A Framework for WirelessHART Simulations

Igor Konovalov < *igor@sics.se* >

Swedish Institute of Computer Science
Box 1263, SE-164 29 Kista, Sweden

SICS Technical Report T2010:06
ISSN 1100-3154

June 15, 2010

Keywords: wirelessHART, hybrid simulations, time synchronization, protocol analyzer

Abstract

Due to stringent timing requirements of the *WirelessHART* protocol, we need to extend previously known hybrid simulation approaches with WirelessHART specific functionality. Because of intermediary devices that connect simulation and real environments a communication delay significantly exceeds time boundaries of a *WirelessHART* slot. Our solution is to add a *WirelessHART*-enabled intermediary device operating at the Physical and the Data Link layers that can handle time critical tasks of the *WirelessHART* protocol. This approach allows to evaluate the performance of a *WirelessHART* network and *WirelessHART*-enabled devices in a hybrid simulation environment.

Acknowledgements

I would like to thank my advisors Fredrik Ősterlind from SICS and Jonas Neander from ABB for their support and kind advices during my work. I would also like to thank all the Networked Embedded Systems group and my examiner Thomas Lindh from the Royal Institute of Technology.

This thesis has been performed within the SICS Center for Networked Systems funded by ABB.

Abbreviations

ACK	Acknowledgment
APDU	Application Layer Protocol Data Unit
ASN	Absolute Slot Number
CCM	Counter with CBC-MAC
DLPDU	Data Link Layer Protocol Data Unit
HART	Highway Addressable Remote Transducer
IETF	Internet Engineering Task Force
ISM	Industrial, Scientific and Medical
MAC	Medium Access Control
MIC	Message Integrity Code
NPDU	Network Layer Protocol Data Unit
OSI Model	Open Systems Interconnection Basic Reference Model
PPDU	Physical Layer protocol Data Unit
TDMA	Time Division Multiple Access
TPDU	Transport Layer Protocol Data Unit
WSN	Wireless Sensor Network

Contents

List of Figures

Chapter 1

Introduction

WirelessHART is an emerging communication protocol for industrial automation. It is an extension of the widely used Highway Addressable Remote Transducer (HART) communication protocol [1]. On April 2010 *WirelessHART* was approved by the International Electrotechnical Commission (IEC) as a first international standard for industry process automation [2].

Existing deployments of legacy HART networks encourages the development and adoption of *WirelessHART* as an extension to a HART infrastructure. *WirelessHART* is a secure and reliable communication protocol operating at the IEEE 802.15.4 2.4 GHz wireless Physical Layer that uses Time Division Multiple Access (TDMA) Media Access Control (MAC) with frequency hopping. The protocol is loosely organized around the OSI-7 interconnection model and requires *WirelessHART*-enabled devices from different vendors to be interoperable[3].

1.1 Problem Statement

Typical hybrid simulation tools are not suitable for simulating *WirelessHART* networks, and thus an enhanced approach for performing hybrid simulations is necessary. Time constrained and secure communications of the *WirelessHART* protocol impose additional requirements that need to be considered when simulating a *WirelessHART* network in a hybrid simulation environment.

Simulations of *WirelessHART* networks present an efficient and cost-effective way to perform network evaluations. For a better understanding of the *WirelessHART* protocol a feasible tool suitable for testing capabilities of *WirelessHART*-enabled devices and a *WirelessHART* mesh network is required.

The goal of the project is to create a simulation tool that can be used to test *WirelessHART*-enabled devices and simulate a *WirelessHART* network. This thesis main goal is to create an interface to connect a simulation environment with real *WirelessHART*-enabled devices and be able to analyze a network traffic and an operation of a network.

The major challenges for this project are constraints imposed by both the hardware and the *WirelessHART* protocol. These constraints include the time critical operation of the protocol, protocol security-enabled communications and a low computational power and memory of the hardware.

The software should be designed to run on wireless sensor nodes and must depend only on the Physical Layer employed. A wireless sensor node shall be capable of a transparent data forwarding from the wireless medium to the simulated environment and vice versa. Devices running the *WirelessHART* protocol stack demand considerable amount of computations. A typical wireless sensor node is best suited for low computational complexity applications and low memory consumption. Hence most of *WirelessHART* capabilities should be implemented outside a sensor node.

1.2 Method

The method used in this thesis is experimental computer science. We start with the implementation of a generic framework and follow with a literature survey of the *WirelessHART* protocol and then proceed to the implementation of additional protocol related features. An evaluation of the implementation is verified periodically using both available literature sources and developed analyzing tools.

1.3 Alternative Approaches

Among the alternative solutions related to this thesis, a software simulation is an another approach. This approach requires a feasible *WirelessHART* stack however currently there is no full implementation of it. A different approach is to perform a non-real time simulation employing an event-based simulation behavior, but this approach requires a suitable event-based simulation framework.

1.4 Scientific Contributions

There are two contributions of this thesis. First a hybrid simulation framework for the *WirelessHART* protocol is presented. Secondly, we show that it is possible to perform the time critical computations on a constrained embedded platform without exceeding the time boundaries imposed by the *WirelessHART* protocol.

1.5 Delimitations

Due to lack of a complete and stable implementation of the *WirelessHART* stack, the evaluation has been done using a Tmote Sky [4] sensor node that is configured with relevant *WirelessHART* capabilities. The Tmote Sky sensor node was chosen based on its popularity and an extensive support. Since most of the *WirelessHART* requirements addressed in this thesis are at the Physical and the Data Link layers, the results are expected to be accurate enough.

