregator is hard-coded consisting of the aggregator's IPv6 address and the UDP-Port number. Each aggregator listens to an individual UDP-Port. This setup is shown in Figure 4.7a. [102]
In the case of a more dynamic network, the TinyIPFIX-Aggregation framework supports an automatic aggregation node discovery procedure as illustrated in Figure 4.7b. This procedure allows an individual connection of sensor nodes to

a free aggregator in their range. The aggregator itself has a predefined degree of aggregation. For example, $DoA = 2$ means two sensor nodes can connect to this aggregator. [102]



(a) Type 1: Static/predefined

(b) Type 2: Dynamic

Figure 4.7: Neighbor discovery strategies performed by the aggregation framework

For the performance of the dynamic discovery procedure, the BLIP stack supports the 'Internet Control Message Protocol for the Internet Protocol Version 6' (ICMPv6) [109]. An aggregator broadcasts an ICMPv6 message to the network including a predefined message type code `200` and its IPv6 address. If an unconnected sensor node receives this message, it stores the aggregator's address. From this time on this sensor node addresses all its TinyIPFIX messages to this addressed aggregator. As a drawback of the first received message from an unconnected node the aggregator binds the node and reduces the degree of aggregation by one. The aggregator broadcasts this message type code `200` until it reaches its predefined degree of aggregation. In this case the aggregator changes its message type code to `201`, which indicates that the receiving sensor node cannot use him as next hop anymore. Now the sensor node must address its packets directly to the sink. If a connection to the aggregator is lost, for example due to node deletion, the aggregator sends out the message type code `200` again and the procedure rolls over again (repeats). [102]

Due to application changes, such as reduction or raise of the degree of aggregation, it is useful to run updates of the aggregators on the fly. The implemented UDP-Shell supports this function [102]. The degree of aggregation can be reduced down to one and raised back to its original value. If $DoA = 1$, the aggregation is a simple forward function as performed in networks without any aggregation functionality. In addition, the UDP-Shell allows to update the performed aggregation function. The TinyIPFIX-Aggregation framework supports the aggregation functions MAX, MIN, and AVG. More available functions are mentioned during the implementation description in Section 5.4. [102]

4.3 Security Considerations

As motivated in Section 3.3 security is important everywhere - especially in wireless sensor networks, because effective attacks are possible on all layers and the transmissions usually include sensitive data. In order to ensure more secure transmission, messages should be encrypted. Therefore, different mechanisms such as AES or RSA exist.

In this doctoral thesis a TinyDTLS solution for constrained devices was integrated into the existing wireless sensor network which was implemented within the scope of the master thesis by Thomas Kothmayr [81] in cooperation with CSIRO ICT Centre (Australia)[6]. The reminder of this section briefly characterizes different security mechanisms in Section 4.3.1 as inspiration for the developed DTLS solution. Finally, the developed TinyDTLS solution is introduced in Section 4.3.2.

4.3.1 Related Security Mechanisms

In this section different cryptographic methods and key management strategies are characterized. Those approaches are standardized and adapted to hardware constraints of wireless sensor networks. They influenced the decision to develop a DTLS solution for sensor devices, called *TinyDTLS*.

4.3.1.1 Cryptographic Methods

It is necessary to combine the mechanisms, such as nonce, timestamp, and index/counter, with cryptographic methods. Due to limited resources, used methods, as described in Section 3.3, are not practical for wireless sensor networks, because too many resources are required. In order to transfer those cryptographic methods to wireless sensor networks, they must be analysed with respect to code and data size, processing time, and power consumption considerations.
Cryptographic methods can be divided into public key cryptography and symmetric key cryptography. The public key algorithm RSA (Rivest, Shamir and Adleman) is common but is not useful for wireless sensor networks, because of computational overhead. For the same reason the Diffie-Hellman algorithm can be rejected. Carman et al. proved that it is more efficient to use symmetric key cryptographic algorithms or hash function, because they require fewer resources [110]. During an energy consumption analysis of RSA and AES on a MC68328 DragenBall processor it turned out that an AES with 128-bit block only consumes 0.104 mJ instead of 42 mJ when RSA with 1,024-bit block is performed. If low power techniques and sensor node constraints are kept in mind, it is possible to adapt common public key algorithms to wireless sensor networks. Mainstream researchers work with RSA and Elliptic Curve Cryptography (ECC). It is important that ECC can work with smaller keys than RSA and support an adequate level of security. [110, 81]

A well tested algorithm, called TinyPK, using RSA and ECC cryptography was developed by Watro et al. [111]. The algorithm was developed for the requirements of MICA2 nodes using TinyOS. The idea of this implementation was it to use a modified version of the Diffie-Hellmann protocol, which offers an authentication check of nodes in a wireless sensor network. Each node uses a static Diffie-Hellmann key pair with a text string processed via a certificate authority private key as a credential. With those keys and credentials a kind of handshake can take place between two parties to verify each other and to determine the session key for the communication. [111, 81]
Another approach of ECC on sensor nodes was implemented by An Liu and Peng Ning, called Tiny-ECC [112]. They used MICA2 nodes as well and based their approach on TinySec, which was developed by Karlof et al. [113]. It is a link layer security architecture offering two different security modes. The TinySec-AE mode offers an authentication encryption in which data payload is encrypted

[6]CSIRO ICT Center - Information and communication technologies `http://research.ict.csiro.au/`

and the packet is authenticated by a CBC-MAC. The TinySec-Auth mode authenticates the whole packet. For a comparison of both modes Karlof et al. assumed an application data amount of 24 bytes. The TinySec-Auth mode performed better than the TinySec-AE mode due to smaller overhead (44 bytes instead of 40 bytes), faster packet transmission in 26.7 ms instead of 28.3 ms, and required only 0.000165 mA per hour. Result showed that the TinySec-Auth mode compared to the TinySec-AE mode saved more resources for constraint hardware and offered more security at the same time. [112, 113, 81]

The above mentioned algorithms work with symmetric cryptography, which is not as expensive in operations as asymmetric cryptography. In general, asymmetric cryptography is often deferred to the gateway node (e.g. SPINS [114]). Those special nodes have more resources and, therefore, can easily perform asymmetric cryptography, provide authentication, and secure data aggregation.

4.3.1.2 Symmetric Key Management Solutions

Next to encryption algorithms key management is very important. Researchers, therefore, focus on this topic at the moment. They discuss different manners for key establishment and key distribution. One objective is the support of node addition and revocation. In general, key management must support key setup, key distribution, and key revocation mechanisms. [115, 116, 81]
Different key management strategies exist that are influenced by hardware resources, network structure, node deployment, and environmental factors of the application [115, 116]. The research groups Xiao et al., Zhang et al., and Wang et al. presented surveys covering key management schemes [117, 56, 9]. Those surveys have in common that existing key management protocols for wireless sensor networks can be divided into distributed, centralized, and probabilistic protocols. [81]

One representative of distributed key schemes is the *Peer Intermediaries for Key Establishment* (PIKE) protocol developed by Chan and Perrig [118]. In this class of key establishment protocols one or more sensor nodes are involved as a trusted intermediary to facilitate key establishment. PIKE is designed to address the lack of scalability of existing symmetric-key distribution schemes. All existing schemes incur linearly increasing cost in either communications per node or memory per node. PIKE achieves a trade-off by achieving overheads in both communications per node and memory per node. This is a highly desirable point in the design space of key distribution protocols. PlKE establishes keys between any two nodes regardless of network topology or node density. This makes it applicable to a wider range of deployment scenarios than random key predistribution, which requires a network deployment with high, uniform node density. Besides the key distribution center approaches that have a high communication overhead, PIKE is more resilient than previous approaches against sensor node compromise. PIKE enjoys a uniform communication pattern for key establishment, which is hard to disturb by an attacker. The distributed nature of PIKE also does not provide a single point of failure to attack, providing resilience against targeted attacks. In contrast to the currently popular random-key predistribution mechanisms, PIKE has the advantage that key establishment is not probabilistic, so that any two nodes are guaranteed to be able to establish a key. [118, 81]

In 2002 Eschenauer and Gligor proposed a probabilistic key predistribution scheme recently for pair-wise key establishment. This scheme is called *Basic random key management system* [119]. The main idea is it to let each sensor node randomly

pick a set of keys from a key pool before the deployment, so that any two sensor nodes have a certain probability to share at least one common key. Chan et al. further extended the idea of Eschenauer and Gligor and developed two key predistribution techniques, called *Q-composite random key predistribution scheme* and *Basic random-pairwise keys scheme* [120]. The q-composite key predistribution also uses a key pool, but requires two nodes compute a pairwise key from at least q-predistributed keys that they share. The random pairwise keys scheme randomly picks pairs of sensor nodes and assigns each pair a unique random key. [119, 120, 81]

4.3.1.3 End-to-End Security Solutions

Previously described key management systems in Section 4.3.1.2 use symmetric-key cryptography. The described networks require that the keys be distributed before deploying the sensor nodes. Those schemes do not scale well and often only offer link layer security instead of end-to-end security. Thus, security of the entire network is at risk if an attacker can compromise a few nodes. In a network, which supports end-to-end security, it does not matter if the underlying network infrastructure is only partially under the user's control as pointed out in reference [121]. An office scenario, which includes a common infrastructure for metering and climate-control purposes, is an example where end-to-end security is favored. Here the infrastructure is shared but the users are still able to keep their devices' data private from other members of the network. The provider of the infrastructure does not need to support security mechanisms if a protocol like DTLS is used, which is integrated between the transport and application layer in the used stack (cf. Figure 5.2). The required security is established between the communicating applications.

The protocol *IPsec*, standardized in RFC 4301 [60], is a network layer security protocol and is embedded in the IPv6 standard. Due to this it can be used when TCP or UDP is chosen as transport protocol. The IPsec protocol provides the following security mechanisms beyond a key exchange mechanism: Authentication and integrity is provided through the IPsec Authentication Header protocol [61]. In addition, confidentiality is provided through the included IPsec Encapsulated Security Payload protocol [62]. In the light weighted version of IPsec, developed by Raza et al. in 2010 [122], the protocol was adapted to the energy and memory constraints of sensor networks. Therefore, a compression technique for the IPsec headers was developed in order to reduce the network overhead. Currently, the existing implementation works with manually deployed keys instead of performing a key exchange as provided by the standardized IPsec protocol. [122, 81]

As mentioned in Section 3.1, the protocols and algorithms (TinyIPFIX, TinyIPFIX-Aggregation, neighbor discovery) developed in this thesis should be integrated in the application layer in order to be independent of the stack structure below, which allows stack exchange. This fact indicates why it cannot be assumed that the stack below the application layer supports end-to-end security. Sardouk et al. described in their article 'Data Aggregation in WSNS: A Survey' another more powerful reason to bring end-to-end security to wireless sensor networks [123]:

> *'To tackle the problem of data integrity and confidentiality in WSN, two methods are proposed: the hop-by-hop encryption and end-to-end encryption. However, both of them are based on encryption mechanisms to insure confidentiality and integrity, while trust-based security methods are recently used to insure a better data integrity.'* [123]

Furthermore, Sardouk et al. assumed support of aggregation functionality, which was located on the application layer, as it is done in this dissertation by TinyAggregation [123]:

> *'In hop-by-hop encryption, the aggregator nodes decrypt the received sensor nodes data to aggregate them. Thus, the aggregator nodes are vulnerable to attack. End-to-end encryption overcomes this vulnerability by keeping the sensor nodes data encrypted till arriving to the sink. Hence, it provides end-to-end privacy between each sensor node and the sink. Indeed, aggregate encryption data is not an easy issue. [...] In end-to-end encryption, the deployment of higher level of security is easier than hop-by-hop encryption. That is due to the ability of the sink to execute more complicated algorithms and store more data.'* [123]

Protocols addressing the previously mentioned issue, which offer application layer security, are Sizzle [124], SSNAIL [125], and Tiny-3-TLS [126]. Such protocols make assumptions of the underlying transport layer (e.g. reliability) and include key exchange mechanisms. Those key exchange mechanisms are based on asymmetric cryptography and ported to wireless sensor networks. [124, 125, 126, 81]
The protocol *Sizzle* is based on elliptic curve public key cryptography, and represents a secure web server stack, including HTTP and SSL [124]. The protocol supports server and client authentication handshakes as well as session resumption. For a successful handshake and data transfer phase a reliable connection is required; for example, TCP can be used here. The research group implemented Sizzle for TelosB platforms and compared needed resources for a SSL implementation using RSA handshake and ECC based handshake. As a result the RSA handshake was completed after six seconds and needed 850 bytes of RAM. In comparison, the ECC based handshake only needed one second and 650 bytes of RAM. In total both handshake protocols required 2.8 kB of static RAM for their whole application. [124, 81]
Nearly the same amount of handshake time is needed for the implementation of *SSNAIL* protocol [125]. A SSL implementation using ECC during the handshake phase is performed by SSNAIL. In contrast to Sizzle a light weighted TCP/IP stack is used for communication purposes. The SSNAIL protocol provides 80 bits of security through 160-bit ECC keys and 1,024-bit RSA keys. But RAM consumption is two times higher on average than the Sizzle implementation. [125, 81]
In 2006 Fouladgar et al. presented a trust delegation protocol for wireless sensor networks called *Tiny-3-TLS* [126]. This protocol is based on TLS and considers the authentication of a sensor node with a receiver (= remote terminal) outside a network. In general, a gateway is used as an intermediary, which removes the burden of asymmetric cryptography from the sensor nodes and supports them in the TLS handshake with the receiver. The established Tiny-3-TLS protocol assumes a common shared key between the gateway and the sensor node. If the gateway is fully trusted and a common key is established, only one symmetric decryption and encryption operation must be performed by the sensor node. In the case of a partially trusted gateway, a Diffie-Hellman key exchange protocol must additionally be performed. [126, 81]

4.3.2 Design Decisions for the TinyDTLS Solution

Depending on the results of the previously discussed approaches in Section 4.3.1 focusing on public key cryptography and the attack possibilities on wireless sensor

network as presented in Section 3.3, a standard-based approach across all communication layers scales best for heterogeneous networks. Furthermore, by taking advantage of the capabilities of a new embedded platform, a stronger two-way authentication handshake is adopted instead of the one-way authentication performed in Sizzle [124]. The chosen implementation must also support network updates which can happen if a node fails or enters the network.

The TinyDTLS solution presented in this doctoral thesis is a security architecture implementation based on standards across the whole network stack. It is assumed that the internet is connected by IPv6 and parts of the sensor nodes also support IPv6 communication by running an implementation of 6LoWPAN such as BLIP (cf. Figure 3.2). The implementation provides end-to-end security for sensor networks connected to the Internet and supports UDP on the transport layer. IEEE 802.15.4 is used for the physical and MAC layer. [81, 121, 127]
As mentioned before it is planned to add new features to the established stack by adding it on the application layer, which is also the case with the Datagram TLS (DTLS) implementation presented in this thesis. Such an implementation on the application layer has advantages and disadvantages at the same time. Easy modifications of the security implementation are possible and can be deployed to the devices directly with other applications at the same time. [81, 121, 127]
As a drawback this implementation method makes some assumptions on lower layers. Those assumptions can be reliability and packet size during transmission. In order to solve the reliability problem, DTLS was chosen, because it implements its own reliability mechanism during the handshake phase. DTLS itself is a modification of TLS for the unreliable UDP and inherits its security properties [23]. The DTLS implementation supports authentication, integrity, and confidentiality that are just a part of general security goals previously presented in Section 3.3. [81, 121, 127]

In the developed TinyDTLS implementation the focus lies on end-to-end communication security applications and relies on other schemes for low communication layer security. A typical scenario could be a shared office building where each party subscribes to only a part of the sensor readings. Those parties wish to keep the data they subscribe private from other parties. In order to reduce network costs, they share a common communication network. This simple and intuitive example shows the request for proper authentication of data publishing devices and access control throughout a network. These problems can be solved by adding an *Access Control Server* (ACS) into the established architecture, which is a trusted entity. Compared to the commonly used sensor hardware the server hardware has much more resources, where for example the access rights for the publishers of the secured network can be stored. In order to fulfill this requirement unique identities for each publisher are needed. In the Internet this requirement can be solved with the help of public key cryptography, where identifiers provide X.509 certificates. Those certificates contain the public key of an entity and its common name together with other information. The certificate is signed by a trusted third party - called *Certificate Authority* (CA). As a result the signature allows the receiver to detect modifications to the certificate and the certificate authority verifies the identity of the entity that requested the certificate. The administrator of the network can either run such a certificate authority or one of the established Internets certificate authorities can be used. [81, 121, 127]

Today notebooks offer the opportunity to use an integrated *Trusted Platform Module* (TPM) for the above described purposes. Such a TPM is a cryptographically chip.

This chip supports RSA key generation, built-in RSA encryption and decryption, and secure key storage. The aim of the development of such a chip in cooperation with the software on the notebook was it to increase the reliability and trustworthiness of a system. The main work is done by a TPM microcontroller. Together with RSA functionality a storage root key can be stored, which depends on the machine configuration (cf. Section 3.3.3). This machine configuration is built upon the hardware and the software of the machine during booting. This key is the root for all required derived keys. If it is lost or damaged, data cannot be restored. A more detailed description of the Trusted Computing strategy can be found in the book 'A Practical Guide to Trusted Computing' written by David Challener et al. [92].

The research institute CSIRO in Australia showed that RSA could be used in sensor networks. Therefore, they developed the sensor platform *OPAL* which has a Trusted Platform Module embedded [128, 33]. Currently, the certificate of a Trusted Platform Module equipped publisher and the certificate of a trusted certificate authority must be stored in the publisher prior to development, which means that an update during boot phase is not possible as suggested in common notebook solutions. [128, 33, 81]
In general, a wireless sensor network also consists of sensor nodes without such a special chip; therefore authentication via the DTLS pre-shared key cipher-suite will be supported in the future. In this case a small number of random bytes are chosen which are preloaded to the publisher before deployment in order to derive the actual key. This newly established secret must also be available for the access control server, which will disclose the key to devices with sufficient authorization. The described system architecture is summarized in Figure 4.8. [81, 121, 127]

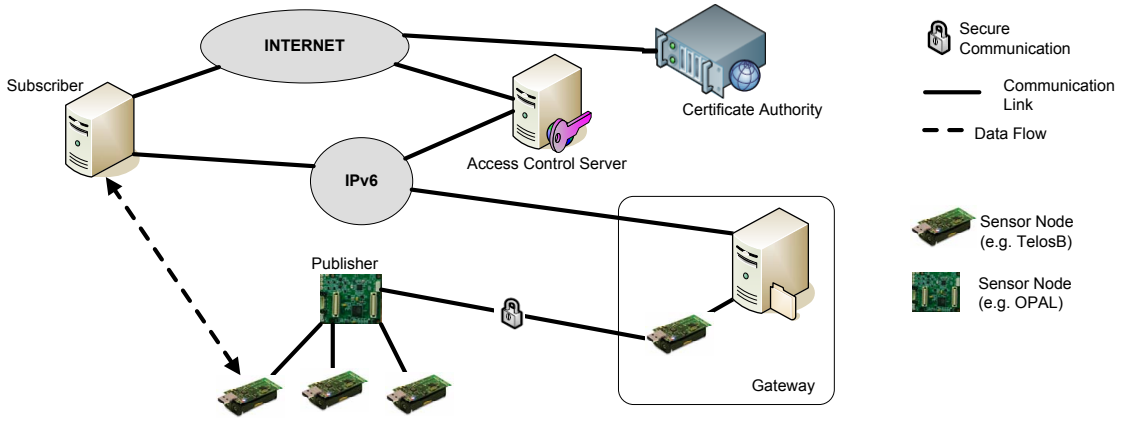


Figure 4.8: Overview of DTLS system architecture

4.3.2.1 Specification of Secure Communication Channel Establishment

For better understanding the setup shown in Figure 4.8 is used as the established network. It is assumed that a data publisher is an OPAL node or a TelosB node. If a data publisher wants to join the network, the node needs to establish a secure connection to a pre-configured subscriber. As mentioned in Section 4.3.2 the nodes in the wireless sensor network can have different capabilities and, therefore, must chose the appropriate DTLS cipher suite. [81, 121]
If the publisher is a node with an embedded TPM chip, it can perform a fully authenticated handshake with the subscriber. In this case, the subscriber acts as

the DTLS server during this handshake. The publisher and the subscriber transmit their RSA certificates in X.509 format. As mentioned in Section 4.3.2 those certificates have been signed by a trusted certificate authority with its private key. This guarantees that the certificate has not been modified. Publisher and subscriber can be sure of each other's identity if both keep their respective RSA private keys secret. [81, 121, 127]

In the presented architecture sensor nodes are the most vulnerable entities in the network. If an attacker can gain access to such a sensor node - called physical tampering - he can also gain access to the stored information, such as the private key. As a result he can compromise the sensor node and receive full access to the network. In order to prevent this, the RSA private key must be stored in a tamper-proof TPM chip and never be passed on outside. As a consequence all operations using this RSA key must be performed in this chip. In addition, the attacker has a lower chance to perform physical tampering and the authentication during the handshake phase is guaranteed. During the handshake the publisher checks the identity information of the subscriber's certificate against the pre-configured identity. The subscriber has the option to do the same with the access control server in order to verify the access rights of the publisher. [81, 121, 127]

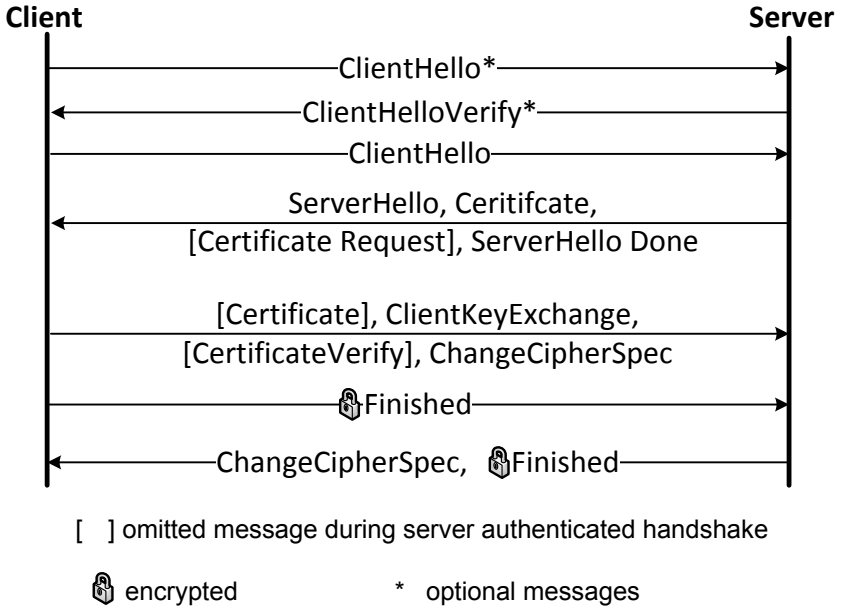


Figure 4.9: A fully authenticated DTLS handshake

Figure 4.9 illustrates the message flow of a fully authenticated DTLS handshake. Individual messages are grouped according to their direction and occurrence sequence. The first two steps are optional messages in order to protect the server against Denial-of-Service (DoS) attacks (marked with *). The client has to prove that he can receive and send data by resending its `ClientHello` message in step 3 with the cookie sent in the `ClientHelloVerify` message by the server. The `ClientHello` message contains the protocol version supported by the client as well as the cipher suites that it supports. In the next step the server answers with its `ServerHello` message that contains the cipher suite chosen from the list offered by the client. The server also sends a X.509 certificate to authenticate itself followed by a `CertificateRequest` message if the server expects the client to au-

thenticate himself. The `ServerHelloDone` message only indicates the end of step 4. If requested and supported, the client sends its own certificate message at the beginning of step 5. The `ClientKeyExchange` message contains half of the pre-master secret encrypted with the server's public RSA key from the server's certificate. The other half of the pre-master secret was transmitted unprotected in the `ServerHello` message. The keying material is subsequently derived from the pre-master secret. Since half of the pre-master secret is encrypted with the server's public key, it can only complete the handshake if it is in possession of the private key matching the public key in the server certificate. Accordingly, in the `CertificateVerify` message the client authenticates himself by proving that he is in possession of the private key matching the client's public key. He does so by signing a hashed digest of all previous handshake messages with its private key. The server can verify this through the public key of the client. The `ChangeCipherSpec` message indicates that all following messages by the client will be encrypted with the negotiated cipher suite and keying material. In step 6 the `Finished` message contains an encrypted message digest of all previous handshake messages to ensure that both parties are indeed operating based on the same unaltered handshake data. Finally, the server answers with its own `ChangeCiperSpec` and `Finished` message to complete the handshake. [81, 121, 127]

If the sensor node is a platform without a TPM chip, it has to perform a variation of the TLS pre-shared key cipher suite [81]. In this case, pre-installed random bytes are used as *protokeys*. Those bytes are used to derive a pre-shared key for a session. The publisher sends its identity during the `ClientKeyExchange` message of the handshake instead of a certificate. In addition, the publisher adds some randomly generated bytes to its pre-shared key identity in order to form a session identity. In the next step the publisher derives the pre-shared key by applying a hash-based message authentication code (HMAC) function to the session identity with the *protokey* as key. The subscriber authenticates with the access control server and requests the pre-shared key for the publisher's session identity. In this scenario the access control server is the only fully trusted entity allowing safe storage of the protokey. Based on the session identity and the protokey the access control server generates the pre-shared key for the subscriber. This key is transmitted to the subscriber via the established (D)TLS secured connection. Here a chain of trust is assumed. [81, 121, 127]
In comparison to the previously presented approach with a TPM including publisher the publisher in this case places less trust in the subscriber. The publisher requests an authentication of the subscriber by the access control server together with the generation of a session key. Due to the embedding of a third party (the access control server) the authentication in this scenario is weaker than in the TPM scenario. [81, 121, 127]

If the DTLS handshake is independent of the chosen authentication method, it is possible to establish an IP communication for upcoming data transfer between subscriber and publisher (cf. Figure 4.10) [81]. It is now assumed that a subscriber wants to establish a communication with a special publisher. [81, 121, 127]
First, the subscriber requests an access ticket from the access control server. Therefore, the subscriber must verify the requested publisher and the communication type. Communication types can be `read`, `write` or `read/write`. If the access right check performed by the access control server is positive, it sends the correlated request to the publisher. This request is an information tuple consisting of peer address, connection type, and connection timeout. An attack is unsuccessful,

because the connection between publisher and access control server is encrypted. Now it depends on the publisher whether the access is allowed or not. If the publisher accepts the request, a DTLS handshake is initialized with the subscriber to establish a secure connection, followed by the data transfer between subscriber and publisher. [81, 121, 127]

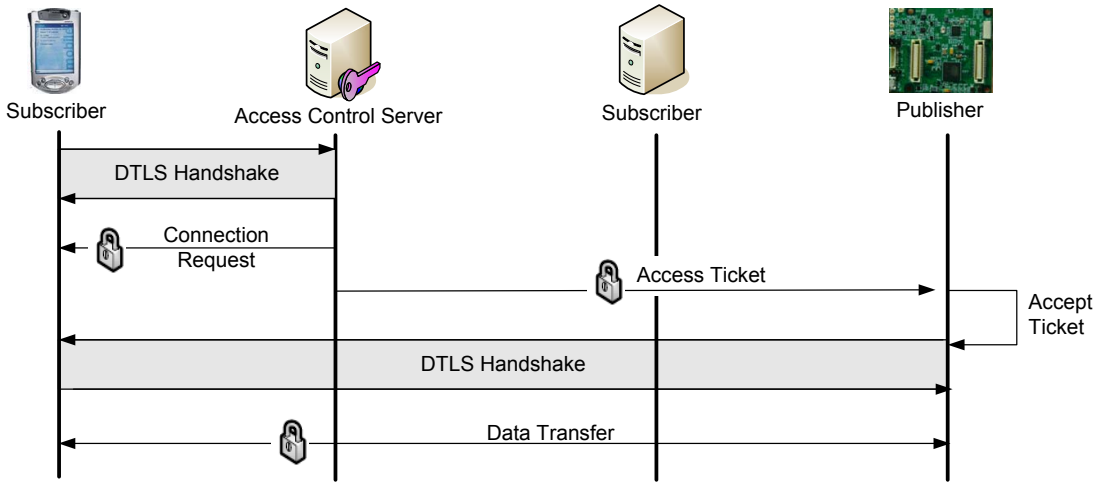


Figure 4.10: Connection establishment for data transfer

4.4 Graphical User Interface

Today the call for graphical user interfaces (GUIs) occurs everywhere. Users want to work by clicking buttons without typing long commands into terminals. On the one hand the graphical user interface should be user friendly and intuitive usable. On the other hand, it should support different vendors and user requirements. In general, those challenges cannot be solved with one graphical user interface for all possible applications. Thus, many different and individual solutions exist, which are developed for one special application (e.g. simulation, visualization [129]). The design decisions can be influenced by available solutions mapping other requirements.

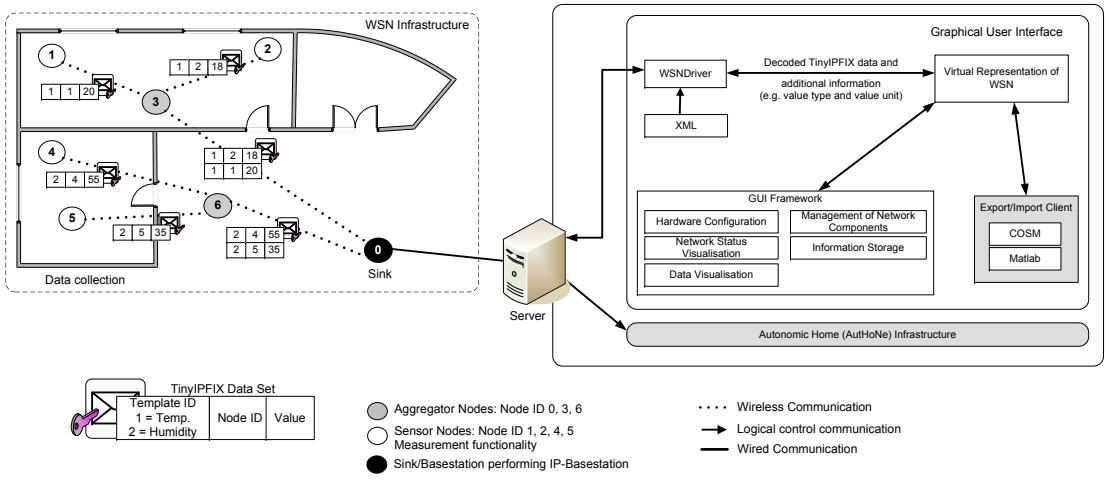


Figure 4.11: Overview of GUI architecture

4.4.1 Related Work

Currently graphical user interfaces for wireless sensor networks are mostly available for simulation tools. Those simulation tools are restricted to special node hardware and operating systems. For example, the simulator AVRORA just supports TinyOS 1.x and the platforms MICA and MICA2 [130]. If someone wants to test the programs on higher versions of TinyOS, it is not possible and the code must be re-programmed following the requirements of TinyOS 1.x. The same problem occurs if the program was developed for other platforms except MICA and MICA2.
Another simulation tool called TosGUI was developed 2002 with the goal to simulate different network topologies in order to analyse the performance of the planned network [131]. This tool also requires the program code to follow TinyOS 1.x requirements. Both simulation tools are open source and based on Java. Another open source tool is TOSSIM [132]. All existing simulation tools have the disadvantage in common that a whole network is configured and the user cannot make updates during runtime.

4.4.2 Design Decisions and Specifications for the GUI

The graphical user interface developed during this doctoral thesis was implemented in Java within the scope of the bachelor thesis by Andre Freitag [133]. It allows the user to setup the network during runtime. The user can program the sensor nodes as needed by following an application specific menu where hardware can be specified in detail and which code should be performed. The user can also define the ID of the node, which should be unique for each node. Additionally, the user can verify the communication channel in order to allow the activation of independent wireless sensor networks at the same time. [133]
The established graphical user interface was especially developed for the performed solution for TinyIPFIX and the optional extensions introduced in Section 4. The supported operation system of the applications is TinyOS 2.1.1 and currently all available platforms working with this operating system are supported. The structure of the implemented graphical user interface can be subdivided into the following tasks (cf. Figure 4.11) [133]:

1. Configuration of the programming code for individual platforms.
2. Displaying the current running status of the wireless sensor network.
3. Exporting data to different analysis tools.
4. Importing analysis and visualization results.

As indicated in Figure 4.11 the graphical user interface is integrated on the server side. Data received by the sink is transmitted via a wired connection to the server. In comparison to Figure 5.7 the received data is now transmitted to the graphical user interface instead of the AutHoNe infrastructure. In this case, the logical control communication link is a bidirectional link in order to allow communication to the GUI-components as well as from the GUI to the components in the wireless sensor network. The latter is used if the sensor nodes are programmed with updates. All logical control communication links within the graphical user interface are bidirectional in order to ensure information exchange between all components. [133]

When data is received, the data is forwarded to the so called *WSNDriver* [133]. The WSNDriver performs decoding procedures in the following two directions using an XML file as input [133]:

1. Decoding of TinyIPFIX packets received from the wireless sensor network into virtual representation data format for graphical user interface.
2. The other way round.