1.6 Thesis Structure

The reminder of this thesis has been structured as follows. In Chapter 2, we will describe some important properties of the *WirelessHART* protocol that has to be considered in the implementation. After this we describe software tools that aid in development of the simulation framework. The chapter finishes with an evaluation of the protocol emphasizing pros and cons of the protocol. The design and implementation is described in Chapter 3 and the evaluation in Chapter 4. Chapter 5 will discuss related work and in Chapter 6 the thesis is concluded.

Chapter 2

Background

Software development tools such as Contiki operating system and COOJA network simulator alongside with the hardware tools including Tmote Sky sensor node and the D2510 Network Manager are introduced in this chapter. Also the layered structure of the *WirelessHART* protocol is described.

2.1 Software Development Tools

In this thesis Contiki operating system[5] is used to develop an application interfacing the 2.4 GHz wireless medium. Contiki utilizes an event-driven kernel, communication stacks and a stackless thread-like abstraction - protothreads[6]. Contiki operating system is an actively developed operating system ported to a number of wireless sensor platforms including a Tmote Sky[7] wireless sensor node. Contiki operating system supports different Media Access Protocols (MACs) and provides three communication stacks - Rime[8], uIP[9] and uIPv6[10]. Contiki operating system is developed using C programming language.

COOJA network simulator[11] is used for developing the simulation framework and *WirelessHART* related applications. COOJA is a cross-level simulation tool that enables simulations at different abstraction levels - the network level, the operating system level, and the machine code level. COOJA is developed using Java programming language, thus it is a cross-platform simulator that can run independently of the underlying hardware architecture.

2.2 Hardware Development Tools

In this thesis wireless sensors are used for a physical connection between *WirelessHART*-enabled devices and the simulation framework. Wireless sensors are typically capable of sensing different physical phenomenon such as temperature, humidity, light and other. In this thesis wireless sensors are used mainly as radio transceivers.

2.2.1 Tmote Sky

The Tmote Sky sensor node (figure 2.1) is a real-time computing constraint embedded system[12] manufactured by Moteiv corporation. It is a low-power wireless sensor node that features an MSP430 microcontroller[13] from Texas Instruments with 10 kilobytes of RAM, 48 kilobytes of internal memory and 1 megabyte of external flash memory.

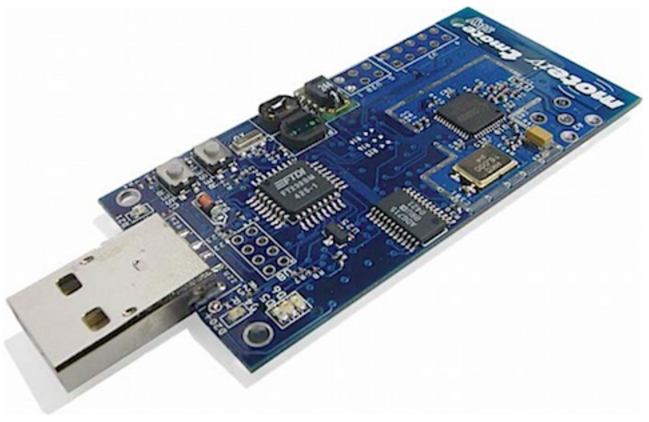


Figure 2.1: The Tmote Sky Sensor Node

The MSP430 is a Reduced Instruction Set Computer (RISC) with an extremely low current consumption both in active and sleep modes thus allowing for an extended battery life. The Tmote Sky uses the IEEE 802.15.4 compliant Chipcon CC2420[14] Radio Transceiver.

The CC2420 is a single-chip 2.4 GHz RF Transceiver with an effective data rate of 250 kbps. The CC2420 RF transceiver provides hardware support for various MAC layer functionalities, including data encryption and data authentication.

2.2.2 The SmartMesh IA-510 D2510 Network Manager

The D2510 Network Manager[15] (figure 2.2) combines the Dust Networks Gateway and the Network Manager for up to 250 SmartMesh IA-510 motes[15]. The D2510 Network Manager is the main unit in a *WirelessHART*-enabled SmartMesh network. Among others, its responsibilities include network configuration and scheduling, packet routing, network maintenance, and data publishing to a wired network.

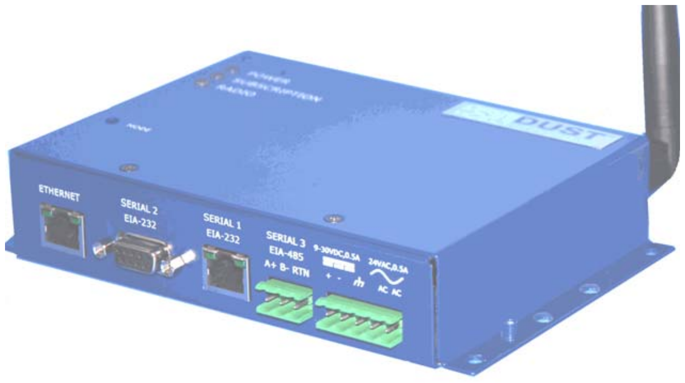


Figure 2.2: The IA-510 D2510 Network Manager

The D2510 Network Manager employs reliable and low latency data communications and a deterministic power management. The operation of the Network Manager can be configured via one of its three maintenance interfaces (i.e. serial, Ethernet or XML). Although the Network Manager allows to configure various features, it is not possible to directly change its operation.

2.2.3 The M2510 Evaluation Mote Module

The M2510 evaluation sensor mote in the figure 2.3 is an ultra low-power wireless transceiver[15]. It can receive serial data from attached sensors (if any) and use an onboard radio to send packets to neighboring motes.

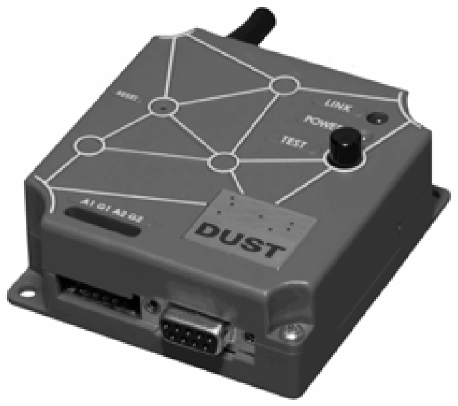


Figure 2.3: The M2510 Evaluation Mote Module

This mote runs the SmartMesh software and forwards data in a series of hops to the Network Manager. In this thesis the M2510 Evaluation Mote Module is used to monitor the *WirelessHART* joining procedure to the Network Manager as well as for further analyses of *WirelessHART* communications.

2.3 The WirelessHART Protocol

The *WirelessHART* protocol is an extension to a legacy HART protocol that adds a wireless capability to its predecessor. The *WirelessHART* protocol main features are summarized below:

- Communication protocol designed for industrial automation
- Extension to HART5, 4-20mA protocol
- Full international standard
- IEEE 802.15.4-2006 Physical Layer
- Secure, reliable, and simple communications
- Frequency hopping TDMA MAC protocol
- Four basic network elements:

1.	Network Manager
2.	Security Manager
3.	Gateway/Access Point
4.	Field Device

- Command-based network management and data communication

WirelessHART has a number of basic capabilities and even greater amount of additional features that can be configured. The reminder of this chapter introduces the *WirelessHART* protocol covering each of its layers which we believe is the most comprehensive way to describe the protocol. The detailed information of capabilities of generic *WirelessHART*-enabled devices such as the Network Manager or the Gateway are not discussed here and an interested reader can refer to a set of *WirelessHART* specifications for an additional information. All of the subsequent sections in this chapter describes the protocol capabilities that are valid to the D2510 Network Manager and the M2510 Field Device.

2.3.1 The WirelessHART Physical Layer

The *WirelessHART* protocol employs the partially adopted[16] IEEE 802.15.4-2006 Physical Layer[17] operating at 2.4 GHz. The Physical Layer PDU (PPDU) (figure 2.4) begins with a synchronization header (SHR). The SHR consists of a preamble sequence followed by a Start of Frame Delimiter (SFD).

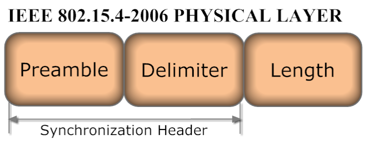


Figure 2.4: WirelessHART Physical PDU Structure

The SHR is always transmitted first for any *WirelessHART* frame. For the 2.4 Ghz Physical Layer the preamble sequence length is 4 bytes and the SFD length is 1 byte. The preamble field is used by the transceiver to obtain chip and symbol synchronization with an incoming message. The SFD field indicates an end of the SHR and a start of a packet data. A length field is 7 bits in length, it specifies the total number of bytes contained in the PPDU payload excluding itself.

2.3.2 The WirelessHART Data-Link Layer

The *WirelessHART* frame or the Data Link Layer Protocol Data Unit (DLPDU)[1] establishes a structure of the *WirelessHART* frame and provides means for reliable and secure communications at the Data-Link Layer (DLL). The DLL resides on top of the IEEE 802.4.15 Physical Layer. The *WirelessHART* DLL differs from the IEEE 802.15.4-2006 DLL introducing frequency hopping and channel blacklisting. The later one can be used to reduce the interference in a presence of a noise that can be introduced by industrial processes.

The *WirelessHART* DLPDU specifies an address format employed, contains a sequence number for every frame transaction and an information about the DLPDU type and priority. With an aid of *WirelessHART* DLPDUs a network of *WirelessHART*-enabled devices is maintained and synchronized in time. The *WirelessHART* DLPDU structure is illustrated in the figure 2.5. Every *WirelessHART* frame (that is any message sent or received) starts with a fixed first byte (0x41) that can be used, if needed, for filtering of *WirelessHART* frames. The first byte alongside with an Address Specifier field is an IEEE 802.15.4 Frame Control Field. Since security is an essential part of the *WirelessHART* protocol, bit four of the first byte is set to indicate that an IEEE 802.15.4-2006 security is enabled.