If TinyIPFIX packets are received from the wireless sensor network, the *WSNDriver* translates those received TinyIPFIX packets. The used XML file includes information about the hardware and equations in order to calculate the measured values into formats (e.g. hexadecimal into decimal) required for upcoming tasks (e.g. visualization). From this point on the whole data is available in a virtual representation of the wireless sensor network within the graphical user interface. [133]
In the performed decoding step the TinyIPFIX data packets are decoded according to the corresponding templates and additional information is added to each entry, such as value type and value unit. If the received TinyIPFIX packet includes an aggregated data packet, it is split into its components from this stage on. If the virtual representation was built, the included GUI Framework and the Export/Import Client within the graphical user interface only work with this virtual representation of the data instead of the original data. [133]

The virtual representation of the wireless sensor network is implemented in the class *WSN*, which is subdivided as illustrated in Figure D.2 representing an UML illustration for an exemplary sensor network analog to reference [133]. The class *WSN* includes information about nodes and the topology of the wireless sensor network. The class *WSN* is connected to the two classes *WSNTopology* and *WSNNodes* offering information about the links within the network (respectively about the data transmitted by the nodes). Those two classes are connected to the class *WSNNodeTopology::Link* which offers information about source and target. The class *WSNNodes* has an additional connection to the class *WSNNode::Datum* including information about the type, value, and unit represented by the sensor node. Figure D.2 shows an example. [133]

This virtual representation is picked up by the layout of the GUI framework, which supports the previously mentioned tasks discussed in detail in Sections 5.4.1 to 5.4.3. The underlying framework of the graphical user interface can be subdivided into two parts: (1) common interface and (2) modules including drivers and data handlers. [133]

The common interface is represented by the class *WSN*. The interface offers the requested information of the underlying wireless sensor network. This support also includes data import of the network towards the modules as well as adding or deleting of modules. The latter includes the required drivers and data handlers, which both are called *WSNModules*. The class *WSN* has only information about nodes and topology. Everything else must be handled by the modules, which include more information about the wireless sensor network architecture. For example, the class *WSNDriver* is responsible for the information extraction from the network components in order to submit the information to the model. This support is only possible, because it has detailed information about the real topology and supports required functions (e.g. manipulation, node shut down). The interactions between the different components are illustrated in Figure 4.12. [133]

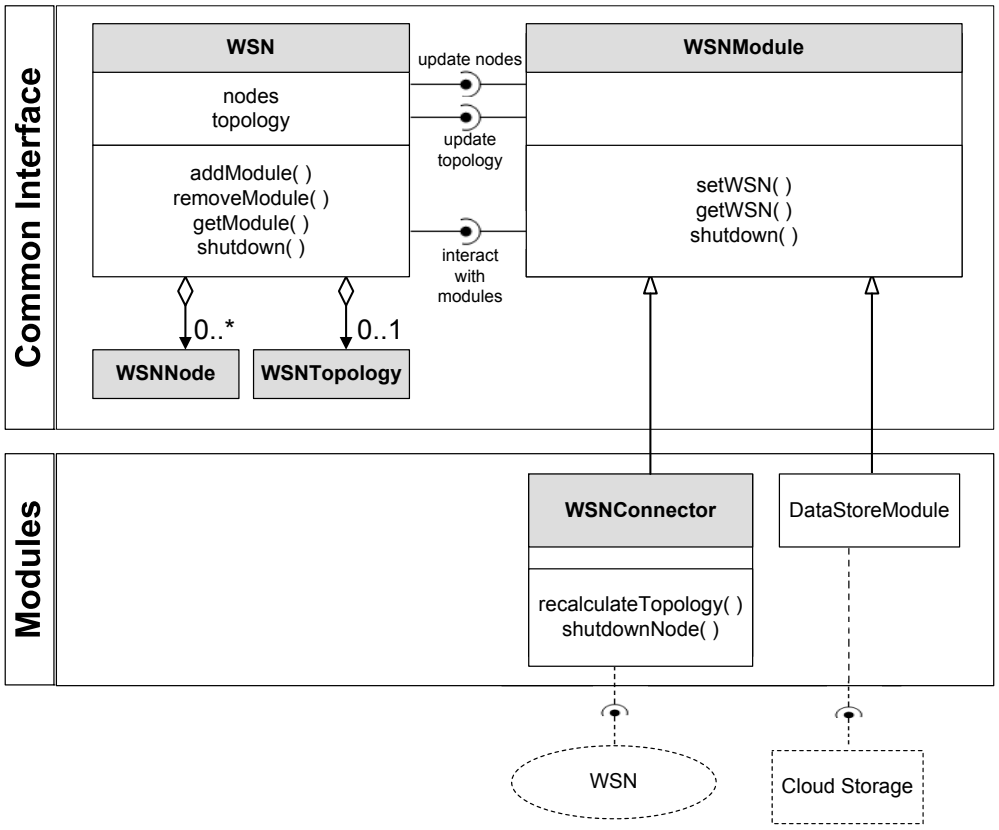


Figure 4.12: UML draft of framework structure

The framework offers different events in order to support manifold functionalities as described in the upcoming sections. The events include calls for `update nodes`, `update topology`, or `interact with modules`. If nodes enter or leave the wireless sensor network, the class *WSN* recognizes this first. It decides for which module this information is interesting and shares it via a forwarding procedure. Due to the virtual representation of the real world wireless sensor network the update is independent of the underlying architecture. In order to extend the functionality of the classes *WSN* or *WSNModule* new events are defined and added. A more detailed description is given in Section 5.4. [133]

4.5 Summary and Findings

This chapter described design decisions for the developed secure data transmission solution in this dissertation keeping the constraints of wireless sensor networks and related protocols from IP networks in mind. The first two Sections 4.1 and 4.2 focused on data transmission in sensor networks. Here related protocols from sensor networks were introduced. Those protocols together with the collected information in Section 3 inspired the finally realized protocols in this dissertation. First of all, the efficient data transmission protocol TinyIPFIX was developed based on the standard IPFIX together with compression techniques in order to reduce the occurring overhead by IPFIX message and Set headers. It is followed by an extension, called TinyIPFIX-Aggregation, in order to reduce the traffic within the network by preprocessing data on selected points within the wireless sensor network.
Due to the connection between data and sensible information as well as the integration of wireless sensor networks to the *Internet of Things*, a secure data transmission

had to be established. Therefore, this chapter in Section 4.3.1.3 pointed out what security options already exist in wireless sensor networks and why end-to-end security became important. End-to-end security was already supported by IPsec included in the 6LoWPAN stack (respectively by BLIP as a derivate of 6LoWPAN) as a network security feature [68]. In order to support stack exchange below the application layer, as motivated in Section 3.1, and 'to tackle the problem of data integrity and confidentiality' in wireless sensor networks [123], it was essential to support end-to-end security on the application layer. Finally, a developed end-to-end security solution, called TinyDTLS, was presented including a sensor node with special hardware (TPM chip).
Last but not least, a graphical user interface was established and introduced in order to optimize user's comfort. The graphical user interface uses a virtualization of the deployed wireless sensor network in order to be flexibly adaptable to other settings. It offers the opportunity to configure hardware, to manage network components, to visualize network status and data, and to store collected information.

Regarding the mentioned research questions on efficiency the following contribution has been made in this chapter:

(E1) Is the IP Flow Information Export (IPFIX) protocol a viable solution for transmission of sensor data in wireless sensor networks?

> In Section 3.4 this research question was answered with 'Yes'. It was mentioned that a drawback is the additional overhead of 20 bytes caused by IPFIX headers (IPFIX message and Set header) that reduce free space in the payload of each message. In section 4.1.3 header compression techniques were introduced in order to solve the drawback. Three different compression techniques were developed that had a pre-header in common: defensive compression, modified defensive compression, and aggressive compression. This pre-header specifies the field sizes of the IPFIX message and Set header. The pre-header in the aggressive compression technique has a minimum size of three bytes which means a compression of 85% (cf. Figure 5.4).

(E2) Is it possible to combine data pre-processing techniques (e.g. aggregation) with the IPFIX protocol within the network?

> Yes. Depending on the chosen application scenario for the deployed wireless sensor network it might be interesting to pre-process data within the network itself. This work incorporates the reduction of network traffic throughout the whole network. Section 4.2 introduced existing aggregation techniques in wireless sensor networks such as TAG, AIDA and SIA. Those protocols are very specified and were developed for special applications. In the case of IPFIX a general pre-processing technique was chosen that only aggregates data. Therefore, the TinyIPFIX-Aggregation framework was developed offering message and data aggregation. For this extension support new templates had to be specified for IPFIX and a neighbor discover algorithm had to be integrated. Additionally, a user can manually log on the aggregator node in order to modify the performed aggregation within the network (e.g. degree of aggregation, performed aggregation function from AVG to MAX).

Regarding the mentioned research questions on security the following contribution has been made in this chapter:

(S1) Is it possible to secure data transmission in wireless sensor networks with known standards from IP networks?

In Section 3.4 this research question was answered with 'Yes'. It was mentioned that restrictions concerning resource consumption existed. Section 4.3 introduced security considerations and a brief overview was given of existing security protocols for wireless sensor networks. First, different cryptographic methods (e.g. RSA, AES, TinyPK, TinySec, Tiny-ECC) were characterized, followed by symmetric key management solutions (e.g. PIKE, solutions by Echenauer and Gligor), and end-to-end security solutions (e.g. IPsec, Sizzle, SSNAIL, Tiny-3-TLS). All introduced protocols require different resources of the sensor nodes, and, therefore, nodes can be exhausted quickly. Due to the comparison of the different approaches, it was pointed out that a standard-based approach across all communication layers scaled best for heterogeneous networks. Depending on the technology development of embedded platforms (e.g. OPAL including TPM chip) more security functions can be supported that allow authentication of the participating parties.

(S2) Can DTLS be performed on strongly constrained hardware as used in wireless sensor networks?

Yes. DTLS can be performed on constrained hardware. Section 4.3.2 described the required modifications for a DTLS transfer on wireless sensor networks. In this dissertation a new platform, called OPAL, was integrated in the wireless sensor network. This platform includes a Trusted Platform Module that allowed it to work with certificates. Therewith, a strong two-way authentication handshake can be adopted to the communication participants in the wireless sensor network. The included TPM chip on the platform allowed it to save the RSA private key to be stored in a tamper-proof location and prohibit to pass it outside. As a consequence all operations using this RSA key had to be performed in this chip. All these characteristics affect the attacker's work. The authentication, which uses certificates during the handshake phase in a standard DTLS handshake (cf. Section 3.3.2), offers the same properties as authentication performed via the conventional TLS protocol in the Internet. In the case of a platform without a TPM chip, it was shown that a weaker authentication can be supported by using a variation of the TLS pre-shared key cipher suite. Here the publisher places less trust in the subscriber and required an authentication of the subscriber by the access control server together with the generation of a session key.

5. Implementation

As shown in Section 2.5 the application area of wireless sensor networks is manifold but the main tasks - collecting data and transporting it to a sink - are the same; the intermediate operations and protocols can vary. A home monitoring scenario was chosen to validate the TinyIPFIX protocol and its extensions. The implemented protocols - TinyIPFIX and its extentions - are flexible in order to support different applications and hardware vendors at the same time, as mentioned in Section 4.1.2. They currently require the operating system TinyOS 2.x. Protocols and functionalities (e.g. UDP-Shell, certificate creation) can be ported to another operating system (e.g. Contiki) with little changes as well as used with other hardware (cf. Section 2.4.3).

Throughout the wireless sensor network experiments in this dissertation the operating system TinyOS 2.1.1 with BLIP support is used as the operating system of choice together with Berkeley Motes IRIS and TelosB, which support IEEE 802.15.4/ZigBee and work on the 2.4 GHz band [31, 32].
For the IRIS platform two different sensor boards are available. The MTS400 has either a temperature and humidity sensor combined or a barometric pressure combined with a temperature sensor on board as well as a light sensor and voltage. GPS is optional and included in MTS420. The MTS300 includes sensors for light, temperature, and acoustic together with an acoustic actuator. Due to the physical positioning of the sensors on the board MTS300, it is not allowed to activate the temperature and light sensor at the same time, because the activation can damage both sensors. If an application uses the MTS300, either the 3-tuple temperature sensor, acoustic sensor and acoustic actuator are activated or the 3-tuple light sensor, acoustic sensor and acoustic actuator. This fact must be taken into account when deploying the wireless sensor network. For technical details it is referred to the 'MTS/MDA Sensor Board Users Manual' from Crossbow Inc. [31].
The TelosB node produced by the company Advantic is a MTM-CM5000-MSP sensor node with an external antenna including on board sensors for temperature, voltage, and humidity [32]. The TelosB platform is also offered by Crossbow Inc., but not used here to prove the protocol's support in a wireless sensor network consisting of different vendors' hardware. In addition, TelosB nodes produced by Advantic include an antenna with a better radio range compared to TelosB node produced

by Crossbow Inc., which doubles the radio range to 300 m outdoors and 40-50 m indoors [32, 31].

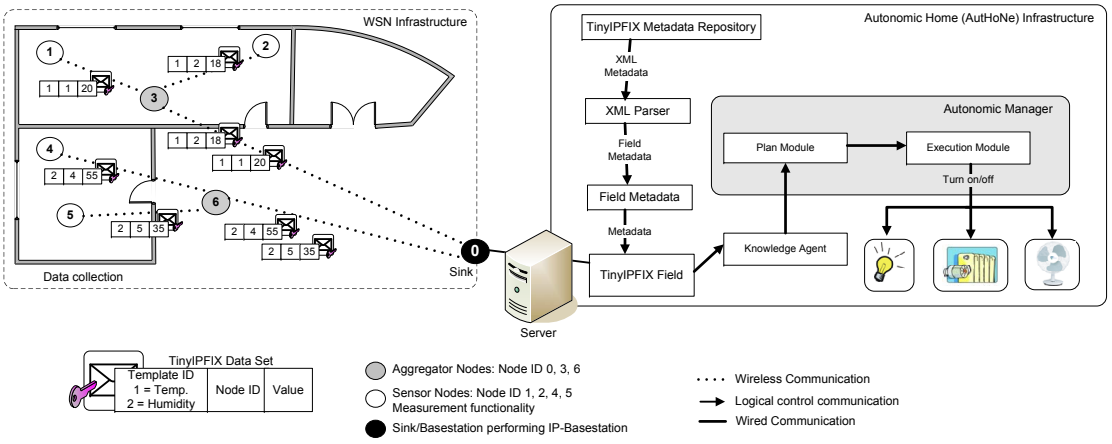


Figure 5.1: AutHoNe setup: simplified TinyIPFIX message structure

The assumed basic scenario for the implementation description is illustrated in Figure 5.1. The left part of the figure shows a building scenario, as assumed in the AutHoNe project, consisting of three rooms where sensor nodes are deployed. Here sensor nodes (marked white) are only data collectors and aggregator nodes (marked grey) forward received individual messages without modification towards the sink (node ID 0). The functionality of intermediary nodes (marked grey) can differ depending on their performed protocols (e.g. message/data aggregation). After collected data is received at the sink, the data is forwarded to the server. The server includes the required infrastructure in order to translate the received data by using XML-based meta data. The translated data is further forwarded to applications, such as the Knowledge Agents as part of the AutHoNe infrastructure, in order to make the collected sensor data available for management units (e.g. Autonomic Manager to coordinate functionality of lightning or heating control). The above described wireless sensor network consists of various hardware from different vendors in order to proof the flexibility of the developed protocols independent of the application scenario in this dissertation. Therefore, protocols used must allow integration of new vendors or setups (e.g. other sensor tuples) with a minimum of manual configuration. The established wireless sensor network uses IPv6 for communication purposes, because it was decided that BLIP is the IP communication support of choice for the experiments in this dissertation, which is included in the operation system TinyOS 2.1.1. as an existing implementation [70]. Today, in the *Internet of Things* it is assumed that networks use IPv6 instead of IPv4, where different arguments exist for this decision such as extended address space [63, 1]. If IPv4 is required on the server side, a parser must be integrated on the server for translation purposes. In the presented wireless sensor network experiments UDP is chosen as the transport protocol of choice, because todays' available wireless sensor network deployments prefer to use UDP.

The final network stack developed during this doctoral thesis is shown in Figure 5.2. Marked in bold in the application layer are the newly developed protocols and supported functionalities (e.g. UDP-Shell required for TinyIPFIX-Aggregation) by the wireless sensor network in the experiments. Due to previously described re-

quirements and limited sensor node hardware (cf. Section 2.2), protocols with little overhead are needed.

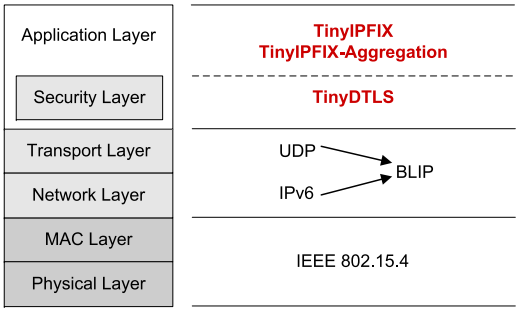


Figure 5.2: Structure of established network stack

The upcoming Sections 5.1 - 5.4 document the implementation of the TinyIPFIX protocol and the new features introduced to the application layer. These features are TinyIPFIX-Aggregation and DTLS support on the sensor node plus export/import support and a graphical user interface (GUI) for configuration purposes on the user side.

5.1 Implementation of the TinyIPFIX Protocol

The developed TinyIPFIX protocol fulfills the requirements for efficient data transmission of sensor data in wireless sensor networks and allows a high level of flexibility for hardware changes (e.g. using TelosB instead of IRIS) and application requirements as introduced in Section 3.2 and Section 4.1. TinyIPFIX is located in the application layer. Therefore, it does not require changes in the protocol specifications if the underlying stack is exchanged (e.g. use 6LoWPAN implementation of Harvan and Schönwälder [67] instead of BLIP [71]). However, when adapting IPFIX to wireless sensor networks their particularities have to be taken into account. The next section thus briefly describes the message format used by the operating system TinyOS, followed by the resulting packet structure with TinyIPFIX messages as the individual payload and the corresponding wiring of the implemented protocol components under TinyOS.

5.1.1 TinyOS and TinyIPFIX Message Format

In general, wireless sensor networks with the presented hardware in Section 2.2 use TinyOS as operating system. The main advantages of TinyOS are the modular structure and the ability to work over the communication standard IEEE 802.15.4 (cf. Section 2.4.1). A maximum transmission unit of 127 bytes is supported by the RF transceiver CC2420 an IEEE 802.15.4 compliant radio transceiver [18]. Figure 5.3 shows the default message structure under TinyOS as described in reference [25]. Grey marked fields are generic 'Active Message fields' defined in the TinyOS library `tos/types/AM.h` and the payload marked with red dashed lines is defined by the application (e.g. TinyIPFIX) [25].

The TinyOS packet starts with a one byte sized `Length` field specifying the message length in total. The following two bytes long `Frame Control Field` (FCF) results from the communication standard IEEE 802.15.4. The next field is the `Frame Control Field` (FCF) for ordering purposes with one byte dimension. The `Destination PAN` identifier (DestPAN) specifies in two bytes the address of the

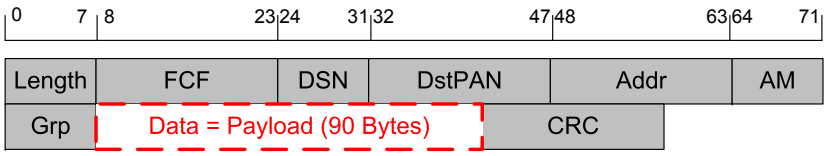


Figure 5.3: Packet structure under TinyOS [bits]

`Personal Area Network` (PAN). In the next two bytes field `Addr` the address of the destination node is specified. The chosen active message type is specified in the one byte `AM` field. The `AM` information can be compared to a UDP port, which the communication partners agreed upon. The `Grp` field is a user defined group identifier, which can be optionally used in order to specify multiplexing techniques (e.g. partitioning). The second to last field is variable in size and includes the individual payload (marked red dashed). This field has a range up to 102 bytes and on default 28 bytes. The maximum transmission unit depicts the maximum size. The TinyOS packet is concluded with the `Cyclic Redundancy Check` (CRC) field. [25]

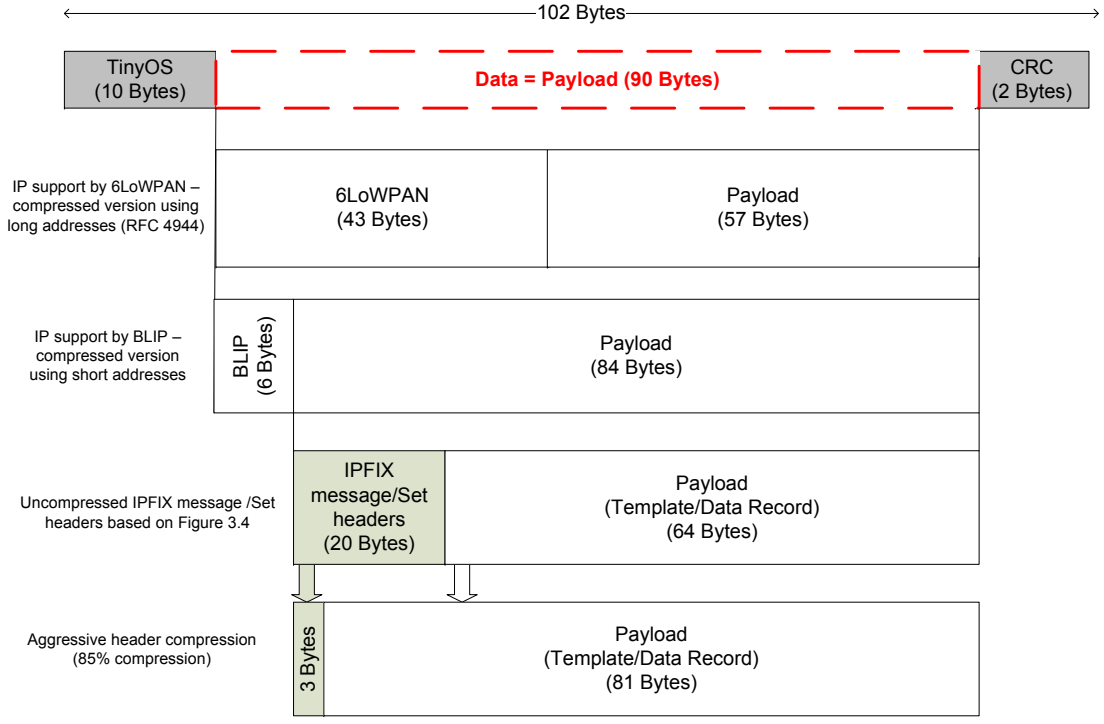


Figure 5.4: Exemplary TinyOS message showing details of payload structure

In the case of TinyIPFIX the maximum transmission unit is extended to 127 bytes allowing a maximum payload of 102 bytes on the MAC layer as the maximum size supported by IEEE 802.15.4. The resized individual payload field offers enough space to transmit TinyIPFIX messages. The TinyIPFIX message starts with the IPFIX headers (pre-header, IPFIX message header, Set header) that have a range from 22 bytes in worst case down to three bytes in the optimal case with aggressive compression (cf. Section 4.1.2). The resulting packet structure is illustrated in Figure 5.4 where the components are listed with their sizes.

The user can select the performed header compression technique by activating the corresponding flag in the file `ipfix.h`. The following opportunities are available [98]:

- $COMPRESSED\_HEADER = 0$ activates the aggressive compression.
- $COMPRESSED\_HEADER = 1$ activates the uncompressed header version with 22 bytes header size (worst-case).

The implementation is specified in the file `tinyipfix.nc`. The next part of the TinyIPFIX packet is either a Template Record or Data Record.

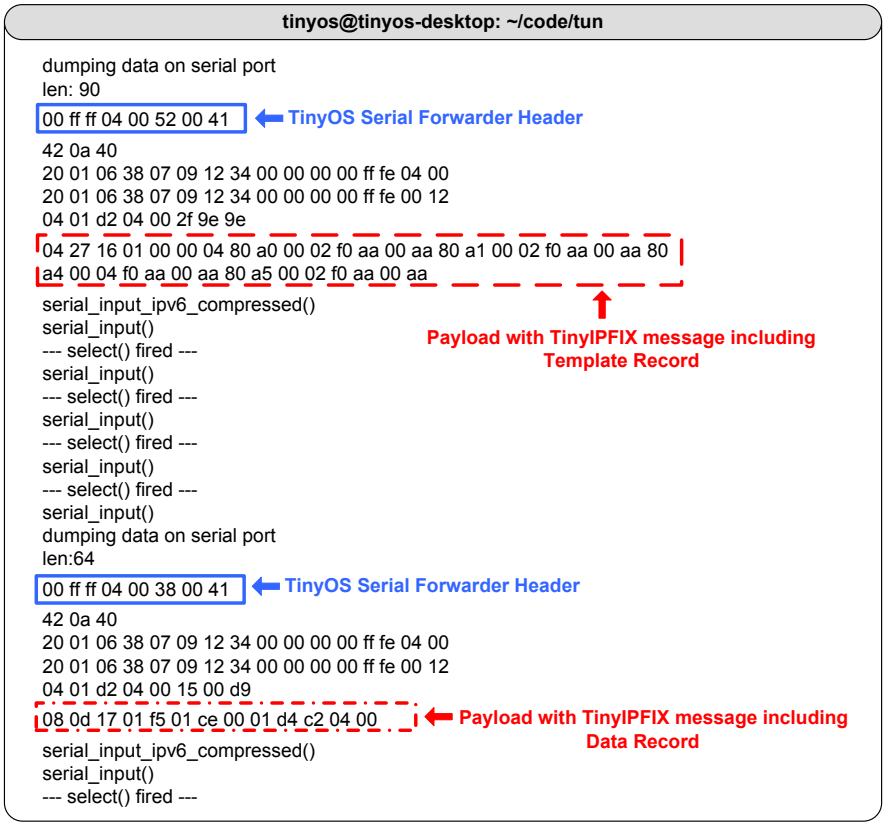


Figure 5.5: Tunnel recording of TinyIPFIX transmission using BLIP

Figure 5.5 shows a transmission example for Template and Data transmission using TinyIPFIX in a TinyOS packet in hexadecimal format, where blue boxes indicate the TinyOS Serial Forwarder Header[7] and red boxes the individual TinyIPFIX payload. In between those boxes information required by the chosen IP communication protocol (here: 6LoWPAN in compressed version using long addresses [69]) is contained, e.g. IP addresses, port information, and payload length.
In the example shown, the value `len` indicates a Template transmission of 90 bytes length (respectively a Data transmission with 64 bytes). Those figures include all headers and individual payloads except the ones for IEEE 802.15.4. In the shown data transmission only one Data Record is transmitted including values for temperature, sound, node time, and node ID.

[7] Source: `http://docs.tinyos.net/tinywiki/index.php/Mote-PC_serial_communication_and_SerialForwarder_(pre-T2.1.1)`

Data transmission could include several Data Records, because the maximum payload size of 102 bytes is not scooped. Assuming the additional Data Records refer to the equal Template, three additional Data Records would fit into the message. With the underlying BLIP stack the maximum IPFIX payload size can be expanded up to 1,024 bytes, because BLIP supports packet fragmentation.

Recorded payload of the template transmission (marked red dashed) shown in Figure 5.5 is 39 bytes long and can be decoded as follows:

- `{04 27 16}` → TinyIPFIX header in aggressive compression format whereas `{27}` indicates a total TinyIPFIX payload of size 39 bytes.
- `{01 00}` → Set ID (here: 256)
- `{00 04}` → Number of Template Fields (here: 4)
- `{80 a0 00 02 f0 aa 00 aa}` → Template Fields for *Temperature* value which include the following information:
  - `{80 a0}` → *Type ID*
  - `{00 02}` → *Data Length ID* in bytes
  - `{f0 aa 00 aa}` → *Enterprise ID* (here: 403767130)
- `{80 a1 00 02 f0 aa 00 aa}` → Template Fields for *Sound* value
- `{80 a4 00 04 f0 aa 00 aa}` → Template Fields for *Node Time*
- `{80 a5 00 02 f0 aa 00 aa}` → Template Fields for *Node ID*

As can be recognized in the Template Record, the Enterprise ID is, in this case, always the same, because all values are collected with one node and all measurement components are produced by the same hardware vendor. In this case, the IANA ID `{f0 aa 00 aa}` was assigned to this hardware vendor. The Type ID is unique for each value. Four values are transmitted in the Template Record. The Data Length ID depends on the value size specified by the developer. [98]
The recorded payload of data transmission (marked red dashed and dotted) shown in Figure 5.5 has a length of 13 bytes. The first three bytes `{08 0d 17}` build the TinyIPFIX header indicating that the aggressive compression technique is activated due to its dimension. It can also be verified by the flag definition in file `index.h` where `COMPRESSION_HEADER` was set to zero. This hexadecimal sequence must be translated into binary format, which results in the term `{000010000000110110111}` and can be interpreted based on the knowledge of the header structure introduced in Section 4.1.3.3 as follows [98]:

- `{00}` → `E1=0` and `E2=0` → The optional fields `Ext.Sequence Number` and `Ext.SetID` are not expected.
- `{0010}` → `Set ID Lookup Index` which specifies the referred Template for decoding purposes (here: 256).
- `{0000001101}` → `Length` of payload including TinyIPFIX header and Data Record (here: 13 bytes).
- `{10111}` → `Sequence Number` (here: 23).

The remaining 10 bytes in the payload include the TinyIPFIX data itself whereas the following information is interpreted by using the referred Template with ID 256 and the stored XML file on the gateway as an input:

- `{01 f5}` → Temperature value: 24.11°C
- `{01 ce}` → Sound value: 462
- `{00 01 d4 c2}` → Node Time: 117.18 sec
- `{04 00}` → Node ID: 1024

5.1.2 TinyIPFIX wiring under TinyOS 2.1.1

As a first step the TinyIPFIX protocol was implemented and modified to the requirements of sensor node hardware as discussed in Section 4.1. The sensor nodes periodically collect data. The measured raw data is then transported to the *TinyIPFIXlibrary*. The task of this library is limited to encoding and preparing Data Records in TinyIPFIX format. The Template specified the value ordering in the Data Record previously. Finally, the prepared packet is forwarded to the Network Handler and transmitted via UDP in the wireless sensor network towards the sink. The Template Record must be announced before data itself is transmitted in order to ensure a successful decoding of outgoing Data Records (cf. Section 4.1.2). Now the sensor node can transmit its TinyIPFIX packets including Data Records. [20, 98]

Figure C.1 illustrates the simplified application wiring in TinyOS. The term *wiring* describes the semantics in which way all components under TinyOS are linked with each other [25]. Red encircled are the main operative components. The generic components (e.g. `IPFIXDataSampler-16C` representing temperature or light) can exist multiple times, because only one value can be reported by each module instance. The control flow of main operative components (interface) can be split into the following steps [20, 98]:

1. `ControllerC` queries the sensors.
2. `ControllerC` reports the sensors' values to `tinyIPFIXC`.
3. `tinyIPFIXC` is now responsible for the construction of the TinyIPFIX packets.
4. `tinyIPFIXC` transmits the TinyIPFIX packets encoded in a byte array back to the `ControllerC`.
5. Those byte arrays are forwarded to the `NetworkHandlerC` by the `ControllerC`.
6. Finally the `NetworkHandlerC` implements the network communications depending on the transmission protocol used and the packet can be transmitted.

The collection of sensor data was implemented via an interface. This interface must ensure an adaptation with minimal changes to the application code. Each sensor measurement is linked to an IPFIX Field and an Enterprise ID. Currently, Enterprise IDs for sensor data are not yet registered by IANA, which must be done

before a real implementation (cf. Table 4.1). Those IDs must include semantics (e.g. purpose and value range) and type length. All these semantics are essential to ensure interoperability. If a new vendor wants to use TinyIPFIX, the corresponding Enterprise IDs and the ID similar to those in Table 4.1 must be specified. [20, 98]

As a result required TinyIPFIX Templates can be generated automatically at the start of a sensor node when the node queries all connected sensors with the pre-known semantics. These semantics are known to compile time for each TinyIPFIX application. For example, the compiling command `'SENSORBOARD=mts400 make iris blip'` specifies a data collector using sensor platform IRIS with sensor board MTS400 and supports routing by BLIP. [20, 98]

Data Records are constructed in the following way: Periodically the sensor node's hardware sends out a read command to its sensor board. This read command is addressed to all connected sensors on the previously specified sensor board. In return sensors answer with their individually measured values. Depending on the latency of the network the order of incoming values vary from the order of read commands. In order to ensure the correct value order in the resulting Data Record each sensor needs to be associated with its respective Field ID, Enterprise ID, and Field Length. The implemented bidirectional interface `IPFIXDataSampler` supports this design. Each sensor is linked to exactly one `IPFIXDataSampler`. Figure C.1 has two `IPFIXDataSampler` links, which means that in the wiring shown two different sensors (here: Temp and Light) are linked [20]. Those multiple wirings allow flexible extensions in the implementation (cf. Figure 5.6). [20, 98]
The required components are initialized in lines 7 to 9 with their corresponding aliases. In lines 12 and 13 hardware sensors are connected to IPFIX wrappers, followed by connecting the wrapped sensors to the applications in lines 14 and 15. The arrow indicates the connection established between components. Those arrows point from the user of an interface to a provider of the same interface. The creation of the IPFIX Template is based on the wiring, which is discussed in detail in reference [98].

```
 1 configuration ControllerAppC{}
 2 implementation{
 3     components ControllerC as App;
 4
 5     // Component Intitialize
 7     components new IPFIXDataSampler16C(0x80A0,0xF0AA00AA) as Temp;
 8     components new IPFIXDataSampler16C(0x80A2,0xF0AA00AA) as Light;
 9     components new TempHumc() as TempSens, new TaosC() as LightSens;
10
11     // Connection of hardware sensors to IPFIX wrappers
12     Temp.Sensor -> TempSens;
13     Light.Sensor -> LightSens;
14     App.Sampler -> Temp;
15     App.Sampler -> Light;
16        }
17 module ControllerC { ...
18     uses interface IPFIXDataSampler as Sampler;
19        }
20 implementation {...}
```

Figure 5.6: Example of wiring *IPFIXDataSampler* providers to *CollectorC* [98]

If a node enters the established wireless sensor network, the first task is the announcement of the Template Record used, followed by the Data Records after data acquisition. Figures A.1 and A.2 show the performed tasks in the style of the interface *AggregatorC.nc*, because it is the same procedure as performed by TinyIPFIX itself just with little modifications in variables. Those tasks can be used for every

construction of sets. Both task blocks are embraced with signaling calls to indicate either start or end of the set construction. In this example the Template ID of 256 is included in the Data Set calls in order to ensure the successful decoding later on in the system. The big difference between both task blocks is in the FOR-Loop. For Template calls the Field ID, the Field Length, and the Enterprise ID is specified. In comparison, in Data Set's FOR-Loop the value and the size of the value are specified. Those two lines are based on the standardization by the IETF for the IPFIX protocol [19] and the IANA registration [74].
Figure 3.3 indicates that a Set can include several Records where each Record consists of one or more Fields. Those cases can also be captured by the described calls with help of wiring TinyIPFIX under TinyOS with multiple interface links.