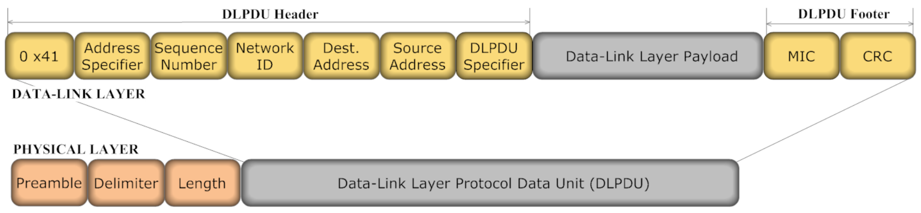


Figure 2.5: WirelessHart Data-Link PDU Structure

The TDMA Data Link Layer Specification[18] defines five *WirelessHART* frame types.

[1]In the reminder of this thesis the *WirelessHART* frame and the DLPDU are used interchangeably

These frame types are:

1. Acknowledgment DLPDU
2. Advertise DLPDU
3. Keep-Alive DLPDU
4. Disconnect DLPDU
5. Data DLPDU

All the DLPDUs except the Data DLPDU are exclusively the DLL PDUs. Essentially, Acknowledgment (ACK), Advertise, Keep-Alive and Disconnect *WirelessHART* DLPDUs are originated at the DLL and are exchanged only between neighboring peer devices.

Advertisement DLPDUs are invitations to a network. Devices wishing to join a network listen for Advertisement DLPDUs.

Keep-Alive DLPDUs are frames without a payload, they can be used for network time synchronization, access communication with a neighbor and a neighbor discovery.

ACK DLPDUs are used to inform a sender of a non-broadcast frame if the DLPDU is accepted or not. ACK DLPDUs contain a Response Code and a Time Adjustment field (a difference between expected and actual time of the DLPDU reception). Destination Devices will respond with an ACK DLPDU in response to all Keep-Alive, Advertise, or Disconnect DLPDUs addressed to a device and successfully receive[19].

Disconnect DLPDUs are used to advise neighboring devices that the device is leaving the network.

2.3.3 The WirelessHART Network Layer

The Network Layer defines and controls the operation of a network and it is a cornerstone for the *WirelessHART* protocol. Figure 2.6 shows a Network Protocol Data Unit (NPDU) structure. The Network Layer responsibilities consist of several functions including packet routing, ensuring secure end-to-end communications and encapsulating the Transport Layer information exchanged across a network.

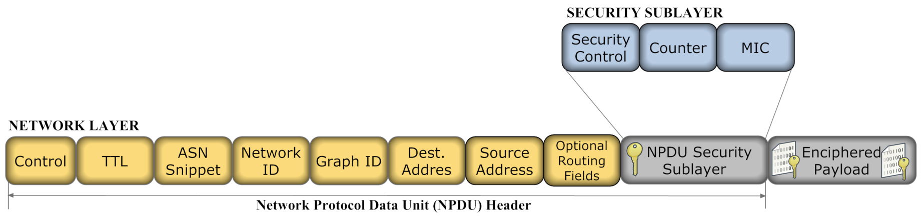


Figure 2.6: WirelessHART NPDU Structure

The NPDU header starts from a Control byte that specifies an addressing scheme employed and indicates if special routes are used in the reminder of the header.

A Time-To-Live (TTL) field is a counter which is decremented at the each next hop, hence determining an amount of hops a packet can travel before it is dropped.

An Absolute Slot Number (ASN) Snippet field provides performance metrics and a diagnostic information of a network operation. This field specifies the time passed since a packet was created.

A Graph ID field is used to route a packet across a network, identifying nodes which can be used along the way.

Remaining fields specifies addresses and additional routing options.

Security sublayer is a part of the NPDU header, it is used for data encryption (encipherment) and the NPDU authentication. Security sublayer Control field is 8 bit in length and it specifies a type of a security employed. The length of the security sublayer (excluding data) depends on the type of a security used. Currently there are three security types:

1. Session Keyed Security: Total length = 6 Bytes
2. Join Keyed Security: Total length = 9 Bytes
3. Handheld Keyed Security: Total length = 9 Bytes

The other two fields in the security sublayer are needed by a security algorithm. The NPDU payload is encrypted using the algorithm specified in the Security Type field.

A Message Integrity Code (MIC) is responsible for checking data integrity. An overall length of the NPDU header may vary depending on the length of the source and the destination addresses, special routes and the counter length. The minimum length of the NPDU header is 21 bytes.

In summary the total NPDU consists of the NPDU header (including security bits) and an enciphered payload. The enciphered payload consists of the Transport Layer PDU and is added to deliver an actual data such as *WirelessHART* commands. After the NPDU is assembled it is passed to the DLL.

2.3.4 The WirelessHART Transport Layer

WirelessHART employs the lightweight Transport Layer which is used to indicate a status of a device and ensure an end-to-end packet delivery. Figure 2.7 illustrates the *WirelessHART* Transport Layer PDU (TPDU) structure.

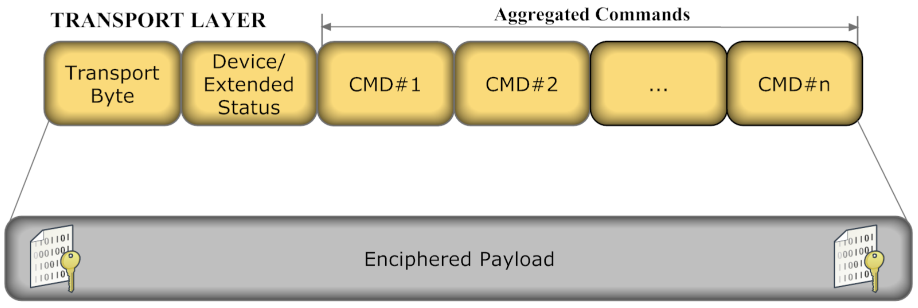


Figure 2.7: Wireless HART TPDU Structure

The TPDU is enciphered using one of the session keys or the join key. Devices that wish to communicate must be provisioned with the identical join keys. The Transport Layer encapsulates *WirelessHART* Application Layer data, that is an array of aggregated commands.

2.3.5 The WirelessHART Application Layer

The Application Layer PDU (APDU) contains an actual command data used in *WirelessHART* communications. Figure 2.8 illustrates the structure of the Application Layer.

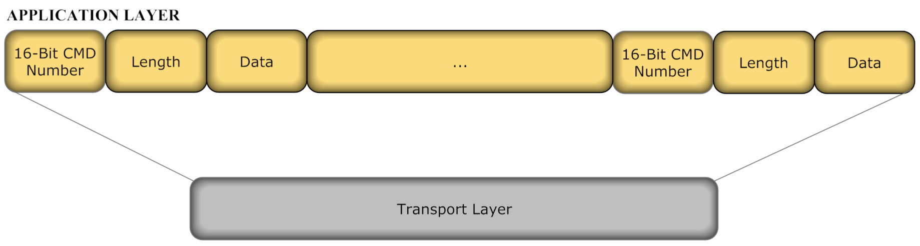


Figure 2.8: Wireless HART Application Layer PDU Structure

Each command starts with a 16-bit number allowing for total of 65535 commands[20]. The length field indicates the size of a command and the Data field contains the command payload.

2.4 WirelessHART Security Analysis

The *WirelessHART* security is reasonably strong[21] however using a symmetric-key encryption standard (i.e. AES) can be potentially dangerous due to a fact that a private key has to be distributed prior communications. At one point a network operator or the Network Manager has to provide an initial key to a remote device. *WirelessHART*-enabled devices use the join key for integration into a network. Usually remote devices are configured with the join key via their maintenance port by an operator. It is also possible to distribute the join key with the Network Manager assistance. Upon powering on a *WirelessHART*-enabled device it does not have any keys except the well-known key that is used as a per-hop key for unauthenticated device before the Network Manager allocates the Network Key. The Well-Known key is used in advertisements and when joining a network.

After performing initial device provisioning which include obtaining the join key, a join request is initiated. The join request is used to query the Network Manager for an admission to a network and to establish a secure channel between a Field Device and the Network Manager. Figure 2.9 illustrates the security keys that are used at different steps of a device integration into a network.

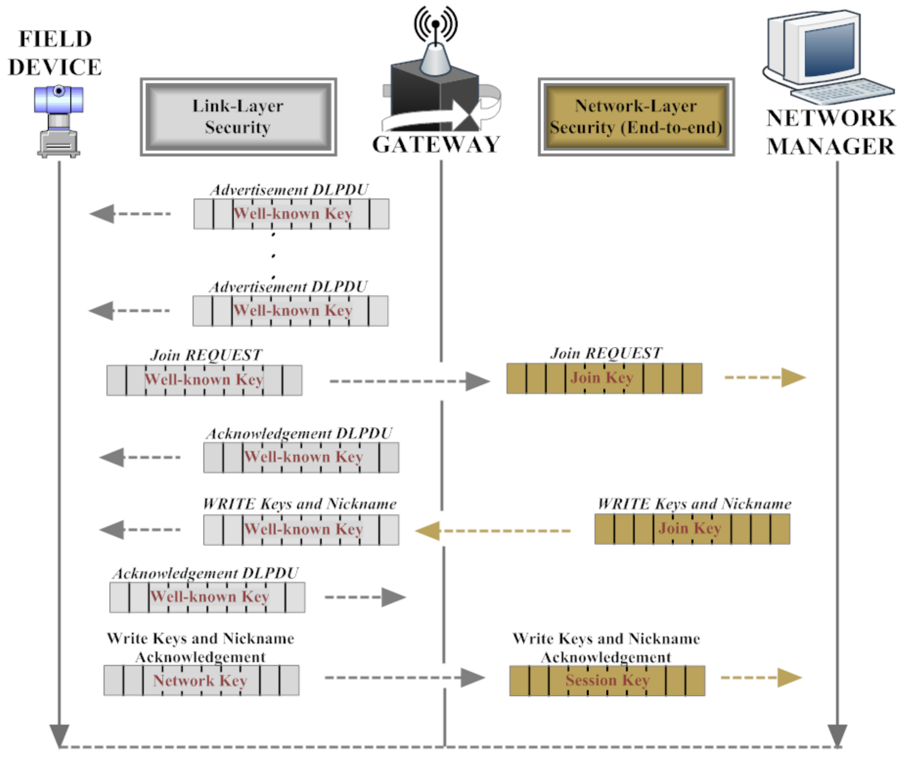


Figure 2.9: WirelessHART Device Integration into a Network and Security Keys Allocation

First, the configured join key is used to encipher the join request between a remote device and the Network Manager. At the Data Link Layer the Well-Known key is used to calculate and confirm the MIC. The Well-Known key is generally known and can be found in the TDMA *WirelessHART* specification [18]. The join key is written via a device maintenance port and it can be extracted by a network operator or by a person who has access to a Network Manager console or a web interface. Obtaining the join key allows to decrypt captured packets that are encrypted using the join key.