5.2 Implementation of TinyIPFIX-Aggregation Framework

The TinyIPFIX protocol itself is an efficient data transmission protocol, which can be extended by optional functions. The TinyIPFIX-Aggregation framework is such an optional function. Compared to Figure 5.1 the aggregator nodes (marked grey) support aggregation techniques related to Section 4.2.2.
Figure 5.7 shows the modified scenario setup. The upper wireless sensor network infrastructure illustrates the message flow when aggregator nodes perform message aggregation compared to data aggregation in the lower wireless sensor network infrastructure. First of all, it can be recognized that the number of messages in the wireless sensor network is reduced on the communication paths from aggregator nodes toward the sink. In the case of performing aggregation within the wireless sensor network, new Template and Data Sets are required depending on the aggregation mode used. The aggregation control logic for both cases is described in the next section.

In Section 4.2.2 the design decisions for the implementation of the TinyIPFIX-Aggregation framework were discussed and implemented within the scope of the bachelor thesis by Benjamin Ertl [102]. In the established wireless sensor network the User Datagram Protocol (UDP) is used on the transport layer included in BLIP. In order to perform aggregation in such a network the incoming UDP messages must be redirected from the radio control to the aggregator interface instead of direct transmission to the next hop in the network. This functionality is performed by the aggregator, which listens to a predefined and pre-bound UDP-port specified during compiling of the protocol code. The sensor node that performs aggregation knows this UDP-port and its own IPv6 address. [102]
If the aggregator receives UDP packets, an internal signaling event is activated. This signaling event is performed by the main application component to the aggregation component by interface commands. In general, after receiving this signaling event the aggregator needs to make two decisions regarding the received packet [102]:

1. The aggregator must distinguish between compressed and uncompressed TinyIPFIX packets (cf. Section 4.1.2).

2. The aggregator must distinguish between Template and Data Set.

In the presented network setup the aggressive compression approach for TinyIPFIX headers is performed as shown in Figure 4.4c. The decision concerning the compression type of the TinyIPFIX packet is determined by the first two header bytes.

If the hexadecimal value 0x000a is coded in those two bytes, the received packet is an uncompressed TinyIPFIX packet; otherwise it is a compressed packet. The pre-header of the aggressive approach includes a `SetID Lookup Index` field, which can have two values. If `SetID Lookup Index` = 1, the packet includes a Template Set; otherwise a Data Set. The above mentioned pre-work is included in the decision tree for the TinyIPFIX-Aggregation framework in the first grey diamond - called `Data preprocessing` - illustrated in Figure B.1. [102]

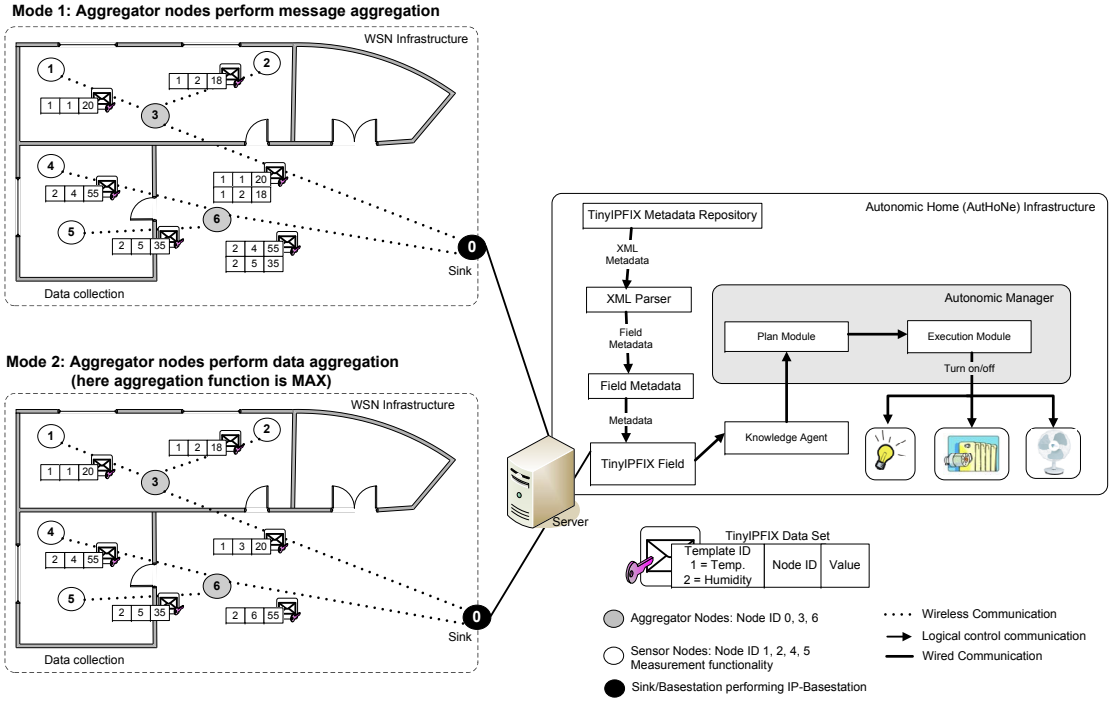


Figure 5.7: AutHoNe setup: aggregation support in simplified message structure

5.2.1 Algorithm for Message Aggregation Functionality

In the following, it is assumed that message aggregation is performed which is illustrated in the left part of Figure B.1 and in the upper left part of Figure 5.7. The functionality was implemented in the aggregation control procedure part of *AggregatorC.nc* shown in Figure B.2 [102].

If those decisions are made, the next step is forwarding Template and Data Sets. The receiving reader function is responsible for pre-processing of data. It checks the data's source in order to ensure the correct assignment of Template and Data Sets for upcoming operations. This is a very important step, because decoding can only be successful if the order of the included Data Fields corresponds to the order announced in the Template Fields in the referred Template Set (cf. Figure 3.3). [102]
Based on this pre-processing the aggregator is able to recognize new information such as new Template and Data Sets or updates. All this pre-processing takes place in the internal buffer of the sensor node. If updated information is received, existing information is overwritten in the buffer. In this case, the aggregation control receives a signal indicating if aggregation can be performed or not. In the latter more incoming packets are requested before the aggregation can be performed. In the case of receiving an unknown Template Set, the buffer is extended by this information together with the IPv6 address of the source node. If unknown Data Sets

are received, they are buffered only if the new Template Set was buffered before; otherwise they are discarded. [102]
Every time an update of the buffer occurs the aggregation control validates the status of the aggregation component. This validation includes the predefined degree of aggregation (DoA) and the predefined buffer number of Template and Data Sets. If predefined thresholds are reached, aggregation control goes on with pre-processing. Depending on the buffered data, the aggregation control disposes of the construction type - aggregated Template Set or aggregated Data Set - and the destination - next sensor node or sink directly. The corresponding procedures are activated. [102]

5.2.2 Algorithm for Data Aggregation Functionality

Now it is assumed that the TinyIPFIX-Aggregation framework performs data aggregation (cf. Figure B.1 right part, and Figure 5.7 lower left case). Data pre-processing routines are the same as in mode 1 with the only exception that the aggregation control logic does not decide to pre-process received data instead of performing message aggregation. The kind of performed aggregation function is user-driven on request. Therefore, it is already known which aggregation function (e.g. MAX, MIN, AVG) should be performed on what kind of data (e.g. temperature, humidity). Those requests have input on the receiving packets and the expected raw data itself. [102]

As pointed out before, each aggregator has a predefined degree of aggregation and a buffer size for Templates and Data Sets. When the threshold of buffered Data Sets is reached, the task `makeAggregateData` in *AggregatorC.nc* is activated. An aggregated value over the buffer set is computed. The activated task generates a new TinyIPFIX Data Set based on the requested aggregated value. At the same time the task `makeAggregateTemplate` computes the newly required Template Set to insure successful decoding at the next hop or sink. Both tasks are shown in Figures B.3 and B.4. [102]

5.2.3 Update Support of Aggregation Functionality

During the system run the aggregation mode can be modified by user input. The user can become connected directly to an aggregator node using the implemented UDP-Shell via *netcat* [102]. For successful performance the IPv6 address of the aggregator and the communication port must be known. The default aggregation type is message aggregation. [102]
If the user is connected, he can decide if the degree of aggregation or the performed aggregation function should be modified. In the first case the subroutine - called `doa` - is activated. Here the degree of aggregation can be reduced down to $DoA = 1$, which is equivalent to normal forwarding without any modification; the degree of aggregation can be increased up to the previously hard coded maximum threshold before node deployment. [102]
If the user wants to modify the performed aggregation function, the subroutine - called `select` - is used. The user can decide between MAX, MIN, AVG, and ALL together with the sensor value he wants to perform the function on, such as temperature, sound or light. [102]
Other offered subroutines in the UDP-Shell are `poll` and `sendOff`. If `poll`-mode is activated, the aggregator directly transmits its buffered Template and Data Sets without performing any aggregation to the next hop. In the `sendOff`- mode all sensor nodes connected to the aggregator are requested to re-address their packets. [102]

The last possible mode is called `reset`. This mode recalibrates the connected aggregator in order to perform message aggregation in the default setup and not data aggregation anymore. It can also be used to be able to jump between the subroutines of the UDP-Shell. Depending on the application scenario new subroutines can be integrated by modifying the TinyOS *ShellCommandC* component, e.g. to ask for the current node configuration. [102]

5.3 Implementation of the TinyDTLS Solution

Corresponding to the described strategies in Section 4.3.2 for secure data communication and transmission in a wireless sensor network, the already established network is extended with an DTLS implementation within the scope of the master thesis by Thomas Kothmayr [81]. Figure 5.8 illustrates the implemented solution. The implemented DTLS code parts are adapted to the hardware resources of the OPAL nodes that include an onboard TPM chip [33]. The underlying operating system is still TinyOS 2.x which supports routing by BLIP implementation and UDP transport. [121, 81]
The implemented DTLS client supports server authenticated and fully authenticated DTLS handshakes. Those handshakes are secured via RSA X.509 certificates. The DTLS server implementation is based on OpenSSL 1.0.0d [134]. The server solution was chosen due to the general characteristic of a wireless sensor network where multiple sensor nodes plan to transmit their data on a secure way (e.g. using DTLS) to the sink. The following subsections give further details about the DTLS client and server side solution as well as the management of certificates. [121, 81]

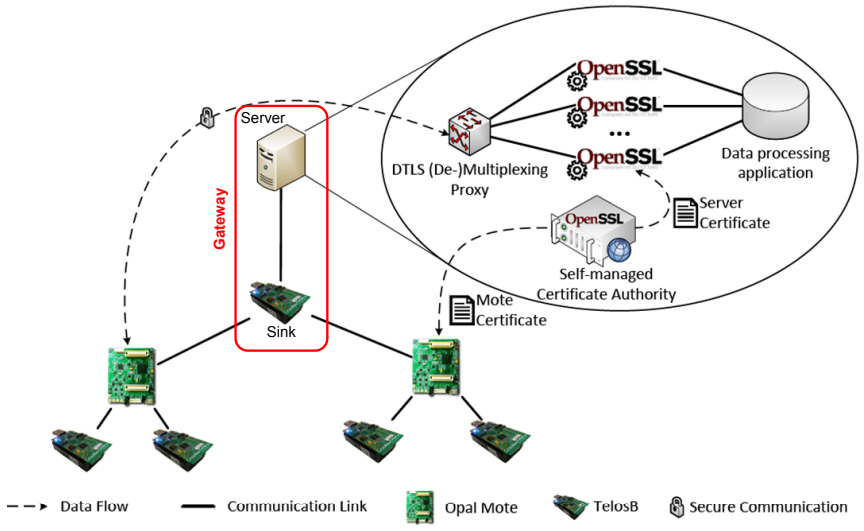


Figure 5.8: Overview of the established DTLS implementation

5.3.1 TinyDTLS Client Implementation

Due to the expected memory consumption IRIS and TelosB nodes are too limited. Therefore, the OPAL node is the technology of choice, which also has a TPM chip included. The global goal for the presented implementation is still the elegant support and usability of components achieved by encapsulation as shown in Figure C.2 [81]. The legend is the same as in Section 5.1.2. The implemented client has to perform the following three steps [121, 81]:

1. Client must issue a connection command.
2. Client has to wait for an event which signals the completion of the DTLS handshake.
3. Client can start with data transmission.

From this point on no further action by the client has to be performed if certificates from the DTLS handshake are stored. The completion of a DTLS handshake is indicated by an event signal. Another signaling event occurs if the client receives any data. [121, 81]

As discussed in Section 4.3, in order to ensure secure communication, cryptographically operations must be performed. In the presented solution the cryptographically operations are divided into two groups. The first group - the symmetric primitives, such as SHA-1, MD5, AES, and an abstract HMAC module - were implemented as software components. Whereas the second group - the RSA functionality - is supported via hardware solution. For the hardware solution the TPM chip is used and is accessed through corresponding drivers. [121, 81]
Another problem that had to be faced was the size of the maximum transmission unit. This was solved by the implementation of a fragmentation layer. This layer is responsible for the interactions between the network and the implemented `DTLSFragmentationLayerP`. One of the tasks of this layer is the fragmentation of messages to meet the maximum transmission unit requirement. Another task is the outgoing packet modification with a DTLS header. Once a secure connection is established another task for this layer is the de- and encryption operation of messages. If those messages have the required format, they are passed on in plain text to higher layers for further processing. The fragmentation layer is also responsible for buffering tasks. Those can occur if messages cannot be passed on directly to their destinations. [121, 81]
On the same level as the fragmentation layer, a second layer exists. This second layer, called the handshake message layer, mainly supports message parsing of received packets as well as synthesizing of outgoing handshake messages as needed for secure connection establishment or session key establishment. As indicated in Chapter 4 different message types exist in the established network. Different modules handle those different message types. In Figure C.2 those modules for different DTLS messages are called `DTLS...P` and drawn in normal boxes. [121, 81]
A special role in the established solution has the handshake layer implemented with the module `DTLSHandshakeP`, which uses both mentioned layers. This module is responsible for the current state of the activated DTLS handshake and also manages the DTLS reliability mechanism. Those tasks are performed in a state machine. If an expected message is missed, it indicates a resend after a predefined time interval. If the missing messages are received, the connection establishment follows the normal task row until a secure connection is established again and data transmission can start indicated by a signaling event of the `DTLSNetworkP` module. [121, 81]
The cipher suite `TLS-RSA-with-AES-128-CBC-SHA` is supported in the established DTLS handshake protocol [121, 81]. This cipher suite supports 128 bits of security and is also fast on the microcontroller used [83].

5.3.2 TinyDTLS Server Implementation

In contrast to the DTLS client solution for the DTLS server implementation nearly no resource limits exist, because its functionality is implemented on the gateway.

It is assumed that the gateway consists of a sink, which is the 'connector' between wireless and wired environment and a PC or a server with unlimited resources.

As illustrated in Figure 5.8 an OpenSSL solution was implemented. In comparison to the standard OpenSSL 1.0.0d implementation different modifications are essential. Here modifications are divided into groups: (1) dealing with hardware requirements and (2) dealing with message handling during the handshake performance [121, 81].

The first group of modifications is essential to support compatibility with the TPM hardware on the OPAL node used [121, 81]:

- First, the padding for RSA signature verification uses RSA Cryptography Standard PKCS#1 version 2 instead version 1.5 [135].
- Second, the client has to sign a SHA1 hash instead of the concatenation of a MD5 and SHA1 hash which would have a size of 36 bytes, and would be too long for the TPM used.

The first mentioned modification concerning the padding is based on the fact that a client sends out a 32-bytes nonce encrypted with the server's public key in the `ClientKeyExchange` message during the handshake using RSA. In the case of the TPM integrated in the OPAL platform a optimal asymmetric encryption padding is applied if an encryption is performed using RSA keys. For the presented implementation OpenSSL was modified in the following way: If decryption with PKCS#1 version 1.5 has failed, decryption of the `ClientKeyExchange` message is retried by using the optimal asymmetric encryption padding. [121, 81]
The second mentioned modification deals with the required modifications of the hashes in the client's `CertificateVerify` message. Here the client has to sign a SHA1 hash instead of the concatenation of a MD5 and SHA1 hash, which would have a size of 36 bytes and would be too long for the TPM used. Thus, the MD5 part was removed. The resulting modified OpenSSL followed the same principle as with the RSA padding: If the verification via a signature over SHA-1 and MD5 fails, the verification of the signature over a SHA-1 hash is just reattempted. [121, 81]
The previously mentioned modifications seam to violate the TLS standard, but do not weaken security significantly [81]. The performed optimal asymmetry encryption padding by OPAL's TPM is a stronger padding than the PKCS#1 version 1.5 padding. Therefore, it is concluded that security is not weakened. Using only a SHA-1 hash instead of a concatenation of SHA-1 and MD5 weakens due to the change in hashes security. It is considered that security is still the same, because the SHA-1 provides 80 bits of security and adding the MD5 hash with a provision of 64 bits of security does not equal 144 bits of security. In order to break the hash codes an attacker must perform $2^{80} + 2^{64}$ operations which does not even equal 81 bit of security, because $log_2(2^{80} + 2^{64}) < 80.00003$. [121, 81]

The second group of modifications deals with handling of missing messages during the handshake performance. Normally a server concludes its handshake with the messages `ChangeCipherSpec` and `Finished`. After that it clears its buffer and monitors DTLS handshake messages during a running connection. Those messages sent by the server are essential for the client to conclude the handshake on its side. Problems occur if one of these messages is lost and the client attempts to resend the message. Those requests will not be recognized by the server, because the

server believes the handshake has successfully been completed. In the established implementation importing bug fixed from DTLS v1.2 down to the used DTLS v1.0 implementation solves this problem. The bug fix requires resending of the last server messages if something is received before it is expected. [121, 81]
The third group of modifications is needed to allow enough reaction time for wireless sensor network components. An example is the timeout value, which must be big enough to allow data processing on receiving data in order to avoid retransmissions.

The available OpenSSL solution does not provide a ready-made server application. Each user has to implement his own server in order to utilize the security functionality provided by the OpenSSL framework. In the case of this dissertation a DTLS proxy application was implemented in order to mediate between the DTLS speaking clients and the existing server application. The presented solution is a DTLS-proxy application [81]. On the one hand, the proxy handles the connections with the wireless sensor network by accepting DTLS handshake requests of the sensor nodes and, on the other hand, directly performs decryption operations of received DTLS application messages. As a result of decryption, data is available in plain text which is forwarded to the application (e.g. packet listener). The communication between sensor nodes is still encrypted. The forwarding via UDP is based on the known IP address and the specific port, which is stored in the DTLS-proxy working as a kind of lookup table for further operations. This implemented multiplexing solution via DTLS-proxy prevents re-implementation of a complete DTLS server for each new application. [121, 81]

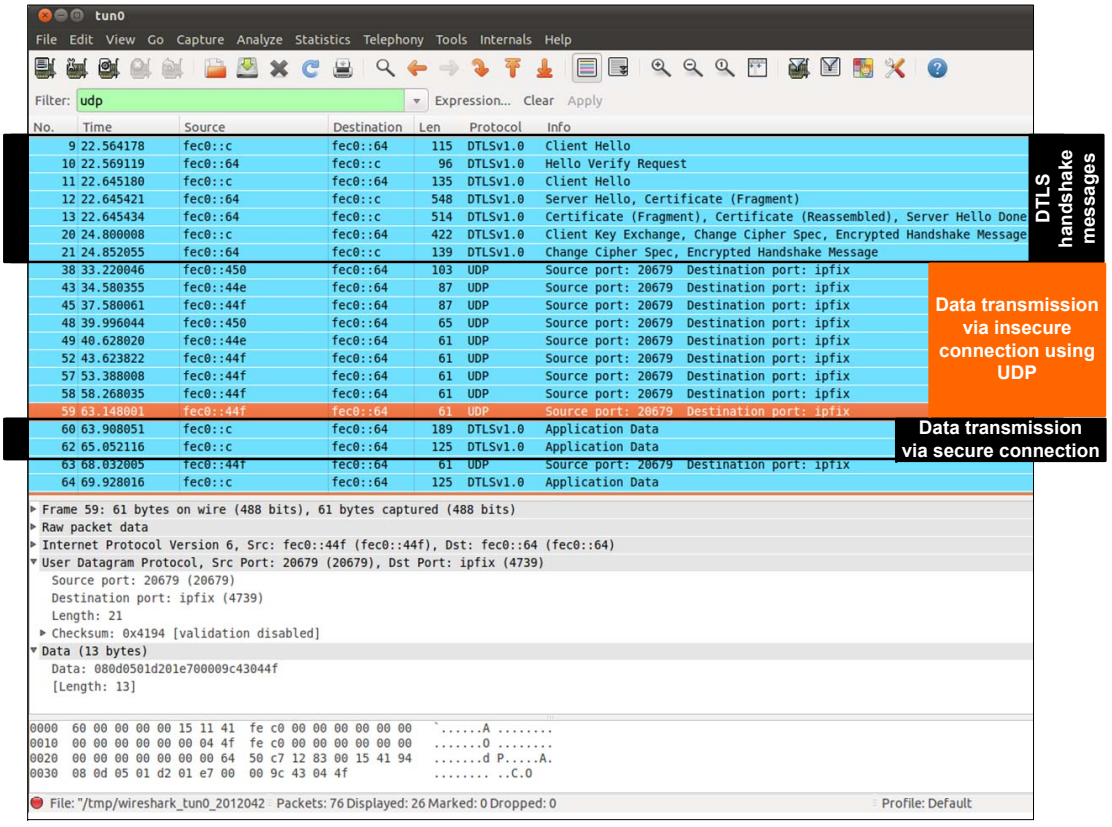


Figure 5.9: Packet transmission captured by Wireshark on channel `tun0`

Figure 5.9 shows a Wireshark[8] recording on channel `tun0`. In the recorded case the OPAL node has the address fec0::c and gateway has the address fec0::64. The first seven black marked and encapsulated messages build the DTLS handshake. Normally, the DTLS handshake consists of six message exchanges (cf. Figure 4.9). Due to the limited message transmission unit, the fourth message of the protocol must be split into two messages (marked with no.12 and no.13) with 548 bytes (resp. 514 bytes length). In the example shown, the OPAL node sends 672 bytes of information and receives 1,297 bytes from the gateway during the handshake.
After the successful DTLS handshake the IRIS nodes with MTS300 (nodeID=1103 resp. 0x44f) or MTS400 (nodeID=1104 resp. 0x450 and nodeID=1102 resp. 0x44e) are activated as measurement entities. They use TinyIPFIX as transmission protocol as indicated by the following recorded packets, for example the orange marked packet.

The OPAL node used also performs TinyIPFIX-Aggregation with $DoA = 2$ (cf. Section 4.2.2). After a few seconds the measurement devices with ID 1104 and 1102 are assigned to the OPAL node. From this point on data from those IRIS nodes are transmitted via the DTLS secured connection to the gateway.
The unassigned IRIS node with ID 1103 still transmits its data via an insecure UDP connection to the gateway, which is indicated under Wireshark with `destination port:ipfix` and `protocol: UDP`. This situation can change in two cases: Either another OPAL node enters the network establishing a secure connection to the gateway and becomes the free node assigned or the currently integrated OPAL node increases its degree of aggregation by one.

5.3.3 Management of Certificates

In the presented implementation different certificates are essential. Each node and each server within the network needs a certificate. Depending on the size of the deployed wireless sensor network the amount of certificates calls for a management solution for the certificates. Because the server side works with OpenSSL, the current implementation also uses management functionalities provided by OpenSSL, such as certificate creation. [121, 81]

For the presented solution a self-managed certificate authority (CA) is implemented which allows the creation of virtually identical certificates. In general, the certification creation for a client includes the following step. The sensor node presents a so called *Certificate Signing Request* (CSR) to the certificate authority including information about the node (e.g. common name, location, institution), and a SHA-1 hash signed with the node's private key. In the current wireless sensor network the client is an OPAL node with a TPM chip. Currently this cannot be programmed with a specific private and public key pair. Instead the TPM generates the key pair manually, whereas the private key does not leave the secure storage of the TPM. A template Certificate Signing Request is, therefore, generated with the node ID and $institution = CSIRO$ as the producer of the hardware. [121, 81]
Due to missing keys on the OPAL nodes, a throwaway RSA key is generated and the information is signed with it. A new calculated signature is then generated in the TPM and replaces the previously used signature with the TPM's signature. Then the finally produced certificate signing request is sent to the certificate authority to obtain the required X.509 certificate for the sensor node in order to perform the DTLS handshake as described in Section 4.3.2. Another drawback of the OPAL

[8] Wireshark - the world's foremost network protocol analyzer `http://www.wireshark.org/`

hardware is the so called factory default mode [33]. This means: all used OPAL nodes in the wireless sensor network have the same TPM chip and also use the exact same key pairs. Developers at the producer CSIRO currently work on this problem to produce individual key pairs on new hardware deployments in order to allow key pair computation during the *Certificate Signing Request* phase directly on the sensor node. [121, 81]

5.4 Implementation of the Graphical User Interface

As motivated in Section 4.4 the existing graphical user interfaces for wireless sensor networks focus on simulation scenarios and not on real-time systems. Therefore, a graphical user interface was developed during this dissertation in order to support configuration of the used hardware and to visualize collected data. On the one hand, the developed graphical user interface supports the established algorithms and protocols such as TinyIPFIX, TinyIPFIX-Aggregation, and data import/export. On the other hand, it can be used for other application scenarios using modified algorithms. This flexibility is realized by working with a virtual representation of the real sensor network and the implemented options requiring user information (e.g. code location, hardware specification) in order to configure the whole system. The graphical user interface has a modular structure, which gives the user the opportunity to integrate new extensions (e.g. include new visualization tools, code buttons for programming purposes) with fewer overheads. [133]

After a detailed description of the realization of the virtual representation of a wireless sensor network in Section 4.4, this section focuses on the support to configuring sensor devices (cf. Section 5.4.1), the included visualization functionality of the real-time network status (cf. Section 5.4.2), and the included data export/import functionality (cf. Section 5.4.3). [133]

5.4.1 Configuration of the Network Components

The implemented graphical user interface supports compiling of programming code as well as the installation of sensor nodes. Independently from the interface the user has to mount the node hardware to an USB port. In the next step the user starts programming tool Eclipse[9] and activates the interface implementation. Now the user switches to the browser and connects to `localhost:8000`. Then the graphical user interface becomes visible for the user. From this stage on, the user only works with this interface instead of using terminals with command lines. [133]

The user can now see which menu options are available in the graphical user interface and which programs were activated due to the interface starting via eclipse. Started components are indicated with green rakes on the home screen (e.g. HTTP Server, TinyIPFIX Listener, module for TinyOS). Some components, which are indicated by an encircled exclamation mark (e.g. COSM Uploader), are not activated from the beginning. For example, the COSM Uploader is started if the user decides to upload measurement data to the online visualization tool COSM. In the blue box `WSN Administration` the user can see the different available menu options of the graphical user interface. [133]

In order to program individual components of the wireless sensor network, the user must change to the submenu option `TinyOS` in menu `Hardware` (cf. Figure D.1).

[9] Eclipse: `http://www.eclipse.org/`

The interface gives the user the information about the supported operating systems, which in this case is TinyOS 2.1.1 together with the included modules. The modules include an interface to program the base station, the individual nodes as well as the activation of the IP Tunnel. [133]
Before starting recording of collected data within the wireless sensor network, hardware must be programmed. Therefore, the user jumps into the submenu `Program BaseStation` or `Program Nodes`. Both submenus are structured similarly and consist of six steps [133]:

1. The user must specify the hardware used. Here he can decide between different vendors such as TelosB, IRIS or MICA for example.
2. If a node is programmed and should support data collection, the user must specify the attached sensor board in a second step. In this case the common used sensor boards such as MTS300 or MTS400 are included.
3. Under the section `Extras` optional available support can be chosen by clicking on the option. Due to the fact that the planned application must support IP communication the option `blip` is activated by default.
4. Now the user must specify under the option `Directory` what code should be performed by the node. In this case the user can choose between performing an aggregation, data collection or others. The field to the right of this option selection indicates the location of the requested directory.
5. As a result of the previous selections in steps one to four the `make` command for step five is constructed. The user must now click on the button `make` to start the compiling process. If this is done, the user can specify the connected port of the hardware, and the individual ID of the node if needed.
6. Corresponding to this verification the installing command is constructed which can be activated by the user by clicking on the button `install` in the sixth step.

If this command was successfully executed, the user can repeat the steps one to six for the next hardware. During the execution of step five and six performed commands are displayed in the box underneath the `install` button in order to give a visual feedback of the process status together with the information about the required ROM and RAM. [133]

As indicated at the beginning of Section 5.4 a general graphical user interface does not exist at the moment. Thus, compiling and installing of programs must be done via a terminal. In order to give the user a comfortable way to perform those tasks, the developed graphical user interface was extended. The submenu `Program Nodes` was extended by an input possibility, called `others`, for the user to verify manually what code should be performed. In this case the user can specify the folder of the code in step five, followed by starting the compiling process and installing the program on the sensor node by pushing the corresponding buttons. Due to the manual input the graphical user interface is independent from the underlying installations (e.g. TinyIPFIX, aggregator or collector functionality) as well as from the supported hardware. The design of this feature is similar to the one shown in Figure D.1. [133]

5.4.2 Graphical Feedback of Network Status

When the user has programmed all hardware components corresponding to steps one to six of Section 5.4.1, the wireless sensor network can be activated. In order to activate the network, the hardware programmed with the code for the base station is attached to a USB port (e.g. USB01). [133]
The user now switches to the option `IP Tunnel` in the submenu option `TinyOS`. Here the user specifies the USB port where the base station is connected and types in the password in order to start the tunnel application. Finally, the user clicks the `start` button and the IP Tunnel is established. The user then receives a visual feedback about the running tunnel process by the box underneath. The information provided in this box includes the activated sensor nodes in the network. At the time the tunnel is started, the `start` button changes into a `stop` button where the user can manual abort the tunnel as well as the wireless sensor network. [133]
After the tunnel is established the user can activate the individual sensor nodes. The sensor nodes perform TinyIPFIX (perhaps) with extensions. They are recognized by the tunnel application when they first send their Template Record which is done directly after booting (cf. Section 4.1.2). [133]

If the first Data Record is transmitted, the user can switch to the submenu option `Topology` under menu `Visualization`. Here he receives a live visualization of the current network status including a routing tree and sensor node information as shown in Figure 5.10. In order to visualize this network information the underlying implementation is dichotomous. The left part visualizes the routing topology of the wireless sensor network. Therefore, the underlying implementation requires that the information of the routing driver shell be supported by the BLIP implementation under TinyOS [71, 25]. The routing driver shell establishes a telnet session to `localhost 6106`. The session supports the following commands and currently must be opened in a terminal if commands other than `dot` should be performed [133]:

- The command `links` displays the link state of the network reported by each node router.
- The command `rebuild` requests an update of the topology information of the network.
- The command `routes` displays the cashed routes of the network from each node to the sink.
- The command `stats` shows information of forwarded packets.
- The command `conf` provides information about the router's configuration.
- The command `dot` allows the printing of a picture of the current network topology.

In order to display the topology of the wireless sensor network the `dot` command is performed. The resulting file is converted into a picture format such as PNG, which is included in the graphical user interface. Due to topology changes during runtime of the system, this graph must be updated. This update happens periodically and is specified in the underlying JAVA implementation. Normally, those updates are required if sensor nodes enter or leave the network as well as if routes are updated and the intermediates change. In the graphical user interface the manual activation

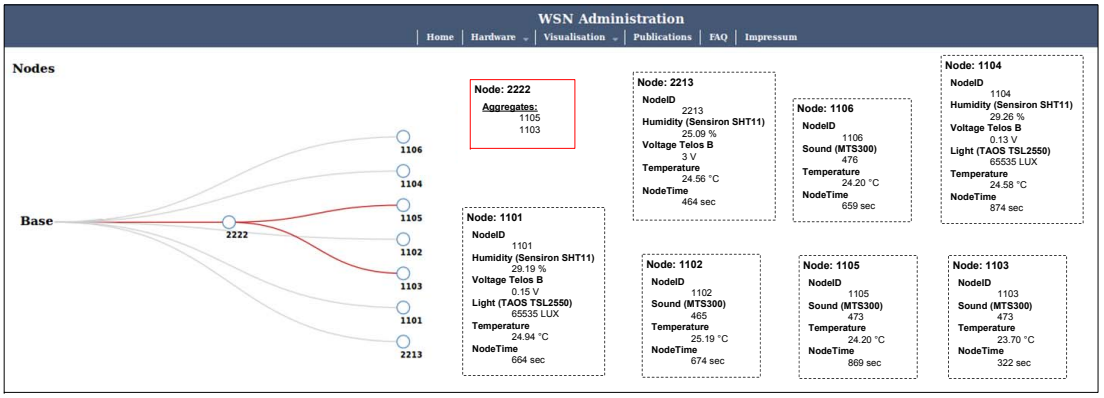


Figure 5.10: GUI - Visualization of the current network status

of the updating command is included as well. Thus, the user can request an update whenever he wants. [133]

The right part of Figure 5.10 illustrates the active nodes within the current wireless sensor network. For each node special information is available. This information is extracted from the received data after it was abstracted corresponding to the functionality of the *WSNDriver* and is available due to virtual representation (cf. Figure 4.11). For example, node 1105 transmits its NodeID, NodeTime, sound value using MTS300, and temperature value. If a node is an aggregator, the available information of the node includes the IDs of the aggregated nodes. An example is node 2222 in this case.

If the user wants to know where exactly a special sensor node can be found in the current topology, he clicks on the dashed node box. As a consequence the communication path of the node down to the sink is highlighted in the routing graph. For example, in Figure 5.10 the user requests information about the node with ID 2222. Therefore, the user clicks on the right part of the visualization on the node 2222 and receives the aggregation information. In this case, the node aggregates the sensor nodes 1105 and 1103. At the same time the corresponding routing is highlighted in the left part of the visualization starting from the source of the data via the intermediate node 2222 down to the sink. With help of this highlighting the user keeps track of each nodes' data flow.

The supported Listener by TinyOS was integrated into the graphical user interface in order to allow the user a more detailed visual feedback of the running wireless sensor network. Here the user can observe the received packet at the gateway live under submenu `TinyIPFIX` in menu `Visualization` (cf. Figure D.3c). The resulting view is a combination of the previously introduced submenu option `Topology` and the Listener recording. Due to the live view of the Listener, the user can directly realize if some problems occur in the system (e.g. decoding problems, format conversion problems using XML file). [133]

5.4.3 Data Export/Import Functionality

The amount of collected sensor data depends on the network's size, on collected values, and on collection intervals. As a consequence this data amount can be huge. In order to analyze this data, it is useful to display the data in plots. Depending on the plotting tools used different plot types are available such as line, point

or step diagrams. Today a variety of tools featuring this task are available as described by Parbat et al. in reference [129]. Those tools can be subdivided into (1) offline and (2) online tools.