In order to be admitted to the network device has to present its credentials which are:

1. The device's Identity
2. The Device's Long Tag
3. The list of Neighbors detected by the device
4. The devices's Join Key

If the Network Manager will decide that a device can be granted an access to a network it will allocate the session and the network keys and the device's nickname (2-byte short address). An encipherment of a data packet containing the session and the network keys alongside with the device nickname is done using the join key.

In the figure 2.9 when the Network Manager sends the NPDU containing the network and the session keys, the NPDU is encrypted with the join key that is known to an administrator or an operator. Hence, during the device integration to a network the session and the network keys can be acquired and all subsequent communications can be tracked. In order to successfully decrypt a *WirelessHART* NPDU payload, one should also keep track of the join/session nonces.

In summary, without considerable knowledge about a network (i.e. knowledge about the join/session key) it is extremely hard to eavesdrop communications. Nevertheless, it is possible to sniff a *WirelessHART* traffic using the knowledge about the protocol operation.

Chapter 3

Design and Implementation

A significant part of this thesis is devoted to implementation of the *WirelessHART* simulation framework in Contiki operating system and COOJA network simulator. First, the simulation framework is shown and then applications that are used for this implementation are introduced. Further, some specific challenges are described such as time constrained *WirelessHART* communications, security and software and hardware ACKs.

There are few WSNs simulators available however none of them is particularly suitable for simulating *WirelessHART* networks. COOJA network simulator is a simulation tool that is easy to adapt to the requirements of the *WirelessHART* protocol such as slotted TDMA communications and security. The major problem for simulating *WirelessHART* networks is that there are no complete and stable open-source implementations of the *WirelessHART*-stack available.

In [22] and [23] prototype implementations of *WirelessHART* architecture are discussed. Despite some promising results these architectures are far from being complete and lack many of essential features of the *WirelessHART* protocol. Clearly *WirelessHART* imposed requirements are hard to meet, hence without a dedicated hardware platform it is complicated to fully implement the *WirelessHART* stack.

Our implementation introduces an approach where the critical tasks of the *WirelessHART* protocol can be implemented outside the resource constrained hardware. The main design goals are as follows:

Limited hardware dependency
Since hardware computational power is likely to be a bottleneck, the aim is to reduce the influence of a hardware platform as much as possible.

Implementation generalization
The implementation should not be specific to any protocol or architecture. An operating frequency and a maximum packet length are the only requirements.

3.1 Simulation framework

WSN simulations can be performed in several different approaches. Software simulation is an approach, where a WSN is entirely realized in a software and wireless sensor nodes are either emulated images or software applications reflecting characteristics of real nodes. Simulation in COOJA network simulator is a typical example of such approach, where emulated nodes run the same code as real nodes. A second approach is to deploy a combined network of both real hardware sensor nodes and simulated sensor nodes. This simulation framework is often called a hybrid simulation or an augmented reality. An example of such framework is discussed in [24].

The simulation framework described in this thesis is based on a hybrid simulation approach. A motivation behind a hybrid simulation tool is an absence of open *WirelessHART* stack implementations. Instead, real *WirelessHART*-enabled devices provided by DUST[25] are used. This setup introduces a possibility to test and analyze the operation of the *WirelessHART* protocol by examining an operation of *WirelessHART*-enabled devices. In order to keep the implementation as independent as possible from the used operating system, only a small subset of capabilities of Contiki operating system has been used.

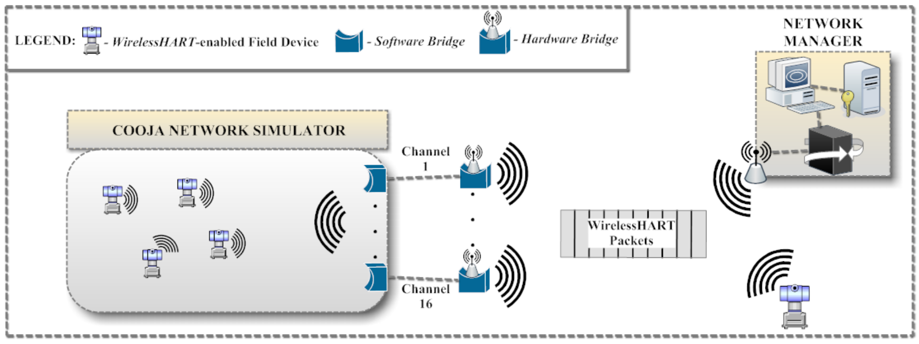


Figure 3.1: The Hybrid Simulation Framework for *WirelessHART*

Figure 3.1 illustrates a simulation environment setup. The framework consists of several components including software and hardware bridges discussed in details in Section 3.2, the *WirelessHART* D2510 Network Manager, Field Devices and COOJA network simulator.

3.1.1 Hardware bridges

Each hardware bridge senses one of the available frequency channels and relays data between COOJA network simulator and the wireless medium. The *WirelessHART* Physical Layer specification[16] defines 16 frequency channels that can be used to transmit or receive data. Initially all of the channels are enabled, however unless significant interference is present, there is no need to use all of the available 16 frequency channels. In this thesis we use 5 frequency channels since it is the minimum amount of channels that has to be enabled. The implementation of the application that runs on a hardware bridge is done in Contiki operating system.

3.1.2 Software bridges

Software bridges are interconnected with their corresponding hardware versions, thus composing a generic Bridge. Software bridges are implemented in COOJA network simulator. Software bridges have similar capabilities as their hardware copies and can be extended with additional features.

3.1.3 Network Manager and Field Devices

The D2510 Network Manager is discussed in Section 2.2.2. The Network Manager is the source of all *WirelessHART* DLPDUs that are sent over the wireless medium. Field Devices and the Network Manager are used for evaluation of a *WirelessHART* network. Information about a *WirelessHART* Field Device can be found in section 2.2.3.

3.1.4 COOJA network simulator

COOJA network simulator contains software applications that reflect an operation of hardware bridges. It also may contain emulated versions of hardware bridges and emulated *WirelessHART*-enabled devices. Hence, the size of a *WirelessHART* network can be extended without a need of additional hardware devices.

There is, however, a little support for *WirelessHART*-enabled devices in COOJA, specifically mainly the *WirelessHART* Network and Transport layers are supported. In this thesis we are mainly focused on lower protocol layers such as the Data-Link Layer and the Physical Layer and on developing interfaces and testing the Network Manager and Field Devices.

3.2 Bridge implementation

The implementation of a generic Bridge contains three components:

- Hardware bridge
- Serial Forwarder Plugin for COOJA
- Software bridge

The main task of a generic Bridge is to forward received data from the 2.4 GHz wireless medium to the simulation environment and vice versa. Data is forwarded over a serial line which is typically a synchronous bus. Before forwarding data over a serial line it is converted into a hexadecimal format. Similarly when a hexadecimal data is received from a serial line it is converted into 8-bit Unicode characters. The figure 3.2 shows the generic algorithm used in a generic Bridge application.

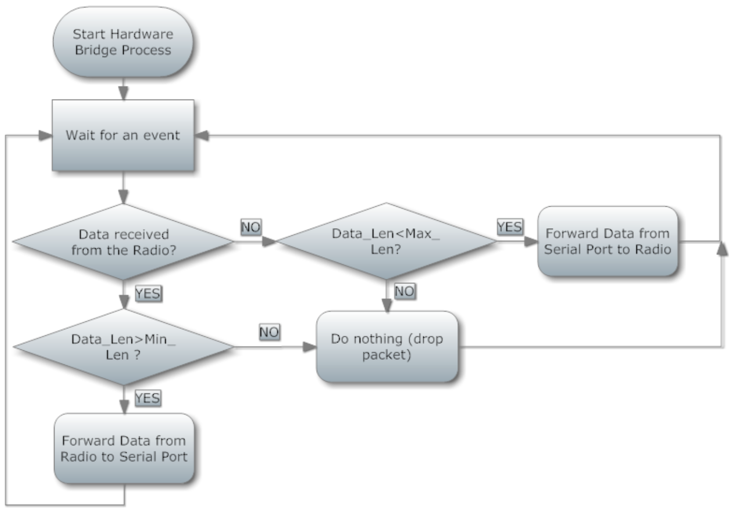
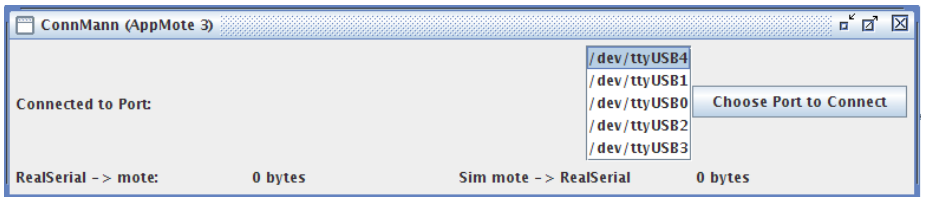

Start Hardware Bridge Process
Wait for an event
Data received from the Radio?
NO
Data_Len<Max_Len?
YES
Forward Data from Serial Port to Radio
YES
NO
Data_Len>Min_Len ?
NO
Do nothing (drop packet)
YES
Forward Data from Radio to Serial Port

Figure 3.2: Generic Bridge Algorithm

Due to the event-based nature of Contiki operating system, after the initialization Bridge process waits for an event to occur. There are two possible events that may happen. Data is either received from the 2.4 GHz wireless medium or from a serial line.

In the first case the length of an incoming data is verified and if it is bigger than the minimum length (in this case 1 byte) data is forwarded over a serial line, otherwise it is dropped. In the second case, received data length is compared with the maximum allowed length (in this case 256 bytes) and if it meets the requirement data is sent via the 2.4 GHz wireless medium, otherwise it is dropped. This simple algorithm allows for a lossless relay of data and is not specific to any hardware platform or operating system.

A Serial forwarder plugin is responsible for sending and receiving data over the USB port. It is implemented using an open-source RXTX library for serial communications[26]. The Serial forwarder in the figure 3.3 is a graphical plugin that is used to connect and listen to the specified serial port and to relay data.