In general, offline tools require a tab-delimited input file including all required information for a correct plotting task. Representatives for offline tools are Matlab, R or GNUplot. The implemented graphical user interface in this doctoral thesis supports the analysis via offline tools. Therefore, received data is directly stored in a big file. Usually, the user just wants to plot one subset of data, which generally consists of measurements of one selected sensor node. The big file must, therefore, be subdivided into its components stored in small files. This processing is performed automatically in the background of the graphical user interface. An example result is shown in Figure D.5. The number of small files corresponds to the number of nodes in the active wireless sensor network. In the presented example the network consisted of four nodes whereas three nodes were data collectors and one node (ID 5678) was responsible for secure data transmission via DTLS with activated message aggregation functionality. Thus, the small file for node 5678 only includes time stamps due to the virtual representation of the wireless sensor network as characterized in Section 4.4.2. In order to reduce overhead in small files, some information such as Enterprise ID was deleted. Small files can then be uploaded into different offline tools and processed by corresponding program. In order to perform this upload, the user must specify the location of the executing program and the location of the stored subset file. Therefore, the user can browse an offered archive by the graphical user interface, including all required information. If the executing program is not installed on the running system, the user receives the location of the stored files and is asked to download them (e.g. on a USB stick).

In comparison, online tools request an Internet connection in order to upload data and to display the analysis result directly. Thus, those tools are among the cloud-based platform solutions. A representative for online tools is COSM better known as Pachube [136]. Online tools differ depending on the data input format they request. COSM requests a live stream, whereas other tools require final stored files analogous to offline tools. In order to support different upload possibilities the implementation of the export client is modular as well. For each tool a special module supporting the required data structure must be implemented and included into the graphical user interface.

COSM is a tool, which stores the uploaded data in feeds that can be shared with other people upon request. Each feed stores data received from exactly one sensor node independent of the number of included sensors on the hardware. For example, if the source node is an IRIS node with MTS300, collected values can contain sound, temperature or brightness. For the MTS400 collected values can contain humidity, brightness, temperature, and voltage.
What kind of data such a feed includes depends on the sensor node hardware and on the data the user decides to upload to the homepage of COSM. COSM accepts data in JSON, XML, and CSV format. The implemented module in this dissertation uploads its data in JSON format and is implemented in JAVA. The reason for choosing JSON as the upload format depends on the smaller size compared to XML (cf. Figure 4.2).

In order to change the upload format, modifications must be done in file `CosmAPI.java`. In the file `WSNCosmModule.java` different variables concerning the upload to COSM can be specified, such as the upload interval in seconds. The

resulting diagrams of the visualization are imported back into the graphical user interface and can be viewed under the submenu option `COSM` in menu `Visualization` as illustrated in Figure D.6. [133]

Figure D.4 illustrates the background information including node location information and log information resulting from TinyOS Listener. For the visualization of data within the graphical user interface the user can choose between different granularity, which is specified in the HTML file `index.html`. In addition, the user can specify which subset of data should be displayed. Figure D.6 shows the different options for approximately three hours experiment duration [133]:

- All transmitted information of node 1101 is displayed including the information about the number of COSM-uploads (cf. Figure D.6).
- Nodes 1103, 1104, and 2213 only display the sensor data, and reject the information about NodeID and NodeTime. NodeTime is not essential to display because it can be recalculated from the other graphics or by the log file. The NodeID can be rejected also, because the information is visual in the dashed box at the beginning of each feed. For visualization purposes the user has deleted the information about the number of COSM-uploads as well.

Today the call for data access independent of a direct login to the running terminal occurs. COSM also allows the view on recorded data during real time from external access points, e.g. from a device located in the USA even though the running systems stands in Germany. In order to gain access, the user logs in on the homepage of COSM and chooses the feed of interest. As long as a feed is stored in the cloud it can be viewed via the homepage and the graphical user interface even if the wireless sensor network is inactive.
Currently, the responsible person for the recorded wireless sensor network can add information about each feed on the COSM homepage. This information can include hardware equipment (e.g. IRIS - MTS 300 with active temperature and acoustic sensor), experiment location, date, and duration. In the future the user will be able to publish this additional information via the developed graphical user interface, which will support the export to the COSM homepage directly. Right now, the user may verify whether or not experiments should be deleted in the cloud. But as a backup the experiments are still stored in an archive linked to the graphical user interface on the performing PC or notebook until the user deletes them manually.

5.5 Summary and findings

This chapter briefly described the implementation of the TinyIPFIX protocol with its extensions - compression and aggregation -, as well as the TinyDTLS solution for end-to-end security and the graphical user interface. As specified in Section 3.1 the existing 6LoWPAN implementation supports a maximum transmission unit of 127 bytes under sensor hardware, which includes the RF transceiver CC2420. Figure 5.4 illustrated the total available packet size on the MAC layer including TinyOS based header information and the individual payload (e.g. IP support, sensor data). This figure shows that IPFIX data can be transmitted in such a limited maximum transmission unit, even if additional overhead due to IPFIX message/Set headers limits the individual payload. If aggressive compression on the additional headers is performed the individual payload leaves space for sensor

readings of 81 bytes.
Regarding TinyIPFIX-Aggregation detailed information of the performed algorithms was presented. Additionally, the implemented UDP-Shell was characterized, which allows the user to manipulate the aggregator nodes within a running application. Therefore, the user can choose between different options, such as `select`, `doa` or `reset`.
The implementation of the TinyDTLS protocol was briefly described including an analysis of the exchanged messages during the handshake phase. For performing the TinyDTLS protocol using certificates for authentication purposes OPAL nodes is the hardware platform of choice, because of its included TPM chip and its resources. Additional memory and computational resources are higher on OPAL than on IRIS and TelosB (cf. Section 2.2), which allows the sensor node to perform additional functionality, such as aggregation.
Finally, the implemented graphical user interface was characterized together with its supported functionalities in order to monitor and configure a wireless sensor network.

6. Evaluation of the developed Protocol and its Extensions

In this chapter the implemented TinyIPFIX protocol together with its extensions will be evaluated. At a minimum each evaluation part is subdivided into an analysis focusing on memory and energy consumption, because those are the most limited resources. In a case of unwise usage of the two resources, lifetime of a wireless sensor network can decrease rapidly, which stands in contrast to the original goals. An exception is the evaluation of the graphical user interface where a proof of functionality is done. Finally, it is shown that the developed application protocol fulfills the requirements for a tight integration of a wireless sensor network into a cyber-physical system in Section 6.5.

From this point on an office scenario is assumed as shown in Figure 6.1. The testbed deployed at our department consists of 14 sensor nodes with different characteristics and a gateway, which is marked black in the figure. The gateway is a component consisting of a node with base station functionality and is connected to a server where all applications are running. The characteristic of each sensor node is listed in Table 6.1.

Node ID Decimal	Node ID Hexadecimal	Node Type	Vendor	Node Characteristic
2232	0x8b8	TelosB	Advantic Sys.	Data Collector
2270	0x8de	TelosB	Advantic Sys.	Data Collector
1104	0x450	IRIS	Crossbow Inc.	Data Collector (MTS300)
2222	0x8ae	TelosB	Advantic Sys.	Aggregator ($DoA = 2$)
1108	0x454	IRIS	Crossbow Inc.	Data Collector (MTS300, housing compartment)
2250	0x8ca	TelosB	Advantic Sys.	Data Collector
2243	0x8ce	TelosB	Advantic Sys.	Aggregator ($DoA = 3$)
1103	0x44f	IRIS	Crossbow Inc.	Data Collector (MTS300)
1101	0x44d	IRIS	Crossbow Inc.	Data Collector (MTS400)
1105	0x451	IRIS	Crossbow Inc.	Data Collector (MTS300)
1102	0x44e	IRIS	Crossbow Inc.	Data Collector (MTS400)
1106	0x452	IRIS	Crossbow Inc.	Data Collector (MTS300)
1107	0x453	IRIS	Crossbow Inc.	Data Collector (MTS300, housing compartment)
1110	0x456	IRIS	Crossbow Inc.	Data Collector (MTS400, housing compartment)
12	0xc	OPAL	CSIRO	Aggregator ($DoA = 2$, TPM active)

Table 6.1: Node characteristic for department testbed (cf. Figure 6.1)

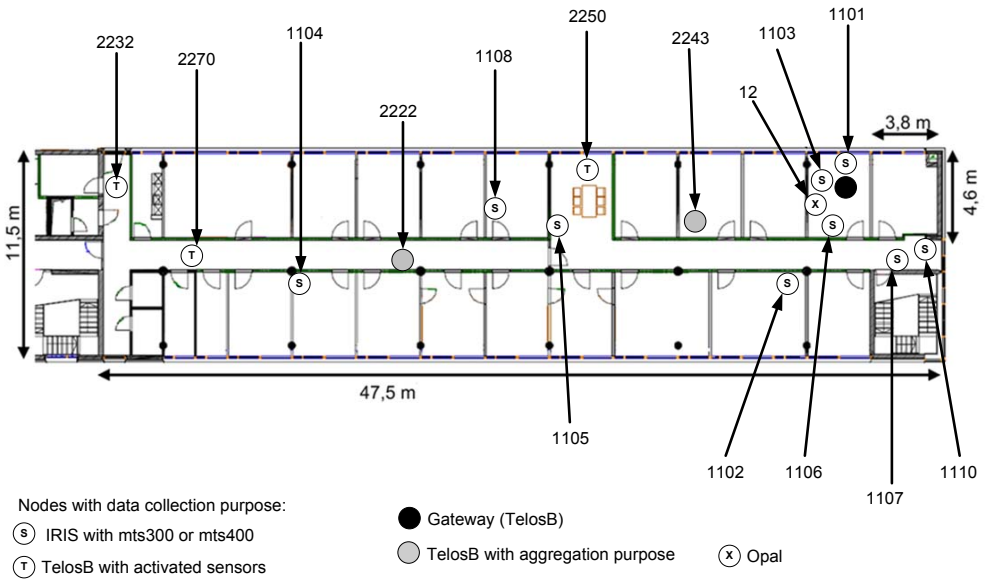


Figure 6.1: Overview of deployed testbed at the department

6.1 Evaluation of the TinyIPFIX Protocol

From this point on we assume a wireless sensor network with the operating system TinyOS 2.1.1. It supports IP communication using the integrated BLIP implementation over UDP. The network is deployed as shown in Figure 6.1.

6.1.1 Memory Consumption

TinyIPFIX was implemented on IRIS and TelosB nodes (cf. Figure 2.2). In order to put memory usage in perspective, each major component is considered separately. Because they interact with each other, memory consumption has been measured in an incremental fashion, starting with the basic scaffold for obtaining measurements and expanding upon that by adding BLIP and TinyIPFIX. The results are shown in Table 6.2. [98]
It shows a maximum RAM consumption of 6,889 bytes when using BLIP and setting the maximum TinyIPFIX package size to 1,024 bytes. Memory consumption of the TinyIPFIX component is only 57 bytes plus twice the maximum defined TinyIPFIX packet size [98]. With those RAM and ROM requests the TinyIPFIX implementation can be performed on both platforms without major problems. Similar results were measured by sensor platform IRIS. [98]

	RAM	ROM	TinyIPFIX packet size
Scaffold	46	2826	-
BLIP	4,738	23,012	-
TinyIPFIX	57	2,972	0
	261	3,182	102
	2,105	3,012	1,024
Total	4,841-6,889	28,810-29,020	

Table 6.2: Memory usage of BLIP and TinyIPFIX on TelosB [bytes]

6.1.2 Transmission Efficiency

As indicated in Section 2.2 resources (e.g. energy) of sensor devices are very limited. In general, energy comes from two AA batteries. In order to ensure less manual input (e.g. changing batteries) and, therefore, a long lifetime, most limiting resource must be used wisely. Energy consumption can be reduced either by supporting energy saving methods or reducing the amount of transmissions. Both aspects depend on each other as mentioned in Section 2.3. The developed transmission protocol TinyIPFIX does not limit the amount of transmission in general, but reduces the periods of active radio due to smaller transmission units. In order to prove this fact, transmission efficiency is the objective of the evaluation in this section. [98]

For the analysis of transmission efficiency the ratio of payload per byte sent using TinyIPFIX is evaluated. Therefore, five assumptions are made [98]:

1. Exactly one Data Set or one Template Set is included in one TinyIPFIX packet.
2. All values in the Template are defined with an Enterprise ID which costs eight bytes for each value.
3. A Data Record includes only one Data Set whereas the number of transmitted values is defined in the referred Template Set.
4. Intervals for retransmissions of Template Record are predefined.
5. For the TinyIPFIX header the modified defensive compression technique is used requiring three bytes (cf. Figure 4.4b).

Together with the above assumptions (1)-(5) the transmission efficiency $t_{eff-TinyIPFIX}$ can be calculated as the percentage of the payload per byte sent as expressed by Equation 6.1 [98]. The upper boundary for transmission efficiency is given by Equation 6.2, where it is assumed that φ is the product of the number of Data Records transmitted between two Template Record transmissions (n), the number of transmitted Data Fields per Data Set (v), and the average size of the transmitted values per bytes (s). It is also assumed that h is the header size used by TinyIPFIX. [98]

$$t_{eff-TinyIPFIX} = \frac{\varphi}{h * (1 + n) + 8 * v + \varphi} \tag{6.1}$$

$$\lim_{n \to \infty} \frac{\varphi}{h * (1 + n) + 8 * v + \varphi} = \frac{v * s}{h + v * s} \tag{6.2}$$

Figures E.1 to E.3 show the results where the transmission efficiency was plotted against the number of data packets transmitted between two Template Records and $s = 2$ bytes was assumed. In order to illustrate different types of packet configuration, the number of values per packet ranges from one to 125. One value per packet is the lowest limit, four values per packet are common for wireless sensor networks, and 125 values per packet is the maximum packet size for IP communication. The analysis of TinyIPFIX transmission efficiency is split into the following cases [98]:

- Case 1: TinyIPFIX packets with default header of size 20 bytes vs header supporting defensive approach in best-case with six bytes size (cf. Figure E.1)

- Case 2: TinyIPFIX packets with default header of size 20 bytes vs header supporting modified defensive approach with three bytes size (cf. Figure E.2)
- Case 3: TinyIPFIX packets with default header of size 20 bytes vs header supporting aggressive approach with one bytes size (cf. Figure E.3)

It can be observed that the header size has a big impact on $t_{eff-TinyIPFIX}$. The impact lowers if the payload becomes larger. The effect of header compression remains important for the performance. Usually in sensor scenarios, a Data Record consists of four values and 16 transmissions between two Template Record transmissions. [98]

In comparison to a Type-Length-Value (TLV) approach data values are always transmitted in one packet together with their meta information. Meta information includes the type and length of the data. In this case the transmission efficiency $t_{eff-TVL}$ is defined as follows [98]:

$$t_{eff-TVL} = \frac{\varphi}{\varphi + v * l_1 + v * l_2} \tag{6.3}$$

Where φ, n, v, s defined as above, l_1 = size of the length declaration in bytes, and l_2 = size of the type declaration in bytes [98]. Assuming $l_1 = 1$, efficiency of a TLV approach only depends on the average size of the transmitted fields s and the number of possible types (l_2). Thus, Equation 6.3 results in $t_{eff-TVL} = \frac{s}{s+1+l_2}$. In order to compare transmission efficiency of the Type-Length-Value approach with the previously evaluated TinyIPFIX transmission efficiency the value s ranges from one to four. The results are summarized in Table 6.3. [98]
As determined in Section 3.2 each Template Field is specified by a `Type ID`, a `Length ID` and an `Enterprise ID`. In the case of IPFIX the Enterprise ID has 2^{32} possible values and Type ID (respectively 2^{15} with two bytes size including the Enterprise bit). Therefore, 2^{47} possible types exist. According to Table 6.3 the Type-Length-Value approach is limited to 22.2% concerning its efficiency. If a header compression is activated in the TinyIPFIX solution and four data packets per Template are assumed, the TinyIPFIX solution already scales better than the Type-Length-Value approach. The TinyIPFIX solution scales better if the number of data packets between Template retransmissions increases. [98]

	$s = 1$	$s = 2$	$s = 3$
$2^8 \rightarrow l_2 = 1$	25.0%	50.0%	66.7%
$2^{16} \rightarrow l_2 = 2$	20.0%	40.0%	57.1%
$2^{32} \rightarrow l_2 = 4$	16.7%	28.6%	44.4%
$2^{48} \rightarrow l_2 = 6$	12.5%	22.2%	36.4%
$2^{64} \rightarrow l_2 = 8$	10.0%	18.1%	30.8%

Table 6.3: Transmission efficiency for a Type-Length-Value approach

6.1.3 Energy Consumption

As mentioned in Section 2.2 energy resources are limited. Therefore, an energy analysis was done in order to verify how much energy is actually spent by transmitting TinyIPFIX packets. Here only TelosB nodes were used, because they are

planned to be used for further extensions such as aggregation and the energy values are bound by networking, which is expected to be similar on other platforms. In order to obtain measurements of the energy spent for transmitting packets, an oscilloscope across a resistor (10Ω) in the circuit of an external power source was connected to a TelosB sensor node. This methodology is described in detail in [15] and follows the experimental setup as shown in Figure 6.2. Calculation of the power consumption by wireless sensor node follows Equation 6.4. Where P_{WS} is the power consumed by the sensor node, V_{SEN} is the voltage across the sensor node, I is the current flow through the sensor node, R_{SHUNT} is 10Ω shunt resistor, V_{SUPPLY} is the voltage of supply unit, and V_{SHUNT} is the voltage across the shunt resistor [15].

$$P_{WS} = V_{SEN} \times I = (V_{SUPPLY} - V_{SHUNT}) \times \frac{V_{SHUNT}}{R_{SHUNT}} \tag{6.4}$$

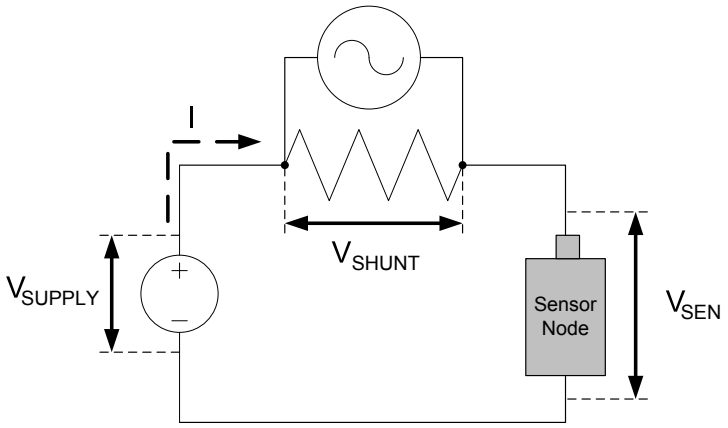


Figure 6.2: Voltage measurement for calculation of energy consumption

In this case, a TelosB node features a CC2420 Radio chip, which is rated at 17.4 mA current draw when transmitting [32]. The average transmission time was measured over 128 samples of IPFIX, TinyIPFIX, and Type-Length-Value approach (TLV) packets. Those include a four byte long time stamp, the two byte long node ID, and two sensor measurements each two byte long. The size of each packet, including meta data, is given in Table 6.4. [98]

Packet Type	t_{send} [ms]	Payload [bytes]	Energy [μJ]
empty	10.48	0	699
TLV 2^8	10.93	14	730
TLV 2^{48}	11.55	34	778
IPFIX Data	11.69	30	779
IPFIX Template	12.3	48	820
TinyIPFIX Data	10.9	13	727
TinyIPFIX Template	11.71	31	780

Table 6.4: Average transmission times and energy for selected packet types

As motivated in Section 4.1.1 one possibility to optimize the transmission efficiency is to use a compressed data format for transmission. The previously introduced

protocols used XML or JSON format for data transmission. Those formats include meta information for each transmitted sensor measurement, which result in low transmission efficiency. Thus, a *Type-Length-Value* (TLV) approach comes to mind for evaluation purposes. Priyantha et al. showed in reference [137] that a possibility for data compressing using binary XML exists where the performed algorithm works with an index, which is mapped to a specific XML tag, thereby reducing the size of a message. This compression approach is comparable to a Type-Length-Value approach where a sensor measurement would always be prefixed with an index into a dictionary, describing the type of a value and the length of the value that will follow in bytes. The length domination can be dropped if the type information is subsuming it, requiring only a type declaration as the only piece of meta information. Assuming that the measurement value itself cannot be compressed, this approach is identical or better than any compressed XML format that uses a mapping dictionary. Thus, it is obvious to compare the transmission efficiency of TinyIPFIX to that of a Type-Length-Value approach where length domination has been dropped. [98]

For evaluation purposes, it is assumed that a Type-Length-Value approach is calculated on total bytes, where one byte is the minimum length of the field `Type` in the Type-Length-Value approach. One byte, therefore, offers the opportunity to represent $2^8 = 256$ values. In order to compare the Type-Length-Value approach with the typical value range of IPFIX a value range of 2^{48} is assumed. This value is calculated by the two bytes (=16 bits) of Type ID plus the four bytes (=32 bits) of the Enterprise ID in a typical IPFIX packet. After the measurement analysis of the results given in Table 6.4 it was found out that the time of energy consumption is largely dominated by a constant factor. On the one hand, this constant factor originates from the medium access protocol; on the other hand, it originates from the time it takes to switch the radio from receiving to sending mode and back. When activating low power listening the variance and average duration of transmission time increased up to one order of magnitude. As a result, the employment of IPFIX, TLV or any other application layer protocol does not have a large impact on overall energy usage when using the default TinyOS settings and relatively small packages. [98]

In order to evaluate the implemented TinyIPFIX protocol on a more complex network, it was remotely tested on the Harvard Sensor Network Testbed (MoteLab)[10]. This wireless sensor network is located on the campus of the Harvard University and distributed over three levels in a brick stoned house with around 184 active TelosB nodes. The run was done with 77 TelosB nodes where 76 sensor nodes sent sensor measurements to a single gateway node over a maximum number of six hops [100]. Several experimental runs with duration of 30 minutes were executed. [98, 100]

The start up phase took two minutes in which no sensor measurements were sent to allow BLIP to establish routes within the network. Next, sensor nodes took measured sensor values (here: temperature, humidity, and internal voltage), and sent them with their node ID and time since they booted to the edge router. [98]

Every time a sensor node tried to send an TinyIPFIX or TLV packet it logged the attempt to a database. This figure was then compared with the number of packets captured with Wireshark on the receiving end to determine the percentage of packets that had arrived successfully. If a TinyIPFIX packet contained sensor data that could not be parsed due to Template Record loss, it was counted as unreadable.

[10]Harvard Sensor Network Testbed (MoteLab), `http://motelab.eecs.harvard.edu/`

For statistical purposes the number of sent or forwarded packets from each sensor node was collected. [98]
The energy amount spent by the whole wireless sensor network was calculated by multiplying statistic with the measurements for energy usage from Table 6.4. Inputs, like routing and overhead, were not included into the calculation of energy consumption. Packets, which could not be traced back to a TinyIPFIX Template or Data Record, were treated like empty packets as in Table 6.4. The results of MoteLab runs are shown in Table 6.5 [98]. The first column shows the percentage of packets that successfully arrived at a PC connected to the edge router. The total energy consumption for the established wireless sensor network is shown in the second column. The last column shows the retransmission intervals for Data/Template packages in seconds. [98]

Packets...	...readable [%]	...sent [kB]	Energy [J]
TLV^{48} 5s	99.42	1179.27	66.188
IPFIX 5s / 180s	98.41	827.8	68.246
TinyIPFIX 5s / 180s	99.42	400.7	63.024
TLV^{48} 15s	99.68	390.9	24.491
IPFIX 15s / 480s	98.60	280.9	24.254
TinyIPFIX 15s / 480s	97.23	135.8	23.046

Table 6.5: Packet analysis on MoteLab testbed

Figures in Table 6.5 show a high overall percentage of data packets, which are delivered and readable. TinyIPFIX packets compared to TLV packets have a slightly lower success rate. This difference is caused by some Template Sets being lost during the initial announcement, rendering the following data packets unreadable. TinyIPFIX only transmits around 35% of the amount TLV sends with the above settings. But high savings in the amount of transferred data did not translate to similar savings in energy consumption. Compared to the TLV approach the wireless sensor network performing TinyIPFIX consumes 5% less energy. This difference results in a gap between the amount of energy needed to send packets of different length. [98]
These presented results indicate that more energy savings can be achieved if the TinyIPFIX protocol is expanded with aggregation features. By limiting the amount of meta data more sensor data can be fitted in a data packet leading to less packets being transmitted and, therefore, an overall reduction in energy consumption can be expected. This optimization is analysed in Section 6.2 based on techniques presented in Section 4.2.2.

6.1.4 Comparison to related Transmission Protocols

As described in Section 4.1.1 different efficient transmission protocols for wireless sensor networks exist, such as COAP and sMAP. Both protocols work with the inspiration of RESTful architectures for web service. In this section the developed TinyIPFIX protocol is compared to those protocols. A compact comparison is given in Table 6.6. [98]

One idea of those services is the separation of information in order to improve transmission efficiency. In the case of sensor network this is the separation of measured

values and their meta information. The protocol standard IPFIX for IP networks supports this idea as motivated in Section 3.2. It separates data into Template Records and Data Records, which result in smaller messages.
sMAP offers the opportunity to support different devices in a wireless sensor network. TinyIPFIX also supports this device heterogeneity. Due to the fact that TinyIPFIX is based on IPFIX, which is a monitoring protocol and not intended for sensor data support, the user has to specify and register the hardware by IANA as illustrated in Table 4.1. [98]
Another idea supported by COAP is the complexity reduction of mapping tasks with HTTP and supporting communication protocols such as UDP and TCP. Such a mapping is also done by the developed TinyIPFIX protocol where the mapping happens between Template Records and Data Record (cf. Figure 3.3). As the underlying transmission protocol UDP is supported by the supported BLIP implementation under TinyOS 2.x. TinyIPFIX and COAP have the application field of building automation and machine-to-machine applications in common. [98]
COAP and sMAP prefer compressed data formats, such as XML or JSON. Both data formats can be supported by the TinyIPFIX implementation. Decoding itself takes place at the server. Here decoding of received messages is done with an XML-based meta data source format, which is used as an input by the TinyIPFIX server application (or by the *WSNDriver* in the graphical user interface). In addition, data is in parallel translated to JSON or XML format as requested in order to upload to different visualization tools (e.g. COSM). With help of the developed graphical user interface, received data is stored in a virtual representation, as described in Section 4.4.2, which makes it independent from a data format and supports every format by adding a translation module. [98]

	COAP	sMAP	IPFIX	TinyIPFIX
Web service inspired	**YES**	**YES**	NO	**YES**
Support of information separation	**YES**	**YES**	**YES**	**YES**
Sensor node support	**YES**	**YES**	NO	**YES**
Heterogeneity Support	NO	**YES**	NO	**YES**
UDP support	**YES**	**YES**	**YES**	**YES**
Complexity reduction	**YES**	**YES**	**YES**	**YES**
Building automation support	**YES**	NO	NO	**YES**

Table 6.6: Comparison of features offered by different transmission protocols

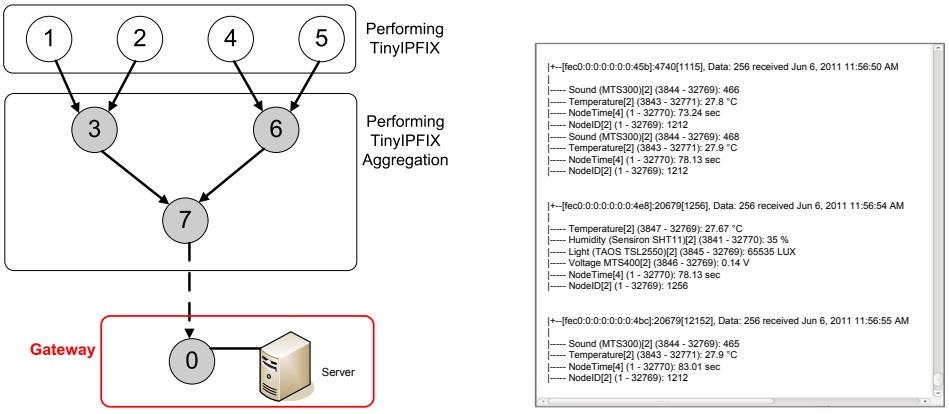
6.2 Evaluation of the TinyIPFIX-Aggregation Framework

In-network aggregation has different impacts on the performance of the implementation presented in this doctoral thesis. Section 4.2.2 describes the implemented aggregation modes in detail, which are called (1) message aggregation and (2) data aggregation.

For the upcoming evaluation on reduction of transmitted messages a simplified testbed is assumed. This testbed is shown in Figure 6.3a with corresponding data packet capture in Figure 6.3b. The testbed consists of several nodes where nodes marked in grey work as aggregators. Furthermore, it is assumed that nodes with IDs 1, 2, 4 and 5 transmit their sensor readings on a regular basis in accordance with the TinyIPFIX protocol described in Section 4.1.2, a template announcing the following data sets to the next template announcement for every node.

In order to analyse the impact of aggregation the following cases are considered:

1. No TinyIPFIX aggregation is performed on any node in the testbed. The grey marked nodes just forward received data down to the gateway without any modification. In this case 12 messages are transmitted in total.
2. TinyIPFIX aggregation is performed on three nodes (grey marked) in the testbed, which only results in seven messages being transmitted in total.



(a) Overview of node arrangement (b) Captured packets by Listener

Figure 6.3: Testbed overview for TinyIPFIX-Aggregation evaluation

6.2.1 Memory Consumption

Due to limited resources of the used hardware IRIS and TelosB (cf. Figure 2.2), memory consumption of the aggregation framework TinyIPFIX-Aggregation becomes important. Hardware choice depends on memory consumption. Memory can be partitioned into variable RAM area and a static programmable ROM area. Table 6.7 shows the required RAM and ROM consumption of the necessary components for the TinyIPFIX-Aggregation framework. [102]

	RAM	ROM
TinyOS functionalities + BLIP	4,766	25,344
UDP Socket	2	296
UDP Shell	288	4,074
TinyIPFIX	559	2,160
TinyIPFIX-Aggregator	404	3,600
Total	6,019	35,474

Table 6.7: Memory usage of TinyIPFIX-Aggregation framework [bytes]

Most of the memory is required for TinyOS (e.g. Boot, LEDs, Timer) and BLIP. Memory required by the applications TinyIPFIX and TinyIPFIX-Aggregator leaves space for further applications and functions. The UDP Socket is needed for the bidirectional communication between user and aggregator performed with inputs in the UDP Shell. The UDP Shell allows 'on the fly' modifications in aggregators. The user can connect to a special aggregator node directly within the wireless sensor network. There the degree of aggregation and the processed aggregation function (MAX, MIN, AVG) can be updated as required in the application.

Finally, the implementation of the TinyIPFIX protocol with its extension TinyIPFIX-Aggregation framework requires a total of 6,019 bytes of RAM and 35,474 bytes of ROM. With those memory requirements a TelosB node is the minimum platform to perform aggregation. In case of message buffer optimization in TinyIPFIX and its applications more resources will be saved. [102]

6.2.2 Impact on Message Amount in the Network

As a result of the described setup the aggregation functionality reduces the number of transmitted messages by 42% [102]. Here a degree of aggregation (DoA) of two messages per aggregate was assumed and the simple message aggregation was performed. If data aggregation functionality is performed additionally, the same result is achieved with additional reduction of the transmitted packet size due to the computation of the aggregate function on the sensors' values. In the positions of the aggregation nodes marked grey, the percent reduction in forwarded messages is calculated per Equation 6.5. [102]

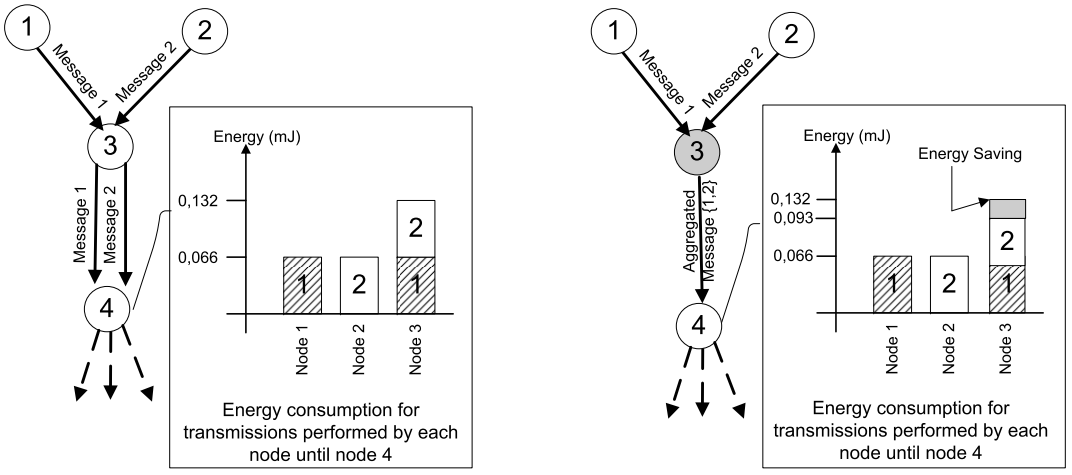
$$MessageAmountReduction[\%] = (1 - \frac{1}{DoA}) * 100. \tag{6.5}$$

In the example shown in Figure 6.3a this results in a reduction of forwarded messages in positions of the aggregation nodes by 50% with ($DoA = 2$) [102]. The reduction of transmitted messages in this example is only related to the reduction of transmitted TinyIPFIX messages. The impact of message reduction on the amount of control messages was not taken into account. However, it is expected to be reduced due to fewer packets being transmitted overall, which leads to less congestion in the network. [102]

6.2.3 Energy Consumption

Parallel to message reduction, energy savings were observed. In order to achieve energy savings on radio transmissions for the aggregator, additional transmission time for the aggregated packets must be compared to transmission time, necessary for forwarding non-aggregated packets. As before $DoA = 2$ is assumed. The trade off between necessary average transmission energy for forwarding two TinyIPFIX packets and necessary average transmission energy for the aggregated packet for TelosB in comparison to just forwarding functionality without any aggregation mode is shown in Figure 6.4. [121, 127]
If data is transmitted in an aggregated format, a saving of 0.039 mJ is achieved compared to a transmission of the same number of packets in individual transmissions over the CC2420 radio of TelosB [102]. Similar results were achieved in bigger testbeds, as shown in Figure 6.1, and in runs on Harvard Sensor Network Testbed. Figure 6.1 visualizes the deployed testbed at our department as described at the beginning of Chapter 6. BLIP offers the opportunity to request a routing tree via an included routing driver shell which visualizes the current status of the system (cf. Figure D.3). [102]

The implemented aggregation techniques lead to increased end-to-end transmission latency. This occurs because a $DoA > 1$ requires the aggregator to wait for more than one incoming packet before being able to perform the chosen aggregation. In the testbeds measurement and transmission intervals were configured to relatively high frequencies in order to test the performance at the breaking point of the system. As a result, no latency could be observed if aggregation in both modes was performed. [102]



(a) Data transmission without aggregation

(b) Data transmission with node 3 performing TinyIPFIX-Aggregation protocol in mode 1

Figure 6.4: Average CC2420 energy consumption per node with TinyIPFIX

6.2.4 Comparison to related Aggregation Techniques

In Section 4.2.1 typical in-network aggregation techniques such as TAG, SIA, and AIDA are introduced. Together with above Sections 6.2.2 to 6.2.3 the established TinyIPFIX aggregation framework is compared to those approaches. Table 6.8 illustrates the comparison in the case of key functionalities. [102]

The original functionality of wireless sensor networks only supports unidirectional communication. In this case, a sensor node collects data and transmits data toward the sink (perhaps over several hops). The sink does not transmit data down to sensor nodes. Thus, it can be understood as an essential functionality. Here aggregation techniques TAG and SIA are an exception. Both were developed exclusively for bidirectional communication with the purpose of aggregation functionality adjustment. In comparison AIDA and the TinyIPFIX-Aggregation Framework can be applied in the same way in application scenarios with unidirectional communication. Bidirectional communication is supported by all four presented techniques. Examples for this communication type are data requests or program updates, e.g. time intervals or key distribution, performed by the gateway. This input should be processed from the sink down to the required data collectors in the network. [102]

Aggregation can be supported in two ways: (1) Message aggregation or (2) Data pre-processing. In the first case several messages are combined in one transmission without any pre-processing. SIA is the only protocol that does not support message aggregation. In contrast only AIDA does not support the second case when data pre-processing is understood as aggregation mechanism. This exception is based on missing information about the aggregated data's application context on the operated layer between the network layer and link layer in AIDA.

Finally, the TinyIPFIX-Aggregation framework supports all four key functionalities (cf. Table 6.8). The framework cooperates with underlying TinyIPFIX protocol, which allows the separation of data values, and meta information combined with a minimum of security support.

Key Functionalities	TAG	AIDA	SIA	TinyIPFIX-Aggregation Framework
Communication Support				
Unidirectional Communication	NO	**YES**	NO	**YES**
Bidirectional Communication	**YES**	**YES**	**YES**	**YES**
Aggregation Support				
Message aggregation	**YES**	**YES**	NO	**YES**
Data preprocessing	**YES**	NO	**YES**	**YES**

Table 6.8: Comparison of features offered by different aggregation approaches

6.3 Evaluation of the TinyDTLS Solution

As described in Section 5.3, the integrated TinyDTLS implementation supports fully authenticated DTLS handshakes between client and server using TinyOS 2.x and BLIP. For the client an OPAL node is essential, because the node has the required resources and an onboard TPM chip. Currently, the node itself has no external sensor board attached; therefore, it does not measure data itself. Therefore, in practice the node forwards received TinyIPFIX messages via UDP using a secured DTLS connection towards the sink. Because of its special location within the network as a clusterhead, it only performs message aggregation. The degree of aggregation can be adjusted to the requirements (e.g. number of nodes in communication range) in order to transmit as much data as possible via the DTLS secured connection.

The client performs the DTLS handshake with an OpenSSL 1.0.0d server implemented on the gateway as illustrated in Figure 5.8. In comparison to the standard OpenSSL 1.0.0d implementation two modifications were essential for the development of TinyDTLS solution in order to support compatibility with the TPM hardware on OPAL. First, the padding for RSA signature verification uses PKCS#1 version 2 instead version 1.5. Second, the client has to sign a SHA1 hash instead of the concatenation of a MD5 and SHA1 hash. For analysis purposes the performed DTLS cipher suite was `TLS-RSA-with-AES-128-CBC-SHA`. The evaluation incorporates aspects of the master thesis by Thomas Kothmayr [81] and the results presented at the conferences SenSys 2011 [127] and SenseApp 2012 [121].

6.3.1 Memory Consumption

For the DTLS implementation a client and a server implementation was chosen. In general, a common PC or a server, which has no limits concerning resources such as power, memory, and computational capacities, represents the server side [10]. The evaluation, therefore, focuses on the client side, which must fit the requirements of a sensor node. In this case, an OPAL node is used, which has 256 kB Program Flash Memory, 52 kB RAM, and an onboard TPM chip used as an additional microcontroller at the moment [33].
For the evaluation of memory consumption on the OPAL node a fully authenticated handshake with 2048-bit RSA keys was performed. This key size was chosen, because it is the worst scenario and keys with a smaller size than 2048 bits are not secure anymore for RSA. It consumes most of the memory space, especially for the required `certificate` and `certificateVerify` messages. The required memory resources are shown in Table 6.9 split into ROM and RAM requirements. [121, 81]

The entries for DTLS purposes in Table 6.9 were split into *DTLS Messages* and *DTLS Network*. The entry *DTLS Messages* summarizes the memory requirements of components needed for generation and parsing of different DTLS handshake messages during the initial key exchange. Additionally, the value includes the code for generating DTLS application messages during bulk data transfer. [121, 81]
In contrast, values represented by *DTLS Network* summarize the components, which are needed for fragmentation, and reassembly of handshake messages as well as the handshake management components. [121, 81]
The category *TinyOS functionalities + BLIP* includes network drivers and routing functions, which are both related to the operating system TinyOS.
In the category *Application*, memory requests for all code running above DTLS and BLIP are included, such as TinyIPFIX. If the TinyIPFIX-Aggregation framework is also installed on the OPAL node, performing message aggregation with $DoA = 2$, only 9,834 bytes for ROM and 1,251 bytes for RAM are additionally required (cf. Table 6.7). [121, 81]

In total, RAM and ROM consumption is below the memory resources of OPAL node (cf. Figure 2.2) and leaves more space for additional functionalities implemented on OPAL for the future. RAM is mostly required for buffering purposes, especially for incoming and outgoing messages. As shown in Figure 4.9, an amount of data must be stored temporarily during the DTLS handshake until a session is concluded. In the current analysis setup of the established wireless sensor network, the send buffer was allocated 1,800 bytes and the receiving buffer with 1,200 bytes. If the chosen scenario varies concerning cryptographically setup and key size, buffers can be reconfigured to smaller sizes. [121, 81]

Component	RAM	ROM
Cryptography	537	10,635
DTLS Messages	1,348	4,204
DTLS Network	3,614	3,104
TPM driver	4,356	6,406
Application	98	2,488
TinyOS functionalities + BLIP	7,284	34,775
Total	17,227	61,612

Table 6.9: Memory consumption of DTLS client implementation [bytes]

Another important analysis question is the scalability of a system, which also requires memory. In the presented case 263 bytes are needed for an active connection. In this case the scalability depends on the required states for each concurrent connection between client and server. Therefore, a connection between states and required keying material for the cipher suite is obvious. As determined in the book 'SSL and TLS Essentials. Securing the Web' by Stephen Thomas, a unique key for generating HMAC and for encrypting messages is needed [90]. In addition, an `Initialization Vector` for the CBC block cipher mode is needed. Keying material is doubled due to the fact that different keying material is needed for incoming and outgoing connections. [121, 81]

In the chosen `TLS-RSA-with-AES-128-CBC-SHA` cipher suite the following keys were established [121, 81]:

- a 16-byte key,
- a 16-byte initialization vector, and
- a 20-byte HMAC key.

In addition to the essential keying material, a session ID with 32 bytes size and a master secret for each connection with 48 bytes is stored. Thus, session resumption is possible for future purposes. Depending on the established connection and required information 53 bytes are spent. Those bytes include i.a. information about sequence numbers, connection state, and selected cipher suite.. In the established wireless sensor networks IP communication is supported and, therefore, an additional 26 bytes are used for the IPv6 address and port information for each node. Some memory space may be neglected due to the number of concurrent connections being small in the final network, because of clustering and OPAL being used as cluster head. [121, 81]

6.3.2 Energy Consumption

As pointed out during hardware description (cf. Section 2.2), energy is one of the most limiting factors in a wireless sensor network. Every application or function added to the default Type-Length-Value approach has the potential to increase the overall resource consumption, which calls for wise energy management within a wireless sensor network.
In Section 6.1.3 energy consumption by TinyIPFIX was analysed in comparison to a common Type-Length-Value approach. It showed that a network performing TinyIPFIX consumes 5% less energy than a Type-Length-Value setup. Additional energy can be saved if aggregation is performed. [20, 98]
In Section 6.2.3 it was shown that the established TinyIPFIX-Aggregation framework can save about 30% of energy if a normal message aggregation is performed; 0.039 mJ for each message aggregation of two messages (cf. Figure 6.4b). This energy saving can be extended if more messages are aggregated. Although energy might be lost if message aggregation is performed, that loss will be relatively small in comparison to the achievements by aggregation. [102]
The OPAL node, therefore, also supports aggregation of mode 1, because no need for data aggregation is assumed currently. For analysis purposes common forward functionality is assumed, which is equal to aggregation with $DoA = 1$. The possible latency caused by waiting for enough buffered packets is omitted.

With the integration of DTLS functionality within the wireless sensor network required energy resources must be analysed again. Each additional function or computation will cause more energy consumption. In this established DTLS handshake scenario the TPM chip on the OPAL node will consume most energy during the handshake phase between client and server. Analysis, therefore, focuses on a TPM chip with a 2048-bit RSA fully authenticated handshake as worst scenario. [121, 81]

$$EnergyConsumption = I * Time * U_{battery} = (U_{probe}/R) * Time * U_{battery} \quad (6.6)$$

	Time [ms]	Energy [mJ]	I=Current [mA]
Computation	35	1.56	30
Radio TX	242	17.42	18
TPM Start	836	174.47	52.2
TPM TWI	688	119.93	43.6
TPM Verify	59	12.22	51.8
TPM Encrypt	39	8.08	51.8
TPM Sign	726	151.51	52.2
Total	2,625	485.19	–

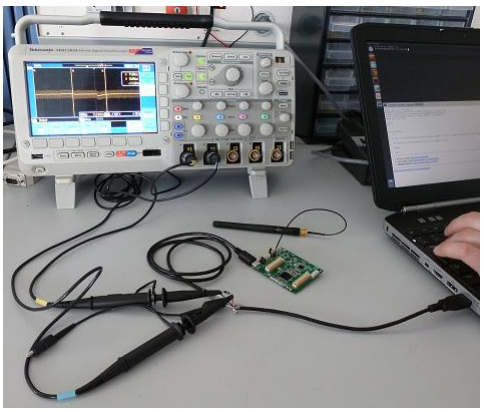
Table 6.10: Energy analysis for TPM chip for fully authenticated handshake

Figure 6.5 shows the recorded energy draw during a fully authenticated DTLS handshake on an OPAL node which is summarized in Table 6.10. If the client receives a request for performing a DTLS handshake, energy consumption significantly rises due to starting the TPM chip. TPM activity can be split into the following five parts where the required power is calculated with Equation 6.6 assuming a battery voltage $U_{battery} = 3,998V$ [121, 81, 127]:

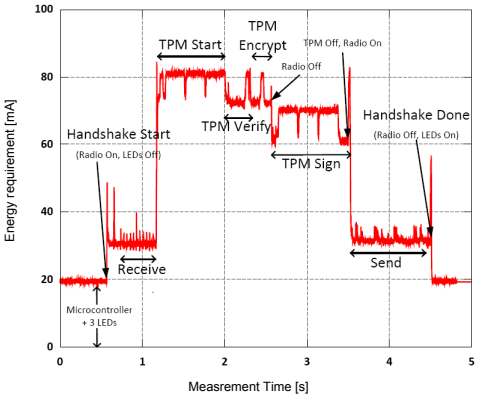
1. ***TPM Start*:** In this phase various self tests are performed which consume 174.47 mJ. Those tests check for tampering and unauthorized commands.
2. ***TPM TWI*:** In this phase 119.93 mJ energy is spent in order to pass data to the TPM and receive data from the TPM via the TWI bus.
3. ***TPM Verify*:** Here the verification operation is performed by the TPM which consumes 12.22 mJ. A server certificate is presented, and must be verified by the TPM using the stored key of the certificate authority.
4. ***TPM Encrypt*:** Here the included *Nonce* in the server message is encrypted with the server's public key requiring 8.08 mJ, followed by data transfer via the TWI bus.
5. ***TPM Sign*:** This part is optional and only performed if the node is expected to authenticate itself during the handshake. The high energy consumption of 151.51 mJ during this phase is a result of using the RSA private key instead of RSA public key.

As soon as the handshake is performed a secure connection is established and the TPM is switched off, which is marked by a lowering of energy consumption down to sending level. Sending and receiving levels are equally high and are a direct result of data exchange between client and server. In addition to the previously mentioned energy consumption, 180.71 mJ must be added for the microcontroller activity and 270.44 mJ for the active radio. Finally, 936.34 mJ are required for a fully authenticated handshake. In order to save more energy, power saving techniques for the microcontroller (e.g. idle mode) and, especially, for the radio (e.g. sleeping modes) must be integrated in the future. [121, 81]

In comparison to Table 6.10, the required energy resources for the server site authenticated handshake is shown in Table 6.11. Most of the energy saving is achieved by drop of *TPM Sign* and the reduction of the value *TPM TWI*. In addition to the



(a) Testbed setup



(b) Energy draw

Figure 6.5: Energy draw for a fully authenticated DTLS handshake on OPAL node

283.86 mJ energy consumed, a total of 103.23 mJ must be added for the microcontroller activity and 160.34 mJ for the active radio. Finally, for a server authenticated handshake 547.43 mJ are required, which is 41.54% less than for a fully authenticated handshake. [121, 81]

An ordinary OPAL either receives its energy via an USB connection to a PC or server, which means no energy limitation exists, or via a battery pack with three AA batteries. This battery pack contains approximately 45,360 J in total. Assuming the OPAL node performs 20 DTLS re-keying operations per day consuming about 485.19 mJ per cycle based on the aforementioned energy analysis in this section, lifetime of a node would be 12.8 years. If energy saving techniques are integrated in the current implementation, lifetime increases. Lifetime of the OPAL is influenced in a negative way if message aggregation is performed, which requires 0.039 mJ per message aggregation with $DoA = 2$. But as an advantage the node can reduce the number of forwarded messages. [121, 81]

	Time [ms]	Energy [mJ]	I=Current [mA]
Computation	33	1.50	30
Radio TX	70	5.04	18
TPM Start	836	174.47	52.2
TPM TWI	476	82.97	43.6
TPM Verify	56	11.6	51.8
TPM Encrypt	40	8.28	51.8
TPM Sign	–	–	–
Total	1,511	283.86	–

Table 6.11: Energy analysis for TPM chip for server authenticated handshake

6.3.3 Comparison to related Security Mechanisms

As motivated in Section 3.3 wireless sensor networks can be attacked on each layer, which shows the possible vulnerability. Today's used security standards in IP networks are to bulky for resources of sensor devices. They would exhaust the following resources: memory, computational capacity and power very soon. Therefore, a long lifetime for a sensor network cannot be supported. Research focuses

on key management strategies in order to secure communication between different entities of a wireless sensor network. Depending on supported cryptographic functions many resources are required. Currently, the best and most secure solution is it to support a combination of software and hardware security as motivated in Section 3.3.3.

In order to provide security in the system, a third node platform called OPAL was integrated. As characterized, the OPAL node includes a Trusted Platform Module chip. This technology is known from today's notebooks and offers the highest security available at the moment. TPM builds a chain of trust when a system boots and, based on the hardware/software configuration, a storage root key is derivated and stored in the secure memory of the TPM. Based on this key all other keys are derived. Messages secured with this key can only be encrypted if the system itself is still the same due to the previously mentioned dependency between TPM, hardware, and software configuration.
The developed TinyDTLS solution is a standard based security architecture with a two way authentication for wireless sensor networks [121]. Authentication is performed during a fully authenticated DTLS handshake and based on an exchange of X.509 certificates containing RSA keys, which we have implemented. In Section 3.3 different security requirements were introduced where the TinyDTLS solution provides message integrity, confidentiality, and authenticity. As proven in the previous section of the TinyDTLS evaluation, the solution copes with the limited resources of sensor devices and, therefore, is a feasible security solution.

6.4 Evaluation of the Graphical User Interface

As indicated in section 5.4 each existing graphical user interface is pegged to a special application, such as simulation or visualization tasks. Therefore, the implemented graphical user interface is based on the requirements for the presented home application, as introduced in Sections 2.5 and 5. In the presented application the operating system TinyOS 2.1.1 and the transmission protocol TinyIPFIX with its extensions described in Section 4 are supported. The implementation of the graphical user interface is modular. This fact allows the adaptation to other application settings with less manual input and also supports the inclusion of extensions, such as more exporting clients. If hardware of new vendors is included in the wireless sensor network, the XML configuration file must be updated to provide the conversion procedure to the *WSNDriver* (cf. Figure 4.11). [133]

The graphical user interface, as described in reference [133], is implemented on the server, which has unlimited resources, rendering energy consumption and memory evaluation unnecessary. Therefore, the presented evaluation is a proof of operability of the implemented graphical user interface.

6.4.1 Configuration of the Network Components

The established interface supports the programming of the required base station as well as the measurement devices. For the measurement devices the user is able to choose between different pre-installed setups covering most of the common hardware specifications (cf. Figure D.1). The pre-installed setups include platform type, sensor board type, TinyOS options, and node functionality. The TinyOS option BLIP is activated on default. With this degree of configuration, the user is able to configure each node in the wireless sensor network individually. In order to ensure unique identities, the user has to choose an individual ID for each node. The

programming interface for the base station is similar. The programming interface also allows new node programming during runtime of the wireless system in parallel as long as the new node is not from the same type as the currently attached base station. This causes a drawback due to the naming by the vendor: when the node is attached to another USB port, the internal software having the same vendor name as the base station causes an error. The only possibility to solve this problem is a request to the vendor to use unique numbers instead of platform types to recognize the newly attached hardware by the underlying operating system (e.g. Linux). [133]

6.4.2 Graphical Feedback of Network Status

As described in Section 5.4.2 in detail, the graphical feedback of the activated wireless sensor network is based on two functions: First, the routing driver shell supported by the BLIP implementation allows a screenshot of the current node links visualized in a tree structure. Second, all information about the individual nodes itself is extracted from the received data record based on the transmission protocol TinyIPFIX. Received data is encoded with help of Template Records (cf. Section 4.1.2) and in the next step abstracted into a virtual representation of the network. All this work is performed by the module *WSNDriver*. The node representation takes part when the first Data Record was successfully decoded. Then the virtual representation following the UML draft in Figure D.2 starts and results in an information visualization in the graphical user interface under menu `WSN Nodes`. [133]

Figure D.3 shows an example of the visualization event: the right part is based on the module *WSNDriver* and the left part on the TinyOS function `BLIP`. The individual node information on the right part is updated periodically when the routing tree updates itself. The user can initiate this update procedure as well. In this case the user can either refresh the browser or change to menu `Hardware` in submenu `TinyOS` and click on the button `rebuild topology` under option `Tools`. The left part illustrates the current routing tree behind the network. If sensor nodes are in range of the gateway, they do not need to route their packets over an intermediate node. This option is chosen if the distance is invalid or an aggregator is part of the network (cf. Figure D.3c). In order to find a special node more easily in the complex routing tree case, the user needs to click on the node in the right hand portion of the presentation. Next the node with all its incoming and outgoing connections is highlighted in the routing tree (Figure D.3c). [133]

6.4.3 Data Export/Import Functionality

In order to prove the established data export/import functionality in the graphical user interface, the following office scenarios were established:

- Experiment 1: Three IRIS nodes and one TelosB recording temperature, humidity, brightness, acoustic, and voltage consumption.

- Experiment 2: Three IRIS nodes and two TelosB nodes where one TelosB supports message aggregation functionality.

6.4.3.1 Experiment 1: Data Collectors of different Vendors

During this dissertation several experiments with slightly different setups and network sizes were performed in order to verify a stable operation of the graphical user interface. The node deployment shown in Figure D.7a is assumed to verify the support of different vendors. The first experiment took place during five hours on a sunny office day on May 8th, 2012. In this experiment the following four sensor nodes were included:

- IRIS from Crossbow Inc.
  - NodeID 1104: Node located next to the window, and activated MTS300.
  - NodeID 1101: Node located on the table next to the window, and activated MTS400.
  - NodeID 1106: Node located on the table next to the office door, and activated MTS300.
- TelosB from Advantic
  - NodeID 2250: Node located on the same table as node 1101, performing only data collection with all onboard sensors activated.

The configuration of the IRIS nodes was the following: IRIS nodes request sensors to collect environmental data every five seconds. They transmit their Template Record periodically every 30 seconds in order to ensure decoding of Data Records. For this test run collected data is uploaded to COSM every 30 seconds. TelosB node has an altered setup. The data-recording interval was set to 60 seconds and the remaining conditions were the same as in the first experiment.
The experiment started at 9 a.m. and had a duration of four hours and the sensor nodes were started with random delay. After all sensor nodes had announced their Templates and the first Data Records were received, the routing tree was established. The final routing tree is shown in Figure D.8. The figure shows the IRIS node with ID 1106 works as an intermediate and forwards packets of nodes 1104 and 2250 towards the base station. Node 1106 also collects data itself and is displayed in the right-hand portion of the figure in the highlighted box. The link between node 1106 and 1104 is highlighted, because a packet was transmitted over this link during the screen shot. The boxes in the right-hand portion of the figure also present node information extracted from the TinyIPFIX packets received at the gateway. A capture of the incoming packets is shown in Figure D.7b, as provided by the Listener tool of TinyOS. This capture displays both - packet type (e.g. Template or Data) and the COSM upload command.

Figure D.11 illustrates the COSM visualization in the implemented graphical user interface. It shows that the user has chosen different visualization intervals ranging from five minutes up to three hours and that all sensor node information was exported to COSM. Even though not all values are essential, the importing result also includes all values some (e.g. NodeID and NodeTime). Analysis of the records is divided into the following parts: humidity analysis, temperature analysis, acoustic analysis, and voltage analysis.

6.4.3.2 Experiment 2: Aggregation Performance

The second experiment is similar to the first experiment. An overview is shown in the implementation chapter of this doctoral thesis in Figure D.4. In this experiment not all nodes support data collection functionality. Therefore, one TelosB node (ID 2222) supports message aggregation. As can be seen in the record of the Listener tool, the node aggregated TinyIPFIX messages from nodes 1101 and 1103 (cf. Figure D.4b). Incoming packets at the gateway are encrypted and split into their components. In the case of aggregator node 2222, the received aggregate was split into two parts corresponding to their sources. From this point on, the virtual presentation of the network steps in, as describe in Section 4.4.2. Thus, the final representation in COSM allows the user to visualize each data collector as shown in Figure D.6. In this COSM virtualization the aggregator 2222 did not occur, because it did not collect data by itself.

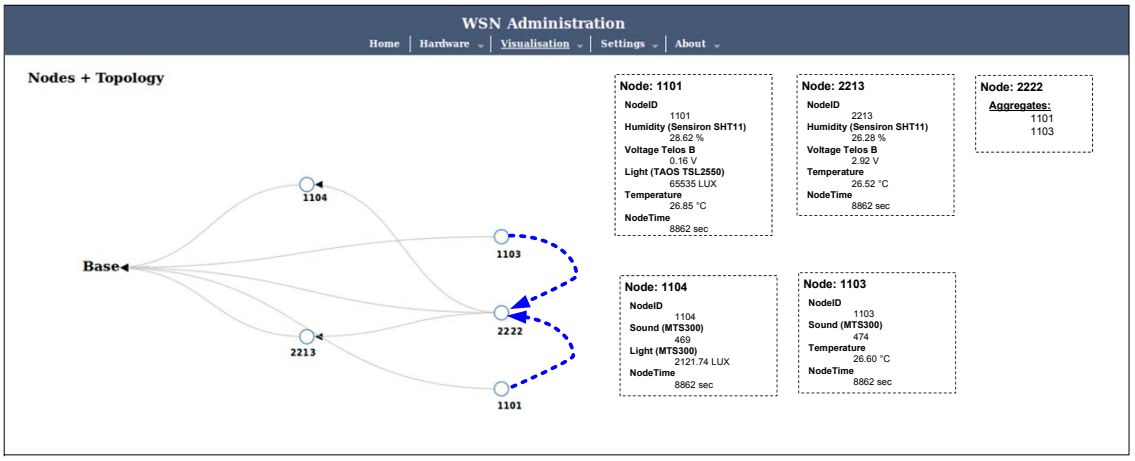


Figure 6.6: Experiment 2 - Established communication links

Figure 6.6 shows the underlying network topology. Due to a code update all established communication links are now visible in the menu `Topology`. Grey drawn arrows represent the established communication links by BLIP as described in Section 5.4.2. Depending on the network status nodes can have a different number of communication links. In any event, they have a link to the sink, which performs in this case the program `IP Basestation`[11]. Here nodes 1104 and 2213 are data collectors and work as intermediates in the network. They forward incoming packets from node 2222.
When an aggregator (here: node 2222) enters the network, the aggregator performs the dynamic neighbor discovery strategy as introduced in Section 4.2.2.2. The result is a binding of nodes corresponding to the predefined degree of aggregation. Here data collectors 1101 and 1103 are bound to the aggregator. Finally, preferred communication links of the data collectors are updated, which is illustrated with blue dashed bold arrows. The before established communication links by BLIP still exist, but are not used anymore as long as the nodes are bound to the aggregator. If the

[11] 'IPBaseStation is a modification of the generic BaseStation which ships with tinyOS-2.x. It alters the serial protocol to pass 802.15.4 frames instead of Serial.h packets. It also adds an out-of-band configuration protocol which allows a driver running over the serial port to reboot the mote, and to set the device address, channel, and retransmission parameters. These changes are useful when one wishes to use a mote attached to a computer as an 802.15.4 interface rather then an actual mote. The actual queuing logic for copying packets is mostly unchanged, and it continues to make use of serial ACKs.' [25]

binding is removed (e.g. node deletion, execution of command `rebuild` by BLIP), the original communication links may be reactivated again.

6.5 Compliance of a Cyber-Physical System

As mentioned in Section 4.1.2, the design space for an application protocol designed to achieve tight integration of wireless sensor networks into a cyber-physical system consists of four areas [12]: (1) standardization, (2) resource efficiency, (3) flexibility, and (4) usability. In this section it is proven that the developed application protocol TinyIPFIX and its extensions fulfill these four criteria.

6.5.1 Standardization

Sensor measurement data is identified by Type ID and the Enterprise Number (EID). As described in Section 3.2 each measurement of a sensor should be assigned a globally unique combination of Enterprise and Type ID. For example, the Sensirion SHT11 Temperature and Humidity sensor would be assigned two different combinations:

1. Temperature channel: $EnterpriseID = 3841$ and $TypeID = 33025$
2. Humidity channel: $EnterpriseID = 3841$ and $TypeID = 33026$

A public repository would allow an application to receive a template with an Enterprise and Type ID combination, which has not been seen before, and obtain the semantics of that measurement from the Internet. Semantic information includes the kind of data (temperature), data type (16-bit integer), how to convert to a sensor independent format (formula to convert to $°C$), and any other required information. This allows the flexible deployment of motes with different sensors. For example, if an existing deployment, which is using Sensirion SHT11 sensors, was augmented with several new nodes that use a Sensirion SHT15 instead, there is no need for extensive reconfiguration within the wireless sensor network itself or the converter application if both use TinyIPFIX to send their measurement data. Nodes can simply transmit raw measurements without having to convert them into another format via potentially complex formulas. The receiving application on a gateway can convert both values to scientific units and combine measurements based on semantic information obtained from the repository.

6.5.2 Resource Efficiency

Aspects considered by resource efficiency focus on memory and energy resources. As assumed in the beginning of Chapter 6 the application protocol TinyIPFIX and its extensions were implemented on top of the BLIP IPv6 and UDP implementation, which is part of TinyOS 2.1.1.
As described in Section 6.1.1 each component is built upon the other. Corresponding memory consumptions are shown in Table 6.2 for TinyIPFIX. The results are shown in Table 6.7 including the extension TinyIPFIX-Aggregation. For the DTLS client implementation memory consumption is shown in Table 6.9. Due to constrained memory resources of the IRIS platform, this node can only perform TinyIPFIX without any extension. Platforms with more resources such as TelosB or OPAL can perform the implemented extensions.

In Section 2.2 the hardware used was characterized as follows: platforms IRIS and TeloB receive their energy from a battery pack with two AA batteries. In contrast the OPAL node can be powered either by a battery pack of three AA batteries or via a micro USB connection. The latter was used in the presented setups, which results in a non-exhaustible energy resource.
As shown in Table 6.4, the transmission of an average TinyIPFIX Data packet costs $727\mu J$, as compared to $780\mu J$ required for a TinyIPFIX Template packet. Due to the construction of the underlying TinyIPFIX protocol, the Template packet is transmitted only once directly after the sensor node's booting. Thus, this is a one-time energy cost. If UDP is used, this packet must be repeated periodically. The interval depends on the stability of the network infrastructure. This means, if the network is stable and no route changes are expected, the interval can be extended resulting in less transmission. It also can be extended if it is assumed that all intermediate nodes, especially the sink, store all Templates for decoding purposes. This storage task is also a question of memory resources. Thus, it is not a good idea solution for constrained platforms like IRIS.

Concerning aggregation support, Section 6.5.1 noted that the IRIS platform is too constrained to support this feature. Thus, TelosB and OPAL can only support this extension. As described in Section 6.2.3 the support of aggregation saves 30% on TelosB compared to common transmission if $DoA = 2$ is assumed. If the degree of aggregation is higher, more energy can be saved. The same energy consumption is gained at the OPAL platform.

Most of the energy is consumed on the OPAL platform if the node's TPM chip is activated. This must be done in order to establish a secure connection as depicted in Sections 6.3.2. The establishment of a connection requires $485.19\mu J$. With a battery pack and 20 re-keying operations per day a lifetime of 12.8 years is possible.

As the analysis of the constrained platform IRIS shows, the established protocol TinyIPFIX can be supported. Further extensions can be used if platforms have more resources, such as TelosB or OPAL.

6.5.3 Flexibility

The developed TinyIPFIX implementation is based on TinyOS 2.1.1 and has been tested successfully for TelosB and IRIS motes as described in the previous sections of Chapter 6. With little modifications the implementation can also support other hardware platforms featuring IEEE 802.15.4 radios (e.g. IMote).
Due to the intuitive structure of the Data and Template Set, each user can easily implement their own Set. The programmer can decide what information is to be transmitted in the packets and, therefore, modify the Set as needed. For example, in order to add a new temperature sensor to the sensor node's programming, the following changes are needed:

1. The `IPFIXDataSampler` interface must be modified, and instantiated with the desired Type ID and Enterprise ID.
   For example: `components new IPFIXDataSampler16C(uint16_t type_id, uint32_t enterprise_id) as Temp;`

2. The fitting sensor must be instantiated, and wired to the sensor interface of the IPFIXDataSampler.
   For example: `Temp.Sensor -> Sht11.Temperature;`

3. Finally all `IPFIXDataSamplers` must be wired to the `Sampler` interface of the main application.
   For example: `App.Sampler -> Temp;`

The program can now automatically generate templates for all connected sensors and obtain their measurement data, which is then automatically encapsulated in a format that complies to the previously generated template. In order to allow a more fine-grained control over the creation of a TinyIPFIX packet, the programmer can also use methods provided by the TinyIPFIX implementation such as the following [20]:

- `tinyIPFIX.start_template_record(uint8_t buffer_no, uint16_t template_id)` to start handcrafting a Template Record or
- `tinyIPFIX.start_data_record(uint8_t buffer_no)` for writing multiple Data Records into a single packet.

The consumed ROM and RAM space of the implementation is shown in Table 6.7. Reported figures make the protocol viable for use on constrained hardware. Due to the modular structure of TinyOS, different parts of the implemented protocol can be excluded for very limited hardware and included on nodes with more resources in a heterogeneous network as shown in Figure 6.1. This advantage makes the TinyIPFIX protocol attractive for all common node platforms.

6.5.4 Usability

In order to evaluate the ability of usability, different application scenarios are assumed working with different hardware vendors (cf. Sections 6.4.3.1 to 6.4.3.2). In Chapter 6 the evaluation is based on an office scenario as illustrated in Figure 6.1. This scenario was driven by the AutHoNe project developed at the department 'Network Architectures and Services' at the Technische Universität München [13]. In this project a wireless sensor network was integrated into an existing smart home infrastructure supporting IPv6. A sample Wireshark record is shown in Figure 5.9 where some TinyIPFIX packets are transported via a DTLS secured connection to the gateway and others via an insecure UDP connection. Figure 5.1 illustrates the message flow within the AutHoNe infrastructure.
For successfully decoding an XML file is used as input at the autonomic home infrastructure. If TinyIPFIX data is encoded, it is passed to the AutHoNe application in concrete to the Knowledge Agent, which makes the data available for other components in the AutHoNe infrastructure (e.g. Autonomic Manager). For example, the Autonomic Manager can access the room temperature value in order to assess the situation and manage the heating system in the room accordingly.

6.6 Summary and Findings

For evaluation purposes in this chapter, a building scenario and the usage of hardware supporting TinyOS was assumed. The developed protocols and solutions of this dissertation were analysed concerning memory and energy consumption and compared to related approaches in wireless sensor networks. It was found out that due to constrained hardware used in wireless sensor networks not every protocol can be performed in its default configuration or by every sensor node.

The developed TinyIPFIX protocol can only be performed on constrained hardware due to the integration of compression functionality for the additional overhead caused by IP headers. With this compression the overhead could be reduced by 85%, which made a transfer to sensor nodes possible. The implemented solution requires a minimum of 7 bytes RAM and 30 bytes ROM. This is not a problem with the used hardware in this dissertation. Concerning energy requests the performance of TinyIPFIX protocol scales as well as the traditional Type-Length Value approach whereas data and meta information is transmitted in the same messages. The TinyIPFIX protocol required 727 μJ for a TinyIPFIX Data packet and 780 μJ for a TinyIPFIX Template packet. Due to the results it was proven that the TinyIPFIX protocol can be used on all node platforms supporting TinyOS. [20, 98]

In-network aggregation is important for wireless sensor networks in order to reduce traffic within the network itself and to optimize efficiency. Therefore, a Tiny-Aggregation protocol was integrated in this dissertation supporting message and data aggregation. This functionality required additional 404 bytes of RAM and 36 bytes of ROM. Concerning energy consumption it was shown that performing aggregation saved 0.039 mJ (respectively 30%) compared to transmission of the same number of packets in individual transmissions over the CC2420 radio used in sensor nodes. In general, such additional functionality is performed on special locations within the network and requires more computational capacity. This dissertation found out that only the TelosB or OPAL platform used had enough resources to support this functionality. [102]

In order to provide end-to-end security this dissertation faced the challenge to bring DTLS on constrained hardware. As described in Section 4.3 a strong two-way authentication can be performed by sensor nodes including a TPM chip, such as OPAL, or a weaker authentication if they do not include the chip. In order to support DTLS, sensor nodes must offer 18 bytes of RAM and 62 bytes of ROM and, therefore, not every hardware can perform this functionality. Concerning energy consumption the TPM chip consumes most of the energy when performing the authentication. It required 485.19 μJ in total when performing a 2048-bit RSA fully authenticated handshake. When assuming a power resource with three AA batteries offering approximately 45,360 J and assuming 20 re-keying operations a day, lifetime of an OPAL node would be 12.8 years. [81, 121]

In addition, this chapter evaluated the implemented graphical user interface by a proof of working. It was not analysed concerning memory or energy requirements, because it is located on a server without such regulations. The previously mentioned tests on different settings were directly visualized by the graphical user interface and exported to the online analysis tool COSM in order to give the user a visual feedback. The graphical user interface worked well even during long term experiments and displayed live stream of data and network configuration / status appropriate. [133]

Regarding the mentioned research questions on efficency the following contribution has been made in this chapter:

(E3) Can all sensor platforms perform TinyIPFIX and its extensions (compression, aggregation), as well as TinyDTLS?

No. Concerning memory consumption it was proven in Section 6.1.1 that TinyIPFIX performing aggressive header compression requires 4,841-6,889 bytes RAM and 28,810-29,020 bytes ROM depending on TinyIPFIX packet size. Assuming a packet size of 102 bytes as specified by the RF transceiver CC2420, the minimum request of memory is 4,784 bytes RAM and 26,008 bytes ROM. This means, the sensor platforms IRIS, TelosB and OPAL, which are used in this dissertation, can all perform TinyIPFIX. In the case of TinyIPFIX-Aggregation additional 1,253 bytes RAM and 10,130 bytes ROM are required, which means that only TelosB and OPAL have enough resources to perform aggregation (cf. Section 6.2.1). Concerning energy consumption the used hardware in this dissertation is well dimensioned in order to perform TinyIPFIX and its extensions. The OPAL platform is the best candidate to perform DTLS, because it offers a strong two-way authentication due to the included TPM chip.
The whole implementation - TinyIPFIX with its extensions and TinyDTLS support - was tested on different network sizes and with different settings and experimental durations. It scaled well and proved its flexible adaptation. Due to its modular structure, implementation can be extended with fewer overheads and adapted to other sensor platforms using TinyOS.

7. Conclusion

This dissertation faced the challenge to integrate a secure and efficient data transmission solution based on standards used in IP networks into wireless sensor networks, which are part of cyber-physical systems [12]. As a representative for cyber-physical systems an intelligent building control system was evaluated, which was deployed in order to achieve a high level of comfort for residents and a high overall energy efficiency of the building. Wireless sensor networks, as part of a smart meter infrastructure, provide the required level of data quality with temporally and spatially fine-grained measurements.
In the solution presented in this doctoral thesis, sensor data (e.g. temperature, brightness, acoustic, humidity) is sent to a gateway, which offers different possibilities for analysis in order to manage the environmental conditions of the building based on the habitants' preferences. The analysis' result can either be directly applied to different entities controlling automatic systems such as those developed for the AutHoNe project. Another possibility is the export of the data to analysis tools, which allow an import of data into a visualization tool in order to display the current status and to help optimizing the carbon footprint due to real-time feedback for the habitants influencing their behavior.

In this doctoral thesis it was pointed out throughout Chapters 2 and 3 what challenges are faced regarding wireless sensor networks and why new protocols are required. Due to the similarity between wireless sensor networks and IP networks, it was obvious to use established IP protocols for wireless sensor networks. The challenge for the protocol transfer was the resource limitation of the sensor hardware in memory, energy and computational capacity. IP protocols are too bulky for the sensor resources (e.g. cryptographically operations, message size), so that they must be modified and (perhaps) customized in their functionalities in order to fulfill the new requirements.
It was briefly described in Chapter 4 what protocols in the fields of IP communication, data transmission, and security were interesting for sensor network applications; followed by performed modifications in order to adapt those standards, such as IPFIX or DTLS, to the resource constrained environment of sensor networks (cf. Chapter 5). In order to facilitate the efficient transfer of sensor data through a heterogeneous wireless sensor network TinyIPFIX was developed due to it's minimal configuration based on the standard IPFIX. TinyIPFIX is a versatile and

light-weight application level protocol for data exchange that can be adapted to different vendors and sensor configurations for heterogeneity support in the system. The performance of the TinyIPFIX implementation and its extensions (e.g. header compression techniques, aggregation) was evaluated regarding the transmission efficiency, system level energy consumption, and memory requirements.
The rate of successfully transmitted measurement was similar to conventional approaches even though the separation of meta data and measurement data into Template and Data Records in different packets increases the risk of unreadable data. The savings in the amount of transmitted data did not translate directly into energy savings on the system level, although a reduction of about 5% could be achieved.

In addition, energy savings of up to 30% were achieved with the integration of aggregation functionality to the system - called TinyIPFIX-Aggregation framework (cf. Sections 5.2 and 6.2). If message aggregation is used, 0,039 mJ can be saved per aggregated transmission. If data aggregation is performed, more energy can be saved due to reduced message sizes within the network. Aggregation functionality can also be changed during the system run on the fly. Due to the limitations of resources and the required memory of the extension - TinyAggregation -, it turned out that not every hardware could perform TinyIPFIX and TinyAggregation at the same time. Thus, sensor nodes with more resources had to be integrated into the wireless sensor network at exquisite locations in order to perform both functionalities. This constraint resulted in a functional distinction of the sensor hardware into data collectors only performing TinyIPFIX and into nodes with additional functionalities (e.g. aggregation).

Due to the chosen application scenario, collected data can include sensitive data, which raises security issues. Currently available security protocols supporting encryption call for more resources due to computation and key management requests which emphasizes dependency between hardware resources and user requests. For this thesis TinyDTLS was implemented in order to face this challenge. It allows the establishment of a secure communication channel between different components (cf. Section 5.3). The implemented TinyDTLS offers one of the highest possible security options, because the hardware uses the functionality of a Trusted Platform Module to perform the DTLS handshake in order to establish a secure communication channel.

Another part presented in this dissertation was the description of how a wireless sensor network must be configured in order to deploy it in a building scenario as a representative for a cyber-physical system. It was accompanied by a proof of functionality via real-life test runs. TinyIPFIX and its extensions are a suitable choice for the cyber-physical systems presented, due to the flexibility for different hardware, suitability to constrained resources, extendibility and scalability of different network sizes (cf. Section 6.5).

In addition, a comfortable graphical user interface was integrated in order to raise comfort for users to configure such a wireless sensor network, to receive visual feedback of the network, and to allow data analysis using online and offline tools (cf. Section 5.4). The established graphical user interface is dependent of the application. Currently, it supports configuration tasks for sensor platforms IRIS, TelosB, and OPAL under the operating system TinyOS 2.1.1, graphical visualization and data analysis following TinyIPFIX requirements. The whole system was implemented in a modular structure, which allows extension integration in order to sup-

port additional functionalities (e.g. hardware, operating system, visualization tools) in the future.

During this dissertation different research questions influenced the design of the presented solution - TinyIPFIX with its extensions. In Section 1.2 a number of research questions were listed, which were answered during this dissertation and are now mentioned again in order to summarize the research impact of this dissertation.

Regarding the mentioned research questions on efficiency the following contributions have been made:

(E1) Is the IP Flow Information Export (IPFIX) protocol a viable solution for transmission of sensor data in wireless sensor networks?

> Yes. As assumed in Chapter 3 IP communication is supported by constrained hardware today and it became interesting to transfer standardized protocols to constrained hardware as used in wireless sensor networks. The IPFIX protocol was introduced in Section 3.2. The message structure of the IPFIX protocol is very interesting for wireless sensor networks, because it separates meta information and data in different messages. It works over UDP, which is the preferred transport protocol for sensor networks. IPFIX is flexible concerning its message structure and reduces retransmissions of known information (e.g. meta information).
> A drawback is the additional overhead of 20 bytes caused by the IPFIX message and Set headers that reduces the free space in the payload of each message. But this drawback was solved as described in Section 4.1.3 by introducing header compression techniques. Three different compression techniques were developed that had a pre-header in common: defensive compression, modified defensive compression, and aggressive compression. This pre-header specifies the field sizes of IPFIX message and Set header. The pre-header in the aggressive compression techniques has a minimum size of three bytes which means a compression of 85% (cf. Figure 5.4).

(E2) Is it possible to combine data pre-processing techniques (e.g. aggregation) with the IPFIX protocol within the network?

> Yes. Depending on the chosen application scenario for the deployed wireless sensor network it might be interesting to pre-process data within the network itself. This work incorporates the reduction of network traffic throughout the whole network. Section 4.2 introduced existing aggregation techniques in wireless sensor networks such as TAG, AIDA and SIA. Those protocols are very specified and were developed for specific applications.
> In the case of IPFIX a general pre-processing technique was chosen that only aggregates data. Therefore, the TinyIPFIX-Aggregation framework was developed offering message and data aggregation. For this extension support new templates had to be specified for IPFIX and a neighbor discover algorithm had to be integrated. Additionally, the user can manually log on the aggregator node in order to modify the performed aggregation within the network (e.g. degree of aggregation, performed aggregation function from AVG to MAX).

(E3) Can all sensor platforms perform TinyIPFIX and its extensions (compression, aggregation), as well as TinyDTLS?

No. Concerning memory consumption it was proven in Section 6.1.1 that TinyIPFIX performing aggressive header compression requires 4,841-6,889 bytes RAM and 28,810-29,020 bytes ROM depending on TinyIPFIX packet size. Assuming a packet size of 102 bytes as specified by the RF transceiver CC2420, the minimum request of memory is 4,784 bytes RAM and 26,008 bytes ROM. This means, the sensor platforms IRIS, TelosB and OPAL, which are used in this dissertation, can all perform TinyIPFIX. In the case of TinyIPFIX-Aggregation additional 1,253 bytes RAM and 10,130 bytes ROM are required, which means that only TelosB and OPAL have enough resources to perform aggregation (cf. Section 6.2.1). Concerning energy consumption the used hardware in this dissertation is well dimensioned in order to perform TinyIPFIX and its extensions. The OPAL platform is the best candidate to perform DTLS, because it offers a strong two-way authentication due to the included TPM chip.
The whole implementation - TinyIPFIX with its extensions and TinyDTLS support - was tested on different network sizes and with different settings and experimental durations. It scaled well and proved its flexible adaptation. Due to its modular structure, implementation can be extended with fewer overheads and adapted to other sensor platforms using TinyOS.

Regarding the mentioned research questions on security the following contributions have been made:

(S1) Is it possible to secure data transmission in wireless sensor networks with known standards from IP networks?

Yes. Concerning security Section 3.3 gave a brief overview of solutions used in IP networks in order to secure data transmissions within a network. In order to answer research question S1, selected security solutions focused on security protocols (cryptographic functions, public key infrastructures), (D)TLS protocol, and trusted hardware component using a trusted platform module. Throughout this section it became obvious that not every solution is interesting and contributing to wireless sensor networks, because of limited resources especially in memory and computational capacities. But the given overview influenced the design decision for the realized security solution in this dissertation and was presented in Section 4.3. First, different cryptographic methods (e.g. RSA, AES, TinyPK, TinySec, Tiny-ECC) were characterized, followed by symmetric key management solutions (e.g. PIKE, solutions by Echenauer and Gligor), and end-to-end security solutions (e.g. IPsec, Sizzle, SSNAIL, Tiny-3-TLS). All introduced protocols request different resources of the sensor nodes, and, therefore, nodes can be exhausted quickly. Due to the comparison of the different approaches, it was pointed out that a standard-based approach across all communication layers scaled best for heterogeneous networks. Depending on the technology development of embedded platforms (e.g. OPAL including TPM chip) more security functions can be supported that allow authentication of the participating parties.

(S2) Can DTLS be performed on strongly constrained hardware as used in wireless sensor networks?

> Yes. DTLS can be performed on constrained hardware. Section 4.3.2 described the required modifications for a DTLS transfer on wireless sensor networks. In this dissertation a new platform, called OPAL, was integrated in the wireless sensor network. This platform includes a Trusted Platform Module that allowed working with certificates. Therewith, a strong tow-way authentication handshake can be adapted to the communication participants in the wireless sensor network. The included TPM chip on the platform allowed to save the RSA private key to be stored in a tamper-proof location and prohibit to pass it outside. As a consequence all operations using this RSA key had to be performed in this chip. All these characteristics affect the attacker's work. The authentication, which uses certificates during the handshake phase in a standard DTLS handshake (cf. Section 3.3.2), offers the same properties as authentication performed via the conventional TLS protocol in the Internet. In the case of a platform without a TPM chip, it was shown that a weaker authentication can be supported by using a variation of the TLS preshared key cipher suite. Here the publisher places less trust in the subscriber and requested an authentication of the subscriber by the access control server together with the generation of a session key.

In the future the developed TinyIPFIX protocol with all its currently available extensions could be leveraged in other scenarios and deployments as well. In addition, the development of extensions, especially security support, will go on after the submission of this dissertation. A long time goal in the future is to extend the DTLS solution to more constrained hardware in order to support a high level of security even on the level of data collectors and not only between cluster heads and gateway. A high security level should be established whitin the cluster itself. In order to support access for mobile devices the established security solution will be extended with the required authentication and validation procedures.
Another field for more research will focus on optimizing the system's lifetime through intelligent energy saving methods, because energy resource is limited due to battery capacity of the nodes (cf. Figure 2.2) Depending on the development of visualization tools, the graphical user interface will be extended.
Depending on different application scenarios it might be preferable to contact a data collector directly and to request an immediately measurement. This request can be realized by expanding the existing push functionality of TinyIPFIX with a pull functionality. This means, a measurement request is translated to a special TinyIPFIX Template Record and directly addressed to the measurement point (e.g. sensor node with ID 1104). If the measurement point receives the Template Record it immediately measures the requested value (e.g. temperature) and transmits it back to the sink referring to the received template.
Finally, a transfer of the implementations developed during this dissertation to the operating system Contiki is planned in order to support more vendors and due to the industrial requests. Therefore, each part must be converted and adjusted to Contiki as indicated in Section 2.4.3. Modules supporting the network configuration by Contiki must extend the developed graphical user interface.

Appendix

A. TinyIPFIX Template and Data Set Construction

This appendix compares the underlying code parts of TinyIPFIX, which are performed in order to create the required Template and Data Sets for a specific sensor node. If a sensor node enters the existing wireless sensor network, it has to prepare its supported Template Set for TinyIPFIX in order to announce the Template to the neighbors in the network. Therefore, the sensor network directly performs the task shown in Figure A.1 in order to build its Template. In comparison Figure A.2 shows the task, which is performed by the sensor node in order to create its Data Set for upcoming data transmissions of measured data. [98]
Both task blocks are bracketed with signaling calls to indicate either start or end of the set construction. In this example, the Template ID of 256 is included in the Data Set calls in order to ensure successful decoding later on in the system. The big difference between both task blocks is in the FOR-Loop. The Field ID, the Field Length, and the Enterprise ID are specified for Template calls. In comparison, in Data Set's FOR-Loop the value and the size of the value are specified. Those two lines are based on the standardization by the IETF for the IPFIX protocol [19] and the IANA registration [74].

```
task void makeTemplate(){
        uint16_t i,j = 0;

        call tinyIPFIX.net_start_template(0);
        call tinyIPFIX.start_template_set(0);
        call tinyIPFIX.start_template_record(0, 256);
        for( i=0; i<degree_of_aggregation; i++ ) {
                for( j=0; j<tinyipfix_set[i].template_set.field_count; j++ ) {
                        call tinyIPFIX.put_template_field(0, tinyipfix_set[i].template_set.field[j].field_id + agg_id,
                                                             tinyipfix_set[i].template_set.field[j].field_length,
                                                             tinyipfix_set[i].template_set.field[j].e_id );
                }
                agg_id = agg_id << 1; // makes Template Field IDs unique for one Template Set.
        }
        call tinyIPFIX.end_template_record(0);
        call tinyIPFIX.end_template_set(0);
        call tinyIPFIX.net_finish_template(0);
        agg_id = 0x0100;
}
```

Figure A.1: Task structure for new Template Set Construction

```
task void makeData() {
        uint16_t i,j,k = 0;
        nx_uint16_t buff32[2];
        buff32[0] = 0x0000;
        buff32[1] = 0x0000;

        call tinyIPFIX.net_start_data(0);
        call tinyIPFIX.start_data_set(0, 256);
        call tinyIPFIX.start_data_record(0);
        for( i=0; i<degree_of_aggregation; i++ ) {
                for( j=0; j<tinyipfix_set[i].template_set.field_count; j++ ) {
                        if( tinyipfix_set[i].template_set.field[j].field_length == sizeof(uint16_t) )
                                call tinyIPFIX.put_data_field(0, &(tinyipfix_set[i].data_set.value16[k++]), sizeof(uint16_t));
                        else if( tinyipfix_set[i].template_set.field[j].field_length == sizeof(uint32_t) ){
                                buff32[0] = tinyipfix_set[i].data_set.value16[k++];
                                buff32[1] = tinyipfix_set[i].data_set.value16[k++];
                                call tinyIPFIX.put_data_field(0, (nx_uint32_t*)buff32, sizeof(uint32_t));
                        }
                }
                k = 0;
        }
        call tinyIPFIX.end_data_record(0);
        call tinyIPFIX.end_data_set(0);
        call tinyIPFIX.net_finish_data(0);
}
```

Figure A.2: Task structure for new Data Set Construction

B. Algorithms for TinyIPFIX-Aggregation

This appendix deals with the packet handling during the aggregation performance within the wireless sensor network. Aggregation is performed by so called aggregator nodes, which are sensor nodes located in strategic positions in the deployed wireless sensor network. As described in Section 4.2.2.1 the developed aggregation framework for TinyIPFIX supports two aggregation modes:

1. *Message Aggregation*: Aggregation of several data messages in one packet.
2. *Data Aggregation*: Data pre-processing within the transmission way to the gateway using aggregation functions.

On the sensor node itself both algorithms for aggregation are installed per default in order to allow switching between them. The aggregation functionality was implemented in the aggregation control procedure as part of *AggregatorC.nc* shown in Figure B.2. In general, algorithm A, which represents the most interesting aggregation mode, is performed to reduce the data amount in the network without losing any information. Due to the implemented UDP-shell, the user is able to switch between the aggregation modes or directly modify the aggregation (e.g. change degree of aggregation) as described in Section 5.2.3. If the aggregation mode is changed to data aggregation (cf. blue dashed box in lower part of Figure B.1), the new required aggregated Template Set is constructed automatically and announced to the wireless sensor network.

Figure B.1 illustrates the underlying decision tree and the performed operations for the implemented TinyIPFIX-Aggregation framework. The upper part shows the decisions and operations for message aggregation (Algorithm A), and the lower part those for data aggregation (Algorithm B). The grey filled rectangles indicate operations where an input from outside is requested. Diamonds indicate questions (checks), which cause reactions depending on the respective answer to the questions. White boxes stand for operations performed by the sensor node in order to handle received data.

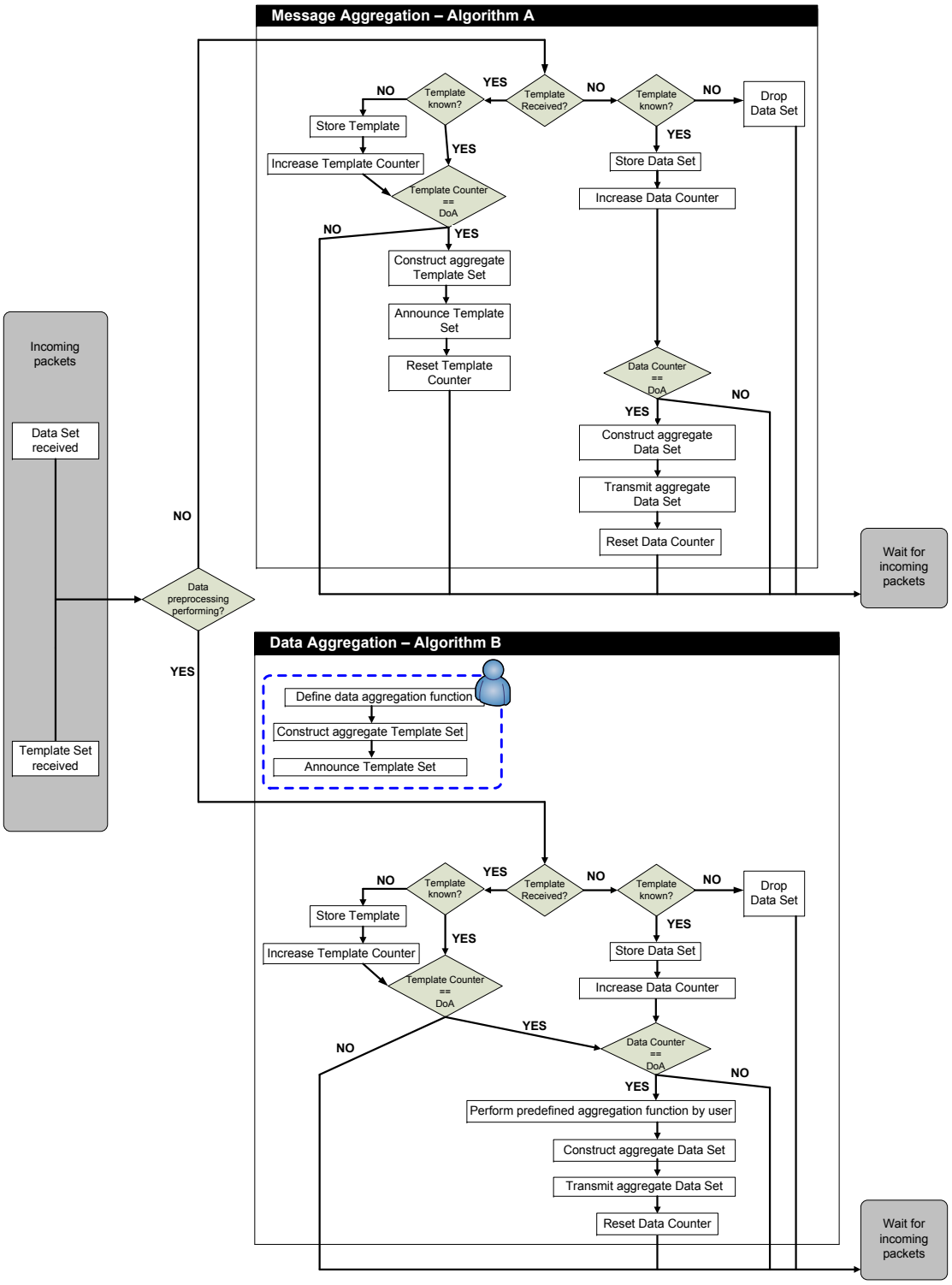


Figure B.1: Decision and operation tree for TinyIPFIX-Aggregation framework

The sensor node (aggregator) receives either a data set or a template set. The aggregator must decide depending on the incoming packet and the installed aggregation mode if the message or data aggregation must be performed by answering the question 'Data preprocessing performing?'. If the answer is 'NO', algorithm A for message aggregation is performed (cf. upper part of Figure B.1), otherwise algorithm B for data aggregation is performed (cf. upper part of Figure B.1). [102]

The next steps are the same in both algorithms: First, it must be checked if the received packet is a Template or a Data Set (check 1). This happens due to the packet structure of TinyIPFIX (cf. Figure 3.3). If the answer of the first check is 'YES', a Template Record is received and it must be checked if the Template is already known (check 2). If for the second check the answer is 'NO', the new received Template must be stored and the internal Template counter is increased. Otherwise nothing happens. If the answer of the first check is 'NO', a Data Record was received. In this case, it must be checked if the Template, which the Data refers to, is already known (check 3). If the required Template is unknown the decoding of the received Data would be unsuccessful and, therefore, the Data is dropped. If the referred Template is known, the new received Data must be stored and the internal Data counter is increased. [102]

From this point on, algorithms differ and are described separately until completion. The upcoming decisions are influenced by predefined information (e.g. DoA, aggregation function) on the sensor node performing the corresponding algorithms.

Algorithm A dealing with message aggregation now checks if the value of Template counter is equal to the before specified degree of aggregation (DOA) (check 4). If this is not the case, the algorithm breaks and waits for more incoming packets and the whole process begins at the first question 'Data preprocessing performing?' again. If the answer is positive of check 4, the required aggregated Template Set is constructed (cf. Figure B.3). The aggregated Template Set is announced to the wireless sensor networks in order to allow decoding by other sensor nodes in the system. Finally, the Template counter is reset. On the algorithm part handling received data packets the next check after a positive check 3 deals with the comparison of the internal Data counter and the predefined degree of aggregation (check 5). If the check is negative, the algorithm breaks and waits for more incoming packets, starting by question 'Data preprocessing performing?' again when receiving new packets. If check 5 is positive, the required aggregated Data Set is constructed (cf. Figure B.4) and transmitted to the next hop in direction of sink. Finally, the Data counter is reset. [102]

Algorithm B dealing with data aggregation checks next, if Template Counter and predefined degree of aggregation is equal (check 6). If the result is negative, algorithm breaks and waits until new packets are received in order to start over with question 'Data preprocessing performing?'. If check 6 is positive, the data counter is compared to the predefined degree of aggregation (check 7). If check 7 is positive the predefined aggregation function (e.g. MAX, MIN, AVG) is performed, the data set including the result and the reference to the Template is constructed. The sensor node transmits the aggregates Data Set to the next hop in direction of the sink. An announcement of the Template is not necessary, because it was already announced when the user specified the performed data aggregation type. Finally, the Data counter is reset. [102]

```
void aggregation_control(bool template_recv, bool template_update, bool data_recv, bool data_update) {

// Data pre-processing protocol
        if(preprocessing){
                        if(data_recv){
                                if(data_update){
                                        if(data_update_count == degree_of_aggregation){
                                                data_update_count = 0;
                                                post makeAggregateData();
                                        }
                                }else{
                                        if(data_count == degree_of_aggregation){
                                                data_count = 0;
                                                post makeAggregateData();
                        } } }
// Message aggregation protocol
        }else{
                if(template_recv){
// If the newly or updated received Template Set count has archieved the Degree of Aggregation,
// it generates a new aggregation Template Set
                        if(template_update){
                                if(template_update_count == degree_of_aggregation){
                                        template_update_count = 0;
                                        post makeTemplate();
                                }
                        }else{
                                if(template_count == degree_of_aggregation){
// The template_count can not be set to 0, because the amount of  buffered template sets is needed  by other
//  procedures. To prevent buffer overwrites the count is checked in the read_template procedure.
                                        post makeTemplate();
                                }
                        }
                }else if(data_recv){
// If the newly or updated received Data Set count has archieved the DoA, it generates a new aggregated Data Set
                        if(data_update){
                                if(data_update_count == degree_of_aggregation){
                                        data_update_count = 0;
                                        post makeData();
                                }
                        }else{
                                if(data_count == degree_of_aggregation){
                                        data_count = 0;
                                        post makeData();
                                }
} } } }
```

Figure B.2: Aggregation control logic for message aggregation

```
task void makeAggregateTemplate() {
        call tinyIPFIX.net_start_template(0);
        call tinyIPFIX.start_template_set(0);
        call tinyIPFIX.start_template_record(0, 256);
        call tinyIPFIX.put_template_field(0, aggregate.template_set.field[0].field_id,
                                             aggregate.template_set.field[0].field_length,
                                             aggregate.template_set.field[0].e_id );
        call tinyIPFIX.put_template_field(0, aggregate.template_set.field[1].field_id,
                                             aggregate.template_set.field[1].field_length,
                                             aggregate.template_set.field[1].e_id );
        call tinyIPFIX.end_template_record(0);
        call tinyIPFIX.end_template_set(0);
        call tinyIPFIX.net_finish_template(0);
}
```

Figure B.3: Task structure for computation of aggregated Template Sets

```
task void makeAggregateData() {
        uint16_t nodeID;

        call tinyIPFIX.net_start_data(0);
        call tinyIPFIX.start_data_set(0, 256);
        call tinyIPFIX.start_data_record(0);
// switch on the requested aggregate function.
        switch (aggregation_func){
                case AGG_MAX:  agg_max();   break;
                case AGG_MIN:  agg_min();   break;
                case AGG_AVG:  agg_avg();   break;
                case AGG_ALL:  agg_all();
        }
// Aggregate function has returned with the aggregated value and Node ID, and are put into the data field.
        nodeID = (uint16_t)*((nx_uint16_t*)(&(aggregate.data_set.value16[1])));
        call tinyIPFIX.put_data_field(0, &(aggregate.data_set.value16[0]), sizeof(uint16_t));
        call tinyIPFIX.put_data_field(0, &nodeID, sizeof(uint16_t));
        call tinyIPFIX.end_data_record(0);
        call tinyIPFIX.end_data_set(0);
        call tinyIPFIX.net_finish_data(0);
}
```

Figure B.4: Task structure for computation of aggregated Data Sets

C. Component Wiring under TinyOS

This appendix deals with the component wiring of the implemented protocols TinyIPFIX and TinyDTLS. The term *wiring* describes the semantics in which way all components under TinyOS are linked with each other [25].

Figure C.1 illustrates the simplified application wiring in TinyOS. Red encircled boxes are main operative components. Boxes represent files. Double rectangles indicate configurations, single rectangles are modules, solid lines stand for singletons, and dashed lines for generic components. Those generic components (e.g. `IPFIXDataSampler16C` representing temperature or light) can exist multiple times, because only one value can be reported at one time. The grey filled circle represents an interface, which is, in this case, provided by the file `NetworkHandlerC`. The `NetworkHandlerC` itself provides the `NetworkHandlerAppC`, which is indicated by an incoming arrow in `NetworkHandlerC`. The `NetworkHandlerC` in turn uses several other interfaces that are specified along the arrows (e.g. `SplitControl`, `Send`). Those interfaces are provided by the components, which these arrows point to. [98]

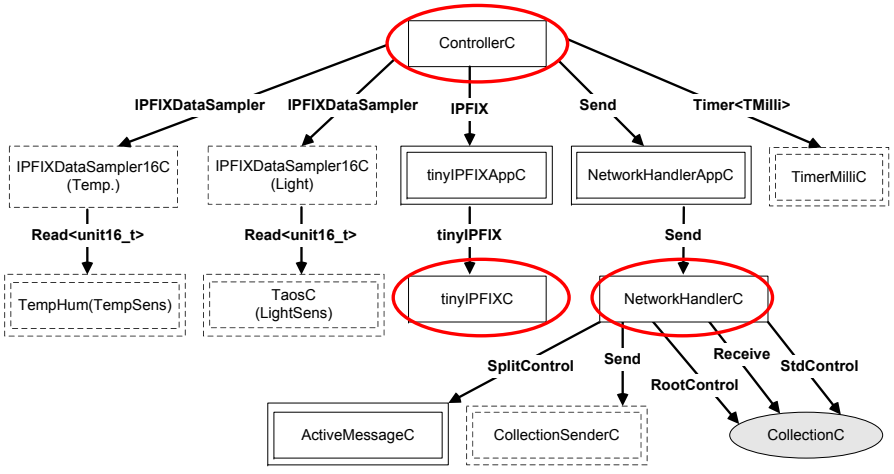


Figure C.1: Simplified node wiring of components for TinyIPFIX

For completeness' sake, the corresponding component wiring for the DTLS client implementation on OPAL is shown in Figure C.2. The interpretation of files, circles, and lines is the same as before. Components supporting DTLS functionality are indicated by `DTLS...` and components supporting hash functionality are indicated by `HMACM...`. The latter components exist several times, because hash functions over different inputs (e.g. SHA, MD5) are required during the handshake performance. [81]

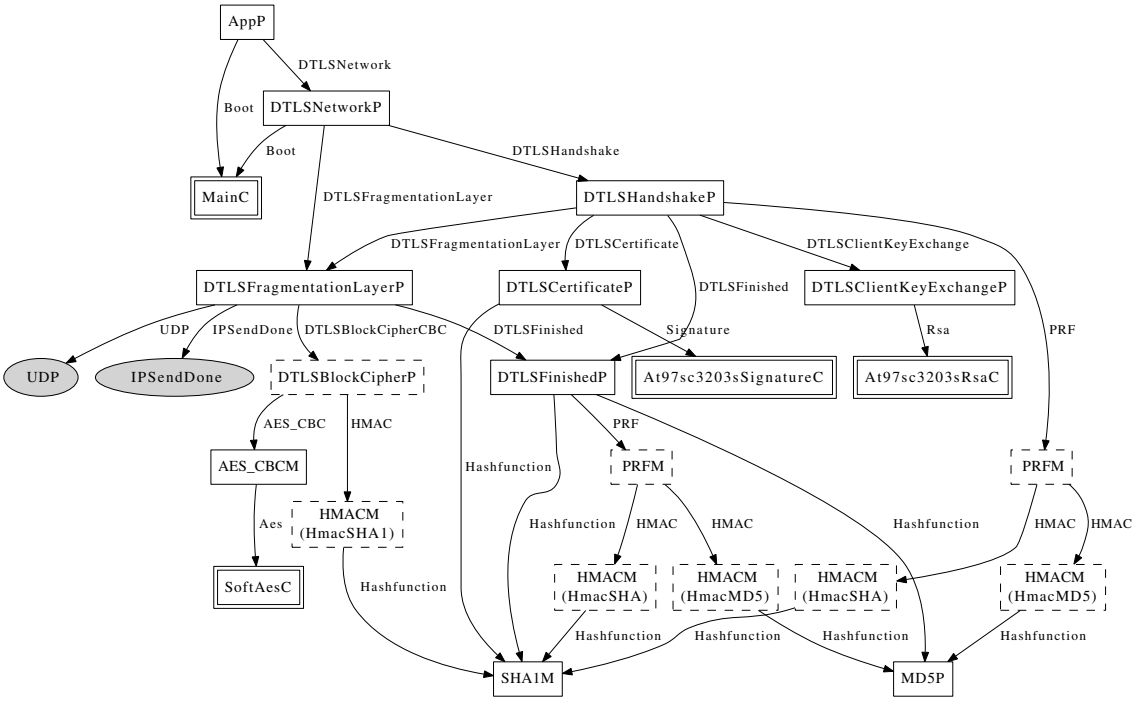


Figure C.2: Simplified wiring of the DTLS client implementation

D. Functionalities of GUI

This appendix includes different figures, which characterize different functionalities supported by the developed graphical user interface in this dissertation. Before deploying a wireless sensor network its components must be configured, which is exemplarily shown in Section D.1. Section D.2 presents an exemplary virtual representation of an established wireless sensor network, which is requested for the internal processes in the graphical user interface in order to offer manifold feedback (e.g. network status, recorded data) to the user (cf. Section D.3).

D.1 Hardware Configuration

As described in detail in Section 5.4.1, the developed graphical user interface offers the user the opportunity to configure network components via clicks instead of complex command lines. In order to program individual components of the wireless sensor network, the user must change to the submenu option `TinyOS` in menu `Hardware` in the so called 'WSN Administration'. The interface gives the user the information about the supported operating systems, which in this case is TinyOS 2.1.1 together with the included modules. The modules include an interface to program the base station, the individual nodes as well as the activation of the IP tunnel.

An overview is given in Figure D.1 showing exemplary the interface for programming a sensor node. Here the user is able to configure the sensor node in six steps as described in detail in Section 5.4.1:

1. Specify target (= hardware platform).
2. Specify attached sensor board.
3. Specify extras, where BLIP support is selected per default.
4. Specify functionality performed by node (e.g. aggregation, collector).
5. Compile program code regarding the above specifications.
6. Specify individual node ID and install program on the hardware.

During the last two steps the user receives visual feedback about the compiling and installing process. Additionally, the user receives information about the final ROM and RAM consumption. [133]

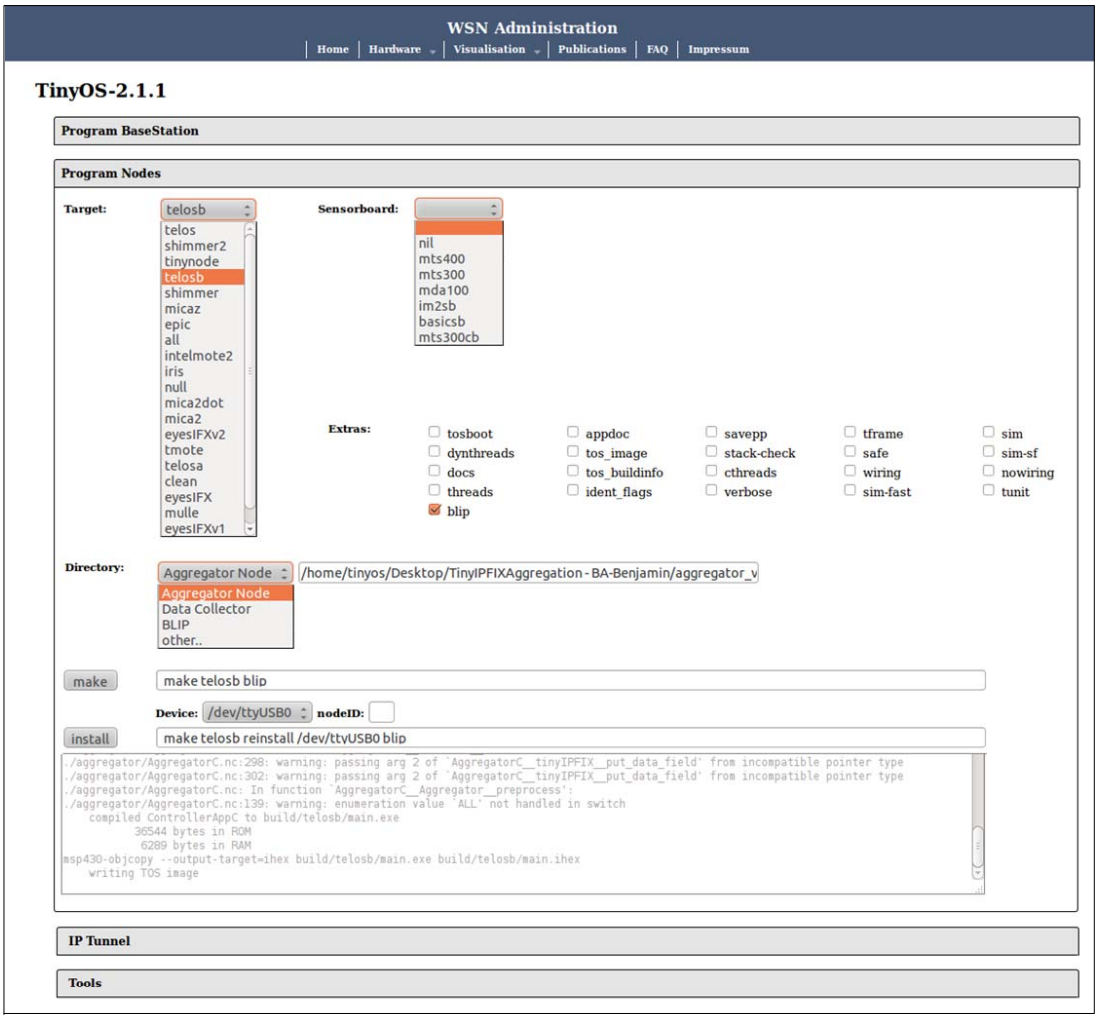


Figure D.1: GUI - Configuration options for sensor nodes

D.2 Virtual Representation of Wireless Sensor Network

As described in Section 4.4.2 the developed graphical user interface virtualizes an existing wireless sensor network by using the class *WSN*, as described in detail in reference [133]. The class *WSN* includes information about nodes and the topology of the wireless sensor network. The class WSN is connected to the two classes *WSNTopology* and *WSNNodes* offering information about the links within the network (respectively about the data transmitted by the nodes). Those two classes are connected to the class *WSNNodeTopology::Link*, which offers information about source and target. The class *WSNNodes* has an additional connection to the class *WSNNode::Datum* including information about the type, value, and unit represented by the sensor node. [133]
Figure D.2 shows an exemplary virtual representation of a deployed wireless sensor network consisting of four sensor nodes (ID 0, 1105, 1104 and 2203) and a server. The corresponding link topology is shown in the lower right corner of the figure. The class *WSN* exists once, including node and topology information. It is linked

to each class *WSNNodes* and class *WSNTopology*. For each sensor node in the wireless sensor network a class *WSNNodes* is established (here: three times). The class *WSNNodes* include ID information and data information for each sensor node, where each class *WSNNodes* has linked as many entities of class *WSNNode::Datum* as data fields are required by the corresponding node. The class *WSNNode::Datum* includes information of type, value, and unit of the recorded data. For example, the node ID 2203 transmits three values (temperature, humidity, voltage), which links three times the class *WSNNode::Datum* to the corresponding class *WSNNodes*. The class *WSNTopology* includes information of the established links in the wireless sensor network, where for each link a class *WSNNodeTopology::Link* exist including source and next hop (target) information. [133]

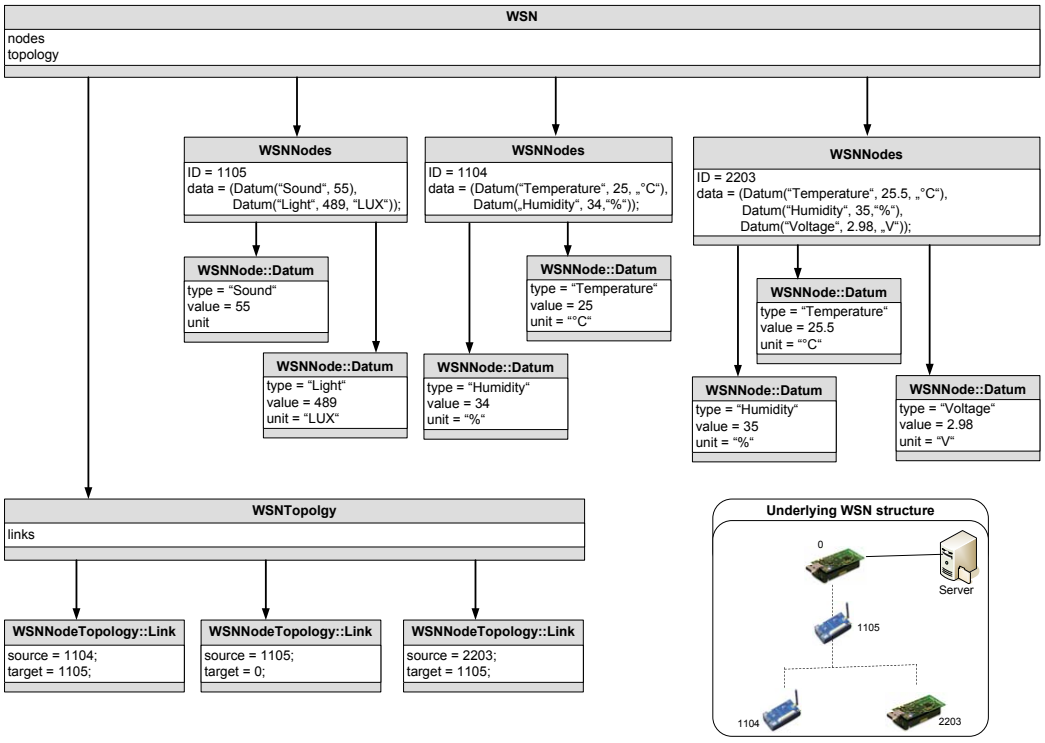


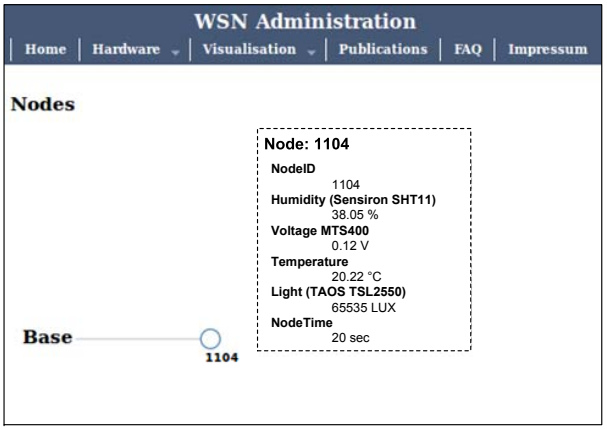
Figure D.2: Exemplary UML representation

D.3 Feedback of Network Status

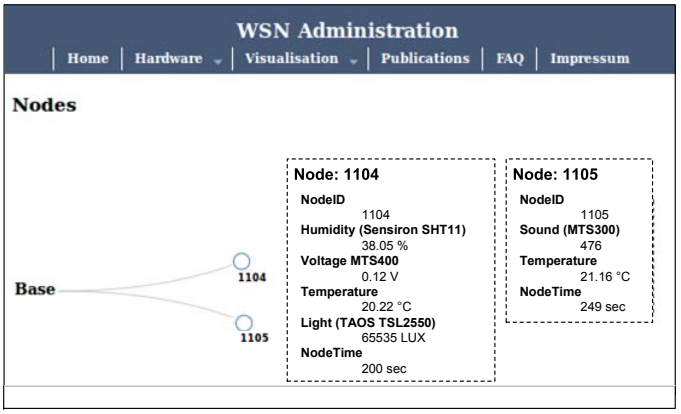
As described in Section 5.4.2 the established graphical user interface offers the user live feedback about the status of the deployed wireless sensor network. In addition, different opportunities are integrated to handle received data.

D.3.1 Feedback about Routing Development

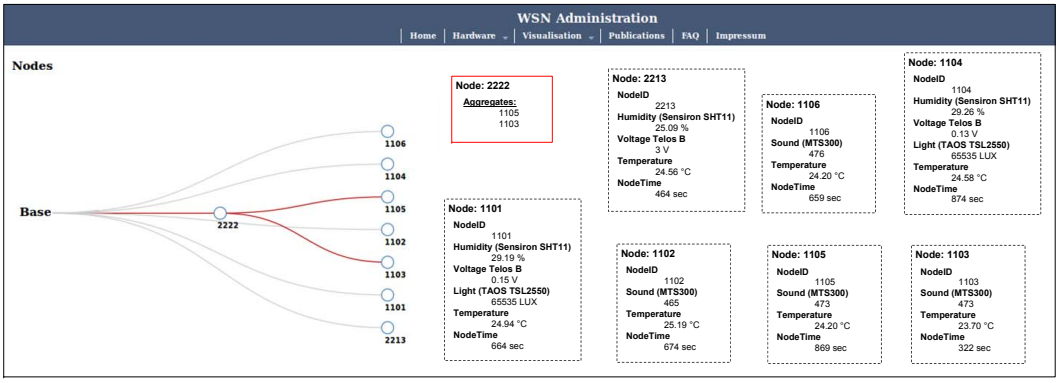
The user can observe the organization and establishment of the communication links between sensor nodes. Figure D.3 illustrates an example of the visualization event, where the right part is based on the module `WSNDriver` and the left part on the TinyOS function `BLIP`. The individual node information on the right part is updated periodically when the routing tree updates itself. In conclusion, Figures D.3a to D.3c show the establishment of the routing tree of the running wireless sensor network. The visualization offers the opportunity to highlight nodes together with their incoming and outgoing connection (cf. Figure D.3c). [133]



(a) First sensor node is activated.



(b) Second sensor node is activated.



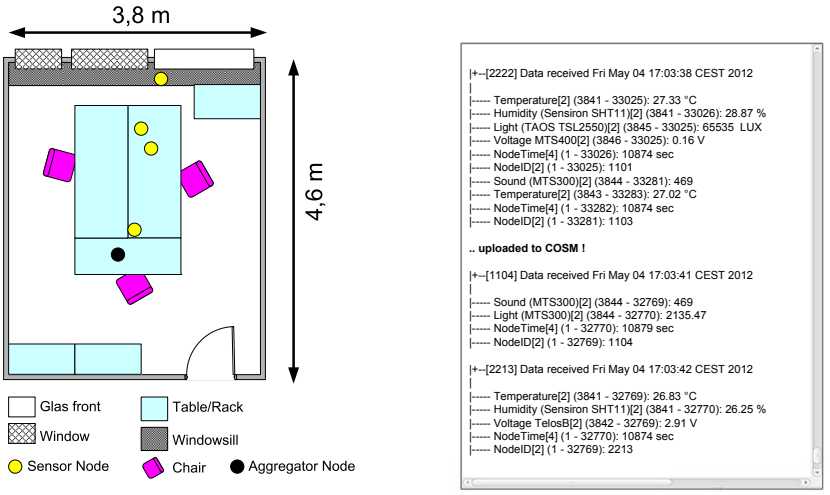
(c) Network with nine active nodes.

Figure D.3: GUI - Development of the network structure visualization

D.3.2 Data Handling

The developed graphical user interface allows the user to handle received data in different ways: (1) Live streaming of data, (2) export data to online visualization tools (e.g. COSM), and (3) store data in order to import data for offline analysis (e.g. Matlab). [133]

The user is able to view live recording of received sensor data. Figure D.4b illustrates the capturing of the Listener provided by TinyOS. The corresponding node deployment in the office is illustrated in Figure D.4a, where the wireless sensor network consists of four sensor nodes and one sink (black).



(a) Node location (b) Recorded data flow in Listener

Figure D.4: Background information to results shown in Figure D.6

Figure D.5 shows an example of processing of the received data. Received data is directly stored in a big file (see left part). Usually, the user just wants to plot one subset of data in general measurements of one selected sensor node. The big file, therefore, must be subdivided into its components, which are stored in small files. This processing is performed automatically in the background of the graphical user interface.

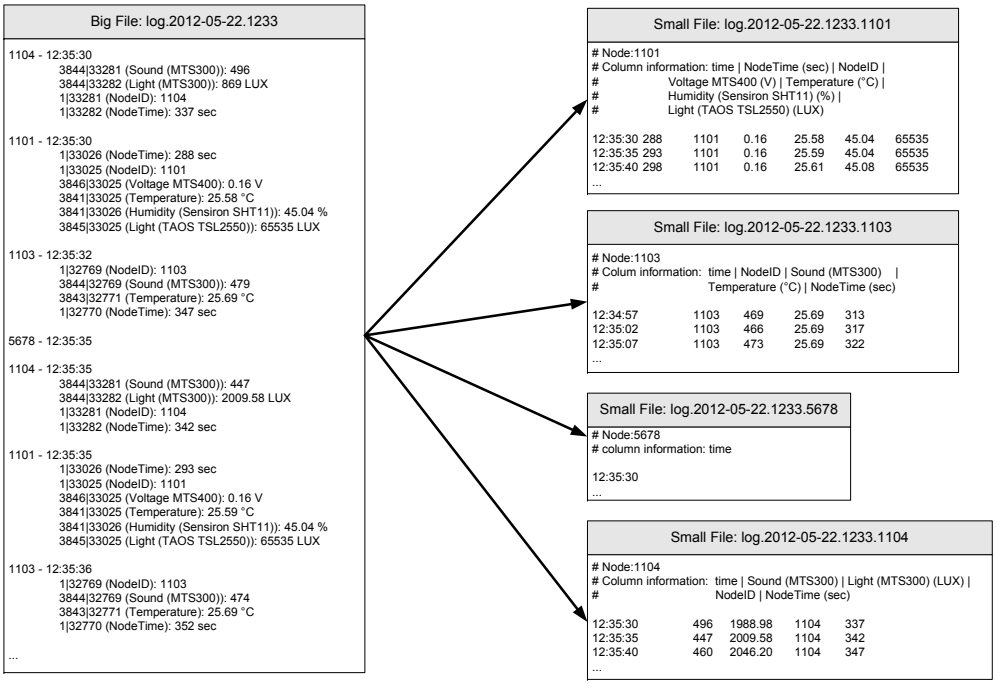


Figure D.5: Data preprocessing for analysis via offline tools

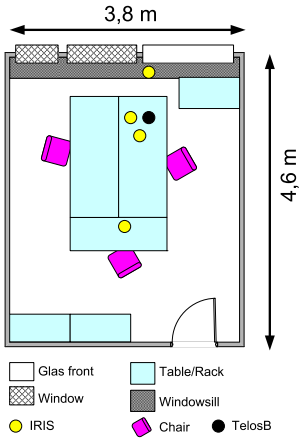
Figure D.6 shows a data export and import example. This Figure shows the user's ability to remove unnecessary information from the visualization of COSM in the implemented graphical user interface. The upper part shows a complete visualization of all exported data to COSM for node 1101. Whereas the lower three nodes show only selected information by the user in a diagram. On the right side of those nodes the user has still the whole exported information in a compact version visible. Figure D.4 shows the node location in the office and a screenshot of the recorded data.

Figure D.6: Data export/import using COSM

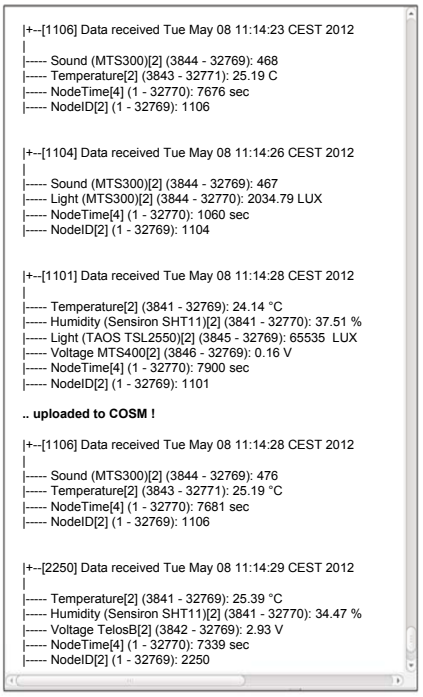
D.4 Experiment 1

In this section an exemplary wireless sensor network is introduced under the name 'Experiment 1' assuming an office scenario. Figure D.7.b illustrates the capturing of the Listener provided by TinyOS. The corresponding node deployment in the office is illustrated in Figure D.7.a, where the wireless sensor network consists of four sensor nodes and one sink (black). The visualization of the wireless sensor network including routing information and all transmitted data of each node is illustrated in Figure D.8.

Sensor nodes 1102 and 2250 had the sensor for humidity measurements activated. Recorded data shows that during recording time humidity differed (cf. the graph of node 1102). At the beginning of the experiment, doors and windows were closed in the office since 9 p.m. the day before. This fact results in a quite low humidity in the room, recorded by node 1101 with 40.09%. In the morning a summer rain raised humidity outside. As a consequence of opening windows humidity changed inside as well. The node recorded a continuous rise up to 42.52% until 10:15 a.m. For the next 25 minutes the value fluctuated a little bit. Then the summer sun came out and temperature outside increased, which resulted in a drying of the air. As a



(a) Node Deployment - Type 2



```
|+--[1106] Data received Tue May 08 11:14:23 CEST 2012
|
|----- Sound (MTS300)[2] (3844 - 32769): 468
|----- Temperature[2] (3843 - 32771): 25.19 C
|----- NodeTime[4] (1 - 32770): 7676 sec
|----- NodeID[2] (1 - 32769): 1106

|+--[1104] Data received Tue May 08 11:14:26 CEST 2012
|
|----- Sound (MTS300)[2] (3844 - 32769): 467
|----- Light (MTS300)[2] (3844 - 32770): 2034.79 LUX
|----- NodeTime[4] (1 - 32770): 1060 sec
|----- NodeID[2] (1 - 32769): 1104

|+--[1101] Data received Tue May 08 11:14:28 CEST 2012
|
|----- Temperature[2] (3841 - 32769): 24.14 °C
|----- Humidity (Sensiron SHT11)[2] (3841 - 32770): 37.51 %
|----- Light (TAOS TSL2550)[2] (3845 - 32769): 65535  LUX
|----- Voltage MTS400[2] (3846 - 32769): 0.16 V
|----- NodeTime[4] (1 - 32770): 7900 sec
|----- NodeID[2] (1 - 32769): 1101

.. uploaded to COSM !

|+--[1106] Data received Tue May 08 11:14:28 CEST 2012
|
|----- Sound (MTS300)[2] (3844 - 32769): 476
|----- Temperature[2] (3843 - 32771): 25.19 °C
|----- NodeTime[4] (1 - 32770): 7681 sec
|----- NodeID[2] (1 - 32769): 1106

|+--[2250] Data received Tue May 08 11:14:29 CEST 2012
|
|----- Temperature[2] (3841 - 32769): 25.39 °C
|----- Humidity (Sensiron SHT11)[2] (3841 - 32770): 34.47 %
|----- Voltage TelosB[2] (3842 - 32769): 2.93 V
|----- NodeTime[4] (1 - 32770): 7339 sec
|----- NodeID[2] (1 - 32769): 2250
```

(b) Capture of Listener

Figure D.7: Experiment 1: Node placement and capture of Listener

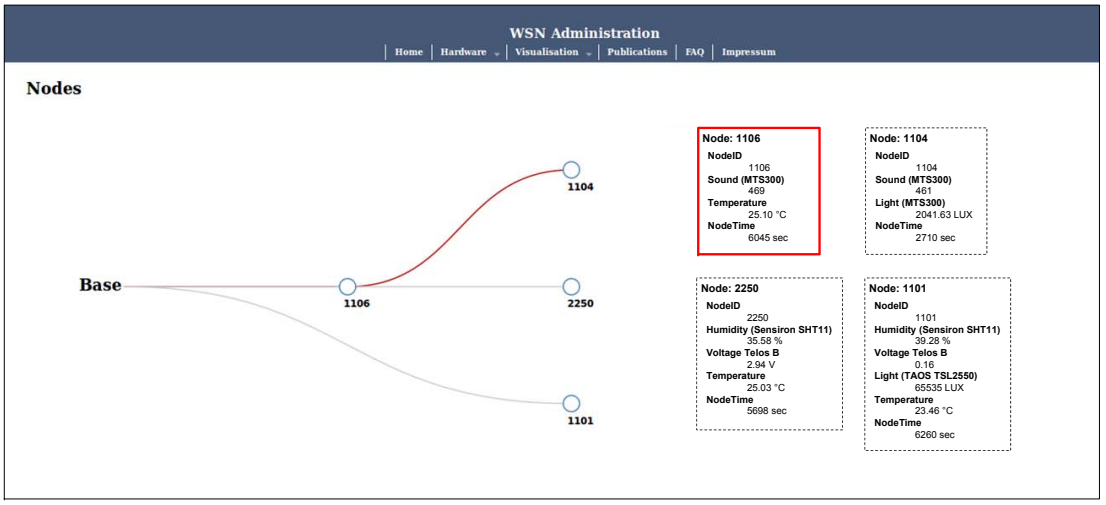


Figure D.8: Experiment 1: Underlying routing structure

consequence humidity inside the office also fell down to 38.4%. From this point on node humidity during the experiment fluctuated, but never reached highest value again. The humidity level was confirmed by node 2250. In the displayed COSM graph the resolution was set to one hour in order to receive a more detailed humidity analysis from 12 p.m. to 1 p.m..

Voltage consumption is recorded by the nodes 1101 and 2250. The TelosB node measures a constant of three Volts while the consumption of the IRIS node fluctuates.

The fluctuations are caused by activation of LEDs, measurement requests and data transmissions.

The sensor nodes 1101, 2250, and 1106 recorded room temperature. The recorded values are nearly the same. Differences are based on the location of the nodes in the office. The node next to the window records lower temperatures if the window is opened in comparison to nodes on the tables, because they are sheltered. The sensor nodes 1101 and 1104 recorded the brightness. Sensor node 1101 can be omitted in the current test run for this value, because the node's driver for this sensor crashed.

Node 1104 is located next to a big monitor which functions as a sun shield. Thus, recorded peaks are based on monitoring light and turned on lamps in the office. The graph shows a suspiciously constant value of 2041.63 LUX for about one hour. This was the last reported value before the node crashed. After the user made a restart the recording resumed. COSM allows the user to add a restarted node to an 'old' feed again if the values are the same. This was performed in this case. The same break down and restart is seen in the corresponding sound versus time curve of this node. In order to confirm the break down even with the restarted internal clock of the sensor node the user zoomed into the corresponding time curve.

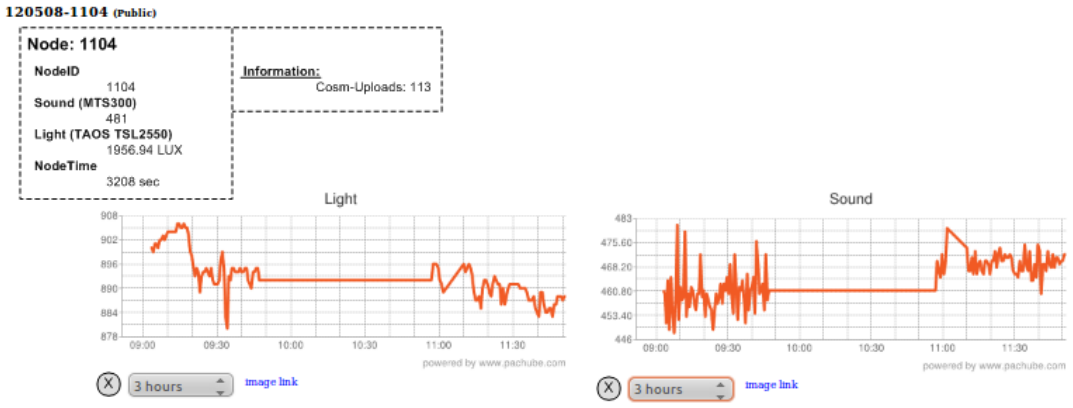


Figure D.9: Experiment 1 - Zoom in node 1104

As mentioned sensor node 1104 measured room brightness as well (cf. Figure D.9). For calculation of raw data into the LUX value Equations D.1 to D.3 is used based on the vendor information [32]. Whereas the photodiodes create a current through a $100k\Omega$ resistor, $V_{ref} = 1.5V$, and ADC_{value} is the measured raw data (e.g. 899). In the first recorded period the measured value rose from 899 respectively 2057.65 LUX periodically to 906 respectively 2069.09 LUX, which is caused by scattered cloud cover outside. As a consequence the automatic shutter went down. This is corroborated by the measured brightness drop to 888 respectively 2032.47 LUX. During the next 10 minutes the shutter was adjusted, which is visible in the recorded brightness curve. Suddenly it was complete sunny outside; therefore the brightness rose up to 900 (respectively 2059.94 LUX) again. The shutter closed completely, which reduced the recorded brightness in the room to 879 (respectively 2011.87 LUX). After some minutes it became more cloudy outside again and the shutter opened up again resulting in a brightness of 895 respectively 2048.49 LUX. From then on up to the node crash recorded values are nearly stable at this value range. Another suspiciously constant region resulted from crashed hardware as described above. Around 11 a.m. the node was restarted. From this

time on a fluctuation between 885 and 896 (respectively 2025.61 - 2050.78 LUX) was recorded due to changing cloud conditions outside. In this duration the automatic shutter was inactive.

$$V_{sensor} = (ADC_{value}/4096) * V_{ref} \tag{D.1}$$

$$I = V_{sensor}/100,000 \tag{D.2}$$

$$Sensor_{LUX} = 0.625 * 10^6 * I * 1000 \tag{D.3}$$

Nodes 1104 and 1106 had the acoustic sensor activated. The sensor is not designed to measure frequency versus time, but rather changes in amplitude (e.g. alarms, phone rings, discussions). Specifically, the recording of node 1106 shows a volume change in the office. A period of high change results in a range of 439 to 486, which is caused by a radio at high volume. The recorded values by node 1104 show nearly the same range.

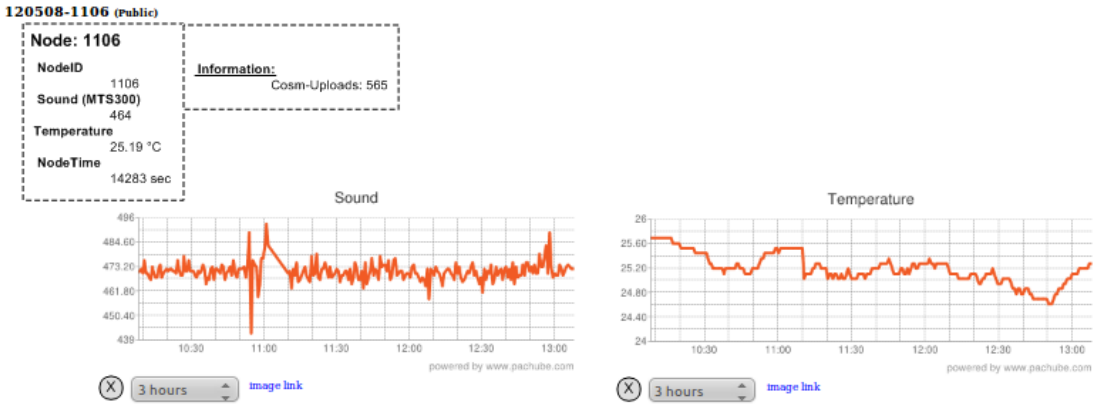


Figure D.10: Experiment 1 - Zoom in node 1106

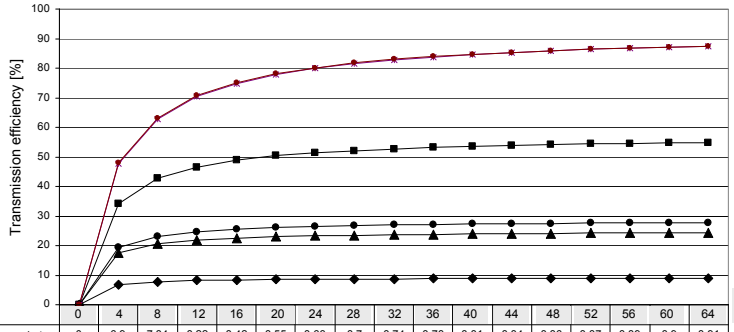
Figure D.11 shows the COSM visualization of experiment 1 for nodes 1101 and 2250. Here the user can specify different visualization intervals for each sensor ranging from five minutes up to three hours. Additionally, the user can delete visualization of sensor data by clicking on the 'X' button in order to reduce not relevant information (e.g. NodeID). In the background the storage of the complete data is performed in order to have all information available for offline analysis.

Figure D.11: Experiment 1 - COSM visualization nodes 1101 and 2250

E. Transmission Efficiency of TinyIPFIX

Figures E.1 to E.3 dealing with the evaluation of the transmission efficiency of TinyIPFIX (cf. Section 6.1.2). Those Figures illustrate results where the transmission efficiency was plotted against the number of data packets transmitted between two Template Records and s = 2 bytes was assumed. In order to illustrate different types of packet configuration, the number of values per packet ranges from one to 125, where one value per packet is the lowest limit, four values per packet are common for wireless sensor networks, and 125 values per packet is the maximum packet size for IP communication. [98]
The analysis of TinyIPFIX transmission efficiency is split into the following cases [98]. In each plot the x-axis represents the number of transmitted Data packets per Template retransmission and on the y-axis the corresponding transmission efficiency in percentage is plotted. [98]

Case 1 analysis the transmission efficiency regarding TinyIPFIX packets with default header of size 20 bytes vs. header supporting defensive approach in best-case with six bytes size. Figure E.1 shows the received data and displays the concrete values [98].

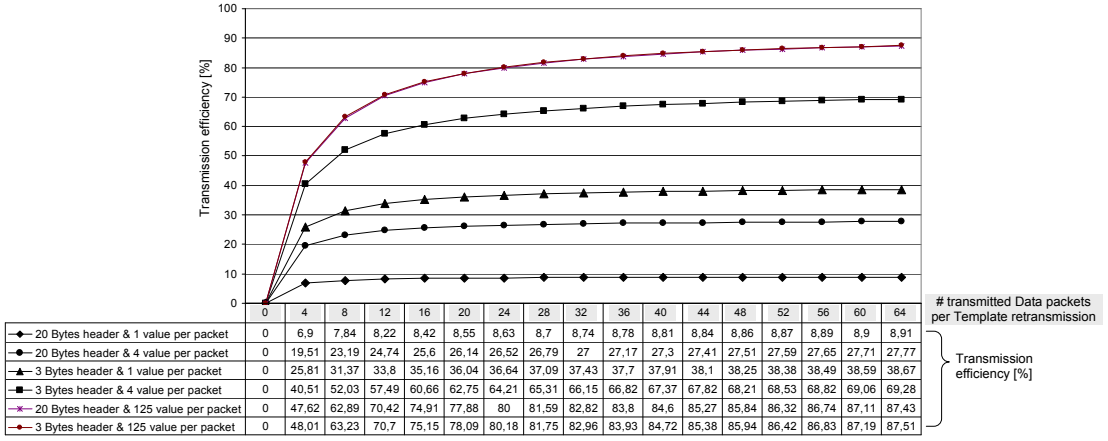


# transmitted Data packets per Template retransmission	0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64
20 Bytes header & 1 value per packet	0	6,9	7,84	8,22	8,42	8,55	8,63	8,7	8,74	8,78	8,81	8,84	8,86	8,87	8,89	8,9	8,91
20 Bytes header & 4 value per packet	0	19,51	23,19	24,74	25,6	26,14	26,52	26,79	27	27,17	27,3	27,41	27,51	27,59	27,65	27,71	27,77
6 Bytes header & 1 value per packet	0	17,39	20,51	21,82	22,54	22,99	23,3	23,53	23,7	23,84	23,95	24,04	24,12	24,19	24,24	24,29	24,33
6 Bytes header & 4 value per packet	0	34,04	42,67	46,6	48,85	50,31	51,34	52,09	52,67	53,14	53,51	53,82	54,08	54,31	54,5	54,67	54,82
20 Bytes header & 125 value per packet	0	47,62	62,89	70,42	74,91	77,88	80	81,59	82,82	83,8	84,6	85,27	85,84	86,32	86,74	87,11	87,43
6 Bytes header & 125 value per packet	0	47,94	63,17	70,65	75,1	78,05	80,15	81,72	82,94	83,91	84,7	85,36	85,92	86,4	86,82	87,18	87,5

Transmission efficiency [%]

Figure E.1: TinyIPFIX transmission efficiency - Case 1

Figure E.2 illustrates results of case 2, where TinyIPFIX packets with default header of size 20 bytes are compared with packets supporting modified defensive approach of header compression with three bytes size [98].



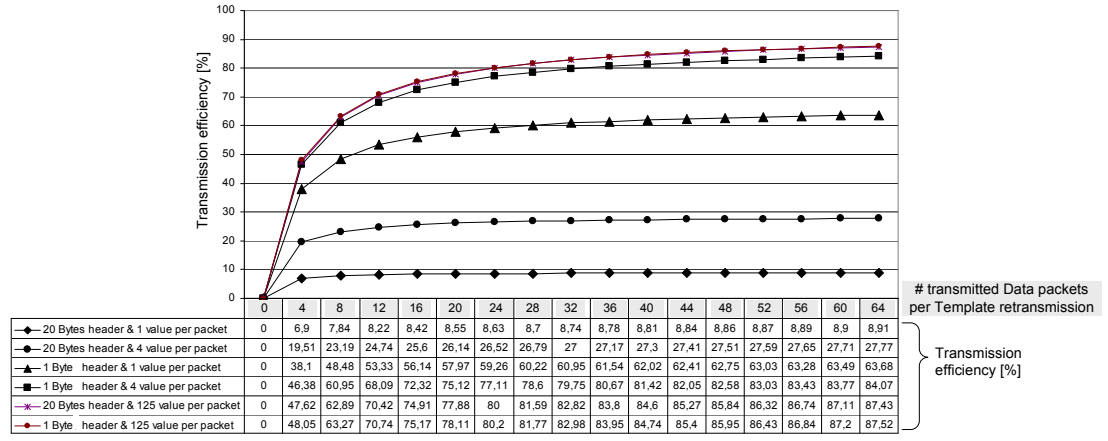
	0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64
20 Bytes header & 1 value per packet	0	6,9	7,84	8,22	8,42	8,55	8,63	8,7	8,74	8,78	8,81	8,84	8,86	8,87	8,89	8,9	8,91
20 Bytes header & 4 value per packet	0	19,51	23,19	24,74	25,6	26,14	26,52	26,79	27	27,17	27,3	27,41	27,51	27,59	27,65	27,71	27,77
3 Bytes header & 1 value per packet	0	25,81	31,37	33,8	35,16	36,04	36,64	37,09	37,43	37,7	37,91	38,1	38,25	38,38	38,49	38,59	38,67
3 Bytes header & 4 value per packet	0	40,51	52,03	57,49	60,66	62,75	64,21	65,31	66,15	66,82	67,37	67,82	68,21	68,53	68,82	69,06	69,28
20 Bytes header & 125 value per packet	0	47,62	62,89	70,42	74,91	77,88	80	81,59	82,82	83,8	84,6	85,27	85,84	86,32	86,74	87,11	87,43
3 Bytes header & 125 value per packet	0	48,01	63,23	70,7	75,15	78,09	80,18	81,75	82,96	83,93	84,72	85,38	85,94	86,42	86,83	87,19	87,51

transmitted Data packets per Template retransmission

Transmission efficiency [%]

Figure E.2: TinyIPFIX transmission efficiency - Case 2

Figure E.3 illustrates the results of case 3. Here TinyIPFIX packets with default header of size 20 bytes are compared to packets with header supporting aggressive approach with one bytes size [98].



	0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64
20 Bytes header & 1 value per packet	0	6,9	7,84	8,22	8,42	8,55	8,63	8,7	8,74	8,78	8,81	8,84	8,86	8,87	8,89	8,9	8,91
20 Bytes header & 4 value per packet	0	19,51	23,19	24,74	25,6	26,14	26,52	26,79	27	27,17	27,3	27,41	27,51	27,59	27,65	27,71	27,77
1 Byte header & 1 value per packet	0	38,1	48,48	53,33	56,14	57,97	59,26	60,22	60,95	61,54	62,02	62,41	62,75	63,03	63,28	63,49	63,68
1 Byte header & 4 value per packet	0	46,38	60,95	68,09	72,32	75,12	77,11	78,6	79,75	80,67	81,42	82,05	82,58	83,03	83,43	83,77	84,07
20 Bytes header & 125 value per packet	0	47,62	62,89	70,42	74,91	77,88	80	81,59	82,82	83,8	84,6	85,27	85,84	86,32	86,74	87,11	87,43
1 Byte header & 125 value per packet	0	48,05	63,27	70,74	75,17	78,11	80,2	81,77	82,98	83,95	84,74	85,4	85,95	86,43	86,84	87,2	87,52

transmitted Data packets per Template retransmission

Transmission efficiency [%]

Figure E.3: TinyIPFIX transmission efficiency - Case 3

Throughout all three analysis cases it can be observed that the header size has a big impact on $t_{eff-TinyIPFIX}$. The impact lowers if the payload becomes larger. The effect of header compression remains important for the performance. Usually in sensor scenarios, a Data Record consists of four values and 16 transmissions between two Template Record transmissions. [98]

F. Abbreviations

6LoWPAN	IPv6 over Low power Wireless Personal Area Network
ACS	Access Control Server
ACU	Aggregation Control Unit
AES	Advanced Encryption Standard
AFU	Aggregation Function Unit
AIDA	Adaptive Application-Independent Data Aggregation
AutHoNe	Autonomic Home Networking
AVG	Average
BLIP	Berkeley Low-Power IP
CA	Certificate Authority
CBC	Cipher Block Chaining
COAP	Constrained Application Protocol
CON	Confirmable message
CPS	Cyber-Physical System
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSR	Certificate Signing Request
DoA	Degree of Aggregation
DES	Data Encryption Standard
DTLS	Datagram Transport Layer Security
DYN	Dynamic
EBHTTP	Embedded binary HTTP
ECC	Elliptic Curve Cryptography
EEPROM	Electrically Erasable Programmable ROM
EID	Enterprise Number
GPS	Global Positioning System
GUI	Graphical User Interface
HMAC	Hash-based Message Authentication Code
HTTP	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Number Authority
ICMPv6	Internet Control Message Protocol for the Internet Protocol Version 6
ICT	Information and Communication Technologies
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force

IoT	Internet of Things
IP	Internet Protocol
IPFIX	IP Flow Information Export
IPsec	Internet Protocol Security
IPv6	Internet Protocol Version 6
ITU	International Telecommunication Union
JSON	JavaScript Object Notation
LAN	Local Area Network
LED	Light-Emitting Diode
MAC	Media Access Control
MAX	Maximum
MIN	Minimum
MTU	Maximum Transmission Unit
NIST	National Institute of Standards and Technology
PAN	Personal Area Network
PDA	Personal Digital Assistant
PIKE	Peer Intermediaries for Key Establishment
PK	Public Key
PKI	Public Key Infrastructure
QoS	Quality of Service
RAM	Random-Access Memory
ROM	Read-Only-Memory
REST	Representational State Transfer
RF	Radio Frequency
RFC	Request For Comments
RFID	Radio Frequency Identification
RSA	Rivest, Shamir and Adleman
RST	Reset message
SCTP	Stream Control Transmission Protocol
SHA	Secure Hash Algorithm
SHM	Structural Health Monitoring
SIA	Secure Information Aggregation
SSL	Secure Sockets Layer
sMAP	Simple Measurement and Actuation Profile
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TAG	Tiny AGregation service
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TPM	Trusted Platform Module
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WBAN	Wireless Body Area Networks
WLAN	Wireless Local Area Network
WoT	Web of Things
WSN	Wireless Sensor Network
XML	Extensible Markup Language

Bibliography

[1] I. Strategy and P. Unit, "ITU Internet Reports 2005: The Internet of Things," *Geneva: International Telecommunication Union (ITU)*, 2005.

[2] "Postscapes - Tracking the Internet of Things." http://www.postscapes.com/internet-of-things-history, 2012.

[3] D. Guinard and V. Trifa, "Towards the web of things: Web mashups for embedded devices," in *In Proceedings of the International World Wide Web Conferences - Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web*, MEM, April 2009.

[4] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. Jubert, M. Mazura, M. Harrison, M. Eisenhauer10, *et al.*, "Internet of things strategic research roadmap," *Internet of Things: Global Technological and Societal Trends*, p. 9, 2009.

[5] D. Guinard, V. Trifa, and E. Wilde, "A Resource Oriented Architecture for the Web of Things," in *Proceedings of Internet of Things International Conference*, IEEE, November 2010.

[6] B. Warneke, M. Last, B. Liebowitz, and K. Pister, "Smart dust: Communicating with a cubic-millimeter computer," *Computer*, vol. 34, no. 1, pp. 44–51, 2001.

[7] J. Sen, "A Survey on Wireless Sensor Network Security," *International Journal of Communication Networks and Information Security*, vol. 1, August 2009.

[8] J. Walters, Z. Liang, W. Shi, and V. Chaudhary, "Wireless Sensor Network Security: A Survey," *Security in Distributed, Grid, Mobile, and Pervasive Computing*, p. 367, 2007.

[9] Y. Wang, G. Attebury, and B. Ramamurthy, "A Survey of Security Issues in Wireless Sensor Networks," *Communications Surveys Tutorials, IEEE*, vol. 8, no. 2, pp. 2–23, 2006.

[10] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. Wiley-Interscience, 2007.

[11] C. Schmitt and G. Carle, "Applications for Wireless Sensor Networks," in *Chapter in Book: Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications, Antonopoulos N.; Exarchakos G.; Li M.; Liotta A. (Eds.), Information Science Publishing*, January 2010.

[12] M. Broy, M. Cengarle, and E. Geisberger, "Cyber-Physical Systems: Imminent Challenges," *Large-Scale Complex IT Systems. Development, Operation and Management*, pp. 1–28, 2012.

[13] Autonomic Home Networking DE Project Page. http://www.authone.de, 2012.

[14] W. Wang, R. O'Keeffe, N. Wang, M. Hayes, B. O'Flynn, Ó. Mathúna, and S. Cian, "Practical Wireless Sensor Networks Power Consumption Metrics for Building Energy Management Applications," in *23rd European Conference Forum Bauinformatik - Construction Informatics*, September 2011.

[15] Y. Panthachai and P. Keeratiwintakorn, "An Energy Model for Transmission in Telos-based Wireless Sensor Networks," in *Proceedings of the International joint Conference on Computer Science & Software Engineering*, JCSSE, pp. 146–150, May 2007.

[16] K. Sha and W. Shi, "Modeling the lifetime of wireless sensor networks," *Sensor Letters*, vol. 3, no. 2, pp. 126–135, 2005.

[17] D. Dessales, A. Poussard, R. Vauzelle, N. Richard, F. Gaudairek, and C. Martinsons, "Transmission Power Adaptation According to the Message Length for Wireless Sensor Networks," in *IEEE 22nd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, pp. 46–50, IEEE, April 2011.

[18] C. A. SmartRF, "CC2420 Preliminary Datasheet (rev 1.2)," tech. rep., 2004.

[19] B. Claise, S. Bryant, G. Sadasivan, S. Leinen, T. Dietz, and B. Trammell, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information (RFC 5101)," tech. rep., The Internet Engineering Task Force (IETF), January 2008.

[20] T. Kothmayr, C. Schmitt, L. Braun, and G. Carle, "Gathering Sensor Data in Home Networks with IPFIX," in *Proceedings of the 7th European conference on Wireless Sensor Networks*, EWSN, (Berlin, Heidelberg), pp. 131–146, Springer-Verlag, February 2010.

[21] G. Wagenknecht, M. Anwander, and T. Braun, "Hop-to-Hop Reliability in IP-Based Wireless Sensor Networks-A Cross-Layer Approach," *Wired and Wireless Internet Communications*, pp. 61–72, 2009.

[22] T. Dierks and C. Allen, "The TLS Protocol Version 1.0," tech. rep., The Internet Engineering Task Force (IETF), January 1999.

[23] N. Modadugu and E. Rescorla, "The Design and Implementation of Datagram TLS," in *Proceedings of the 11th Annual Network and Distributed System Security Symposium*, NDSS, August 2004.

[24] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Sensor Networks," *Communications Magazine, IEEE*, vol. 40, pp. 102–114, August 2002.

[25] UC Berkeley, "TinyOS Homepage." http://www.tinyos.net/, 2012.

[26] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System Architecture Directions for Networked Sensors," *ACM SIGPLAN Notices*, vol. 35, pp. 93–104, November 2000.

[27] J. Hill and D. Culler, "Mica: A Wireless Platform for Deeply Embedded Networks," *Micro, IEEE*, vol. 22, pp. 12–24, November-December 2002.

[28] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Structural Health Monitoring of the Golden Gate Bridge." http://www.cs.berkeley.edu/ binetude/ggb/.

[29] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems," *ACM SIGPLAN Notices*, vol. 38, pp. 1–11, May 2003.

[30] TIK WSN Research Group @ ETH Zurich, "The Sensor Network Museum." http://www.snm.ethz.ch/Main/HomePage, 2012.

[31] Crossbow Technologies Inc. http://www.xbow.com/, 2012.

[32] Advantic Sistemas Y Servicios S.L. http://www.advanticsys.com, 2012.

[33] R. Jurdak, K. Klues, B. Kusy, C. Richter, K. Langendoen, and M. Brunig, "Opal: A Multiradio Platform for High Throughput Wireless Sensor Networks," *Embedded Systems Letters, IEEE*, vol. 3, pp. 121–124, December 2011.

[34] S. Pearson, "Trusted Computing Platforms, The Next Security Solution," *HP Labs*, 2002.

[35] A. Sinha and A. Chandrakasan, "Dynamic Power Management in Wireless Sensor Networks," *Design Test of Computers, IEEE*, vol. 18, pp. 62–74, March-April 2001.

[36] W. Seah, Z. Eu, and H. Tan, "Wireless sensor networks powered by ambient energy harvesting (wsn-heap)-survey and challenges," in *Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology, 2009. Wireless VITAE 2009. 1st International Conference on*, pp. 1–5, IEEE, 2009.

[37] M. Farooq and T. Kunz, "Operating Systems for Wireless Sensor Networks: A Survey," *Sensors*, vol. 11, no. 6, pp. 5900–5930, 2011.

[38] W. Dong, C. Chen, X. Liu, and J. Bu, "Providing OS Support for Wireless Sensor Networks: Challenges and Approaches," *Communications Surveys & Tutorials, IEEE*, vol. 12, no. 4, pp. 519–530, 2010.

[39] "Contiki - The Open Source OS for the Internet of Things." http://www.contiki-os.org/, 2012.

[40] E. Brewer, D. Culler, D. Gay, P. Levis, R. von Behren, and M. Welsh, "nesC: A Programming Language for Deeply Networked Systems." http://nescc.sourceforge.net/, 2004.

[41] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN, (Washington, DC, USA), pp. 455–462, IEEE Computer Society, 2004.

[42] Freie Universität Berlin, "ScatterWeb." http://cst.mi.fu-berlin.de, 2012.

[43] C. Buratti, A. Conti, D. Dardari, and R. Verdone, "An Overview on Wireless Sensor Networks Technology and Evolution," *Sensors*, vol. 9, no. 9, pp. 6869–6896, 2009.

[44] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Wireless Sensor Networks for Structural Health Monitoring," in *Proceedings of the 4th International Conference on Embedded networked Sensor Systems*, SenSys, pp. 427–428, ACM, November 2006.

[45] A. Manson and T. Addington, "Emergency Alert System," April 2003. US Patent 6,543,051.

[46] NOAA's National Weather Service, "Pacific Tsunami Warning Center." http://ptwc.weather.gov/?region=1, 2012.

[47] F. Ramirez and P. Perez, "The Local Tsunami Alert System ["SLAT"]: A Computational Tool for the Integral Management of a Tsunami Emergency," *Natural Hazards*, vol. 31, no. 1, pp. 129–142, 2004.

[48] "SAFER - Seismic eArly warning For EuRope." http://www.saferproject.net/, 2012.

[49] J. Khan and M. Yuce, "Wireless Body Area Network (WBAN) for Medical Applications," *New Developments in Biomedical Engineering. INTECH*, 2010.

[50] M. Welsh, S. Moulton, T. Fulford-Jones, and D. Malan, "CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care," in *International Workshop on Wearable and Implantable Body Sensor Networks, April, London, UK*, BSN, April 2004.

[51] M. Winkler, K. Tuchs, K. Hughes, and G. Barclay, "Theoretical and Practical Aspects of Military Wireless Sensor Networks," *Journal of Telecommunications and Information Technology*, vol. 2, pp. 37–45, 2008.

[52] M. Hussain, P. Khan, and K. K. Sup, "WSN research activities for military application," in *Proceedings of the 11th International Conference on Advanced Communication Technology*, vol. 01 of *ICACT*, pp. 271–274, February 2009.

[53] M. Becker, B. Wenning, C. Görg, R. Jedermann, and A. Timm-Giel, "Logistic Applications with Wireless Sensor Networks," in *Proceeding of the Workshop on Embedded Networked Sensors*, vol. 2010 of *HotEmNets*, Citeseer, 2010.

[54] A. Garcia-Sanchez, F. Garcia-Sanchez, F. Losilla, P. Kulakowski, J. Garcia-Haro, A. Rodríguez, J. López-Bao, and F. Palomares, "Wireless Sensor Network Deployment for Monitoring Wildlife Passages," *Sensors*, vol. 10, no. 8, pp. 7236–7262, 2010.

[55] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-efficient Computing for Wildlife Tracking: Design Tradeoffs and early Experiences with ZebraNet," *SIGOPS Operating Systems Review*, vol. 36, pp. 96–107, October 2002.

[56] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi, "Hardware Design Experiences in ZebraNet," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys, (New York, NY, USA), pp. 227–238, ACM, November 2004.

[57] UvA Bird Tracking System. http://www.uva-bits.nl/, 2012.

[58] M. Swan, "Sensor Mania! The Internet of Things, Wearable Computing, Objective Metrics, and the Quantified Self 2.0," *Journal of Sensor and Actuator Networks*, vol. 1, no. 3, pp. 217–253, 2012.

[59] H. LeHong, "Hype cycle for the internet of things," tech. rep., Gartner Inc., 2012.

[60] S. Kent and K.Seo, "Security Architecture for the Internet Protocol," Tech. Rep. 4301, The Internet Engineering Task Force (IETF), December 2005.

[61] S. Kent, "IP Authentication Header," Tech. Rep. 4302, The Internet Engineering Task Force (IETF), December 2005.

[62] S. Kent, "IP Encapsulating Security Payload (ESP)," Tech. Rep. 4303, The Internet Engineering Task Force (IETF), December 2005.

[63] J. Hui and D. Culler, "IPv6 in Low-Power Wireless Networks," *Proceedings of the IEEE*, vol. 98, pp. 1865 –1878, November 2010.

[64] ZigBee Alliance. http://www.zigbee.org/, 2012.

[65] G. Mulligan, "The 6LoWPAN Architecture," in *Proceedings of the 4th Workshop on Embedded Networked Sensors*, EmNets, (New York, NY, USA), pp. 78–82, ACM, 2007.

[66] "IPv6 over Low power WPAN Working Group IETF." http://tools.ietf.org/wg/6lowpan/, 2005.

[67] M. Harvan, "Connecting Wireless Sensor Networks to the Internet - A 6lowpan Implementation for TinyOS 2.0," Master's thesis, Jacobs University Bremen, School of Engineering and Science, May 2007.

[68] N. Kushalnagar, G. Montenegro, and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals - RFC 4919," tech. rep., The Internet Engineering Task Force (IETF), August 2007.

[69] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "IPv6 over Low Power Wireless Personal Area Networks (6LowPAN) - RFC 4944," tech. rep., The Internet Engineering Task Force (IETF), September 2007.

[70] S. Dawson-Haggerty, "Design, Implementation, and Evaluation of an Embedded IPv6 Stack," Master's thesis, Master of Science in Computer Science - Electrical Engineering and Computer Sciences, University of California, Berkeley (USA), 2010.

[71] Berkeley WEBS, "Project Page for blip, the Berkeley IP implementation for low-power networks." http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip, 2012.

[72] R. Silva, J. Silva, and F. Boavida, "Evaluating 6lowPAN implementations in WSNs," *Proceedings of 9th Conferncia sobre Redes de Computadores Oeiras, Portugal*.

[73] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," tech. rep., The Internet Engineering Task Force (IETF), December 1998.

[74] Internet Assigned Numbers Authority. http://www.iana.org/, 2012.

[75] Z. Shelby and C. Bormann, *6LoWPAN: The Wireless Embedded Internet*, vol. 43. Wiley, 2011.

[76] R. Shirey, "Internet Security Glossary - RFC 2828," Tech. Rep. 2828, The Internet Engineering Task Force (IETF), May 2000.

[77] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment*. Springer, 2003.

[78] C. Kaufman, R. Perlman, and M. Speciner, *Network Security: Private Communication in a Public World*. Prentice Hall Press, 2002.

[79] D. Dolev and A. Yao, "On the Security of Public Key Protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[80] C. Liedl, "Security Analysis for very Constrained Objects based on Pre-Shared Keys," Bachelor Thesis, Department of Computer Science, Technische Universität München, October 2012.

[81] T. Kothmayr, "A Security Architecture for Wireless Sensor Networks based on DTLS," Master Thesis, Software Engineering Elite Graduate Program, Department of Computer Science, Universität Augsburg, December 2011.

[82] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "NIST Special Publication 800-57: Recommendation for Key Management - Part 1: General (Revision 3)," *NIST Special Publication*, vol. 800, no. 57, pp. 1–142, 2007.

[83] J. Großschädl, S. Tillich, C. Rechberger, M. Hofmann, and M. Medwed, "Energy Evaluation of Software Implementations of Block Ciphers under Memory Constraints," in *Proceedings of the conference on Design, Automation and Test in Europe*, DATE, pp. 1110–1115, EDA Consortium, April 2007.

[84] T. Kleinjung, K. Aoki, J. Franke, A. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. Montgomery, D. Osvik, *et al.*, "Factorization of a 768-bit RSA modulus," *Advances in Cryptology*, pp. 333–350, 2010.

[85] B. Kaliski, "PKCS# 1: RSA encryption version 1.5 - RFC 2313," tech. rep., The Internet Engineering Task Force (IETF), March 1998.

[86] V. Shoup, "Oaep reconsidered," in *Advances in Cryptology*, CRYPTO, pp. 239–259, Springer, 2001.

[87] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern, "RSA-OAEP is secure under the RSA assumption," in *Advances in Cryptology*, CRYPTO, pp. 260–274, Springer, 2001.

[88] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. MIT press, 2001.

[89] C. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (RFC 5280)," tech. rep., The Internet Engineering Task Force (IETF), May 2008.

[90] T. Stephen, *SSL and TLS Essentials. Securing the Web.* Wiley, 2000.

[91] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security - RFC 4347," tech. rep., The Internet Engineering Task Force (IETF), April 2006.

[92] D. Challener, K. Yoder, R. Catherman, D. Safford, and L. Van Doorn, *A Practical Guide to Trusted Computing.* IBM press, 2007.

[93] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, 2000.

[94] L. Richardson and S. Ruby, *RESTful Web Services.* O'Reilly Media, May 2007.

[95] Z. Shelby, M. Garrison Stuber, D. Sturek, B. Frank, and R. Kelsey, "CoAP Requirements and Features," internet-draft (work in progress), IETF, February 2010.

[96] Z. Shelby, K. Hartke, C. Borman, and B. Frank, "Constrained Application Protocol (CoAP)," internet-draft (work in progress), IETF, November 2011.

[97] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler, "sMAP: A Simple Measurement and Actuation Profile for Physical Information," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys, (New York, NY, USA), pp. 197–210, ACM, November 2010.

[98] T. Kothmayr, "Data collection in Wireless Sensor Networks for Autonomic Home Networking," Bachelor Thesis, Department of Computer Science, Technische Universität München, January 2010.

[99] S. Dawson-Haggerty, A. Tavakoli, and D. Culler, "Hydro: A Hybrid Routing Protocol for Low-Power and Lossy Networks," in *First IEEE International Conference on Smart Grid Communications*, SmartGridComm, pp. 268 –273, October 2010.

[100] Maxwell Dworkin Laboratory, "MoteLab: Harvard Sensor Network Testbed." http://motelab.eecs.harvard.edu/, 2012.

[101] C. Schmitt, L. Braun, and G. Carle, "IPFIX for Wireless Sensors," internet-draft, The Internet Engineering Task Force (IETF), October 2009.

[102] B. Ertl, "Data Aggregation using TinyIPFIX in Wireless Sensor Networks," Bachelor Thesis, Department of Computer Science, Technische Universität München, August 2011.

[103] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 131–146, 2002.

[104] T. He, B. M. Blum, J. A. Stankovic, and T. Abdelzaher, "AIDA: Adaptive Application Independent Data Aggregation in Wireless Sensor Networks," *ACM Transactions on Embedded Computing System, Special Issue on Dynamically Adaptable Embedded Systems*, vol. 3, pp. 426–457, May 2004.

[105] B. Przydatek, D. Song, and A. Perrig, "SIA: secure information aggregation in sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys, (New York, NY, USA), pp. 255–265, ACM, November 2003.

[106] B. Krishnamachari, D. Estrin, and S. B. Wicker, "The Impact of Data Aggregation in Wireless Sensor Networks," in *Proceedings of the 22nd International Conference on Distributed Computing Systems*, ICDCSW, (Washington, DC, USA), pp. 575–578, IEEE Computer Society, 2002.

[107] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IPv6 - RFC 4861," tech. rep., The Internet Engineering Task Force (IETF), September 2007.

[108] A. Woo, T. Tong, and D. Culler, "Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys, (New York, NY, USA), pp. 14–27, ACM, November 2003.

[109] A. Conta and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification - RFC 4861," tech. rep., The Internet Engineering Task Force (IETF), March 2006.

[110] D. Carmen, P. Kruus, and B. Matt, "Constraints and Approaches for Distributed Sensor Network Security," darpa project report, NAI Labs, 2002.

[111] R. Watro, D. Kong, S.-f. Cuti, C. Gardiner, C. Lynn, and P. Kruus, "TinyPK: Securing Sensor Networks with Public Key Technology," in *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, SASN, (New York, NY, USA), pp. 59–64, ACM, October 2004.

[112] A. Liu and P. Ning, "TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks," in *Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, IPSN, (Washington, DC, USA), pp. 245–256, IEEE Computer Society, April 2008.

[113] C. Karlof, N. Sastry, and D. Wagner, "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, SenSys, (New York, NY, USA), pp. 162–175, ACM, November 2004.

[114] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. Culler, "SPINS: Security Protocols for Sensor Networks," *Wireless networks*, vol. 8, no. 5, pp. 521–534, 2002.

[115] Y. Law and M. Palaniswami, "Key Management in Wireless Sensor Networks," in *Guide to Wireless Sensor Networks* (S. C. Misra, I. Woungang, and S. Misra, eds.), Computer Communications and Networks, pp. 513–531, Springer London, 2009.

[116] J. Lee, V. Leung, K. Wong, J. Cao, and H. Chan, "Key Management Issues in Wireless Sensor Networks: Current Proposals and Future Developments," *Wireless Communications*, vol. 14, no. 5, pp. 76–84, 2007.

[117] Y. Xiao, V. K. Rayi, B. Sun, X. Du, F. Hu, and M. Galloway, "A Survey of Key Management Schemes in Wireless Sensor Networks," *Computer Communications*, vol. 30, pp. 2314–2341, September 2007.

[118] H. Chan and A. Perrig, "PIKE: peer intermediaries for key establishment in sensor networks," in *Proceedings of the 24th Annual Joint Conference of the*

IEEE Computer and Communications Societies, vol. 1 of *INFOCOM*, pp. 524–535, March 2005.

[119] L. Eschenauer and V. D. Gligor, "A Key-Management Scheme for Distributed Sensor Networks," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS, (New York, NY, USA), pp. 41–47, ACM, November 2002.

[120] H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes for Sensor Networks," in *Proceedings of Symposium on Security and Privacy*, SP, pp. 197–213, May 2003.

[121] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle, "A DTLS based End-to-End Security Architecture for the Internet of Things with two-way Authentication," in *37th IEEE International Workshop on Practical Issues in Building Sensor Network Applications*, LCN, October 2012.

[122] S. Raza, T. Chung, S. Duquennoy, T. Voigt, U. Roedig, *et al.*, "Securing Internet of Things with Lightweight IPsec," Tech. Rep. T2010:08, Swedish Institute of Computer Science (SICS), August 2010.

[123] A. Sardouk, L. Merghem-Boulahia, R. Rahim-Amoud, and D. Gaiti, "Data Aggregation in WSNS: A Survey," *Materials Science Research Journal*, vol. 4, pp. 327–370, 2010.

[124] V. Gupta, M. Millard, S. Fung, Y. Zhu, N. Gura, H. Eberle, and S. C. Shantz, "Sizzle: A Standards-Based End-to-End Security Architecture for the Embedded Internet," in *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications*, PERCOM, (Washington, DC, USA), pp. 247–256, IEEE Computer Society, April 2005.

[125] W. Jung, S. Hong, M. Ha, Y.-J. Kim, and D. Kim, "SSL-Based Lightweight Security of IP-Based Wireless Sensor Networks," *International Conference on Advanced Information Networking and Applications (Workshops)*, vol. 0, pp. 1112–1117, May 2009.

[126] S. Fouladgar, B. Mainaud, K. Masmoudi, and H. Afifi, "Tiny 3-TLS: A Trust Delegation Protocol for Wireless Sensor Networks," in *Proceedings of the 3rd European conference on Security and Privacy in Ad-Hoc and Sensor Networks*, ESAS, (Berlin, Heidelberg), pp. 32–42, Springer-Verlag, September 2006.

[127] T. Kothmayr, W. Hu, C. Schmitt, M. Brünig, and G. Carle, "Securing the Internet of Things with DTLS," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (Poster Session)*, SenSys, (Seattle, USA), November 2011.

[128] W. Hu, P. Corke, W. C. Shih, and L. Overs, "secFleck: A Public Key Technology Platform for Wireless Sensor Networks," in *Proceedings of the 6th European Conference on Wireless Sensor Networks*, EWSN, (Berlin, Heidelberg), pp. 296–311, Springer-Verlag, February 2009.

[129] B. Parbat, A. K. Dwivedi, and O. P. Vyas, "Data Visualization Tools for WSNs: A Glimpse," *International Journal of Computer Applications*, vol. 2, pp. 14–20, May 2010. Published By Foundation of Computer Science.

[130] B. Titzer, D. Lee, and J. Palsberg, "Avrora: Scalable Sensor Network Simulation with Precise Timing," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, IPSN, pp. 477–482, IEEE, April 2005.

[131] M. E. Miyashita, "TosGUI: TinyOS Graphical Simulation Project Summary," tech. rep., Kent State University, 2002.

[132] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: accurate and scalable simulation of entire TinyOS applications," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys, (New York, NY, USA), pp. 126–137, ACM, November 2003.

[133] A. Freitag, "Framework Development for Wireless Sensor Networks facing Conficuration and Information Exchange/Export Tasks," Bachelor Thesis, Department of Computer Science, Technische Universität München, May 2012.

[134] The OpenSSL Project , "OpenSSL - Cryptography and SSL/TLS Toolkit." http://www.openssl.org/, 2012.

[135] J. Jonsson and B. Kaliski, "Public-Key Cryptography Standards (PKCS)# 1: RSA Cryptography Specifications Version 2.1," Tech. Rep. 3447, The Internet Engineering Task Force (IETF), February 2003.

[136] COSM Connect to your world. http://www.cosm.com, 2012.

[137] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny Web Services: Design and Implementation of Interoperable and Evolvable Sensor Networks," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys, (New York, NY, USA), pp. 253–266, ACM, 2008.

List of Figures

List of Tables

ISBN 3-937201-36-X
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)
DOI: 10.2313/NET-2013-07-2