ConnMann (AppMote 3)

Connected to Port:

/dev/ttyUSB4
/dev/ttyUSB1
/dev/ttyUSB0
/dev/ttyUSB2
/dev/ttyUSB3

RealSerial -> mote: 0 bytes Sim mote -> RealSerial 0 bytes

Figure 3.3: Serial Forwarder Plugin for COOJA

Finally, a software bridge has the same capabilities as its hardware image. Since it resides on a powerful computer a software bridge can perform very intense computations, for instance, it can be used to store data and print logs. The idea behind a generic Bridge is to be entirely transparent to higher layers of any communication protocol. Its operation should be similar to the Ethernet bridge in a learning state. The *WirelessHART* protocol, however, imposes some very strict timing requirements at the DLL, and therefore specifically for this protocol implementation an extended version of a generic Bridge (discussed in Section 3.4) is introduced.

3.3 WirelessHART packet analyzer

The *WirelessHART* protocol is loosely organized around the OSI-7 layered architecture. The protocol defines 5 separate layers - the Physical Layer, the Data-link Layer, the Network Layer, the Transport Layer and the Application Layer. Figure 3.4 shows the *WirelessHART* layered architecture.

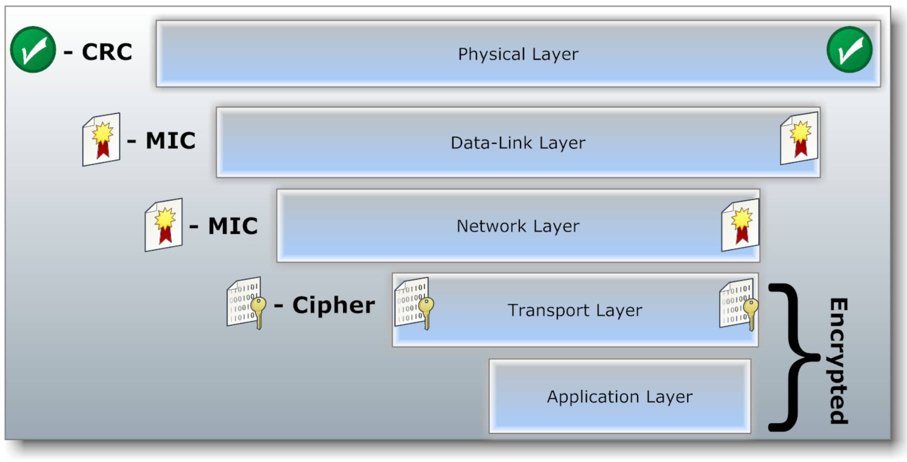


Figure 3.4: WirelessHART Layered Architecture

Each layer of the *WirelessHART* protocol can be described with its corresponding PDU. During this thesis a *WirelessHART* packet analyzer has been developed aiding in understanding of an operation of the protocol and *WirelessHART*-enabled devices. The packet analyzer is designed in a modular fashion, that is each layer of the protocol is processed individually and in order.

PPDU

The PPDU encapsulates the *WirelessHART* DLPDU and its contents are processed and stripped away by the underlying hardware. In the figure 3.4 a PPDU contains the Cyclic-Redundary-Check (CRC), however strictly speaking this a DLPDU field. Implementations of most of the CRC-checks are performed in hardware, thus if a packet arrives without errors CRC is stripped away and a packet is passed to the DLL for further processing.

DLPDU

The DLPDU is an actual data payload that is transferred from a radio transceiver to the DLL. The *WirelessHART* DLPDU is not encrypted, however its contents starting from the first byte (refer to figure 2.5) till the end of the payload, are authenticated using a Counter with CBC-MAC (CCM)[27] security algorithm. If the *WirelessHART* DLPDU contains the Network Layer payload it is passed to the Network Layer for processing. For some *WirelessHART*

DLPDUs such as keep-alive DLPDUs there is no Network Layer overhead, hence it is processed depending on the DLPDU type. The DLPDU's first byte can be used to filter only *WirelessHART* frames.

NPDU

Data DLPDUs contain the Network Layer information and are processed after the DLPDU header and footer. The NPDU header is not enchiphered, however an NPDU payload is enchiphered using CCM security algorithm and then the entire NPDU is authenticated using CCM authentication. First step in processing the NPDU is to process the NPDU header to determine the size of the payload and the security parameters used for data encryption and authentication. After security information is extracted, the corresponding nonces are constructed and the appropriate key is selected. Keys and nonces are stored in hash tables and are selected depending on the security session with the neighboring device.

TLPDU

After decryption of a packet is completed, the plain payload is passed to the Transport Layer for further processing. The Transport Layer information is used to get the status of a device and information about the session (i.e. broadcast or unicast). The TLPDU may contain several commands merged into an array. Each of the commands related to a packet is stored in the array with its corresponding 16-bit command number.

APDU

The Application Layer contains a command data and all of the commands that are stored in the temporary command arrays are processed one after another. There are more than 65000 possible commands in the *WirelessHART* protocol. Some of the commands are generic and are common for every device, however some of the commands are specific to the manufacturer. Clearly it is not possible to know all the specific commands of a proprietary device, therefore the packet analyzer is designed to make it easy to add new commands. Whenever a new command is to be added, a new module is introduced into the code.

After processing of command data is finished, all the buffers are released and the analyzer enters its initial state, that is waiting for an incoming data from a serial port.

The Algorithm

The implementation of the packet analyzer is made as a plugin to the existing radio logger in COOJA network simulator. The basic algorithm of the *WirelessHART* packet analyzer is shown in the figure 3.5. The packet analyzer is designed in a sequential fashion and it is easy to complement with new capabilities, for instance, it can be used to store data in a text file.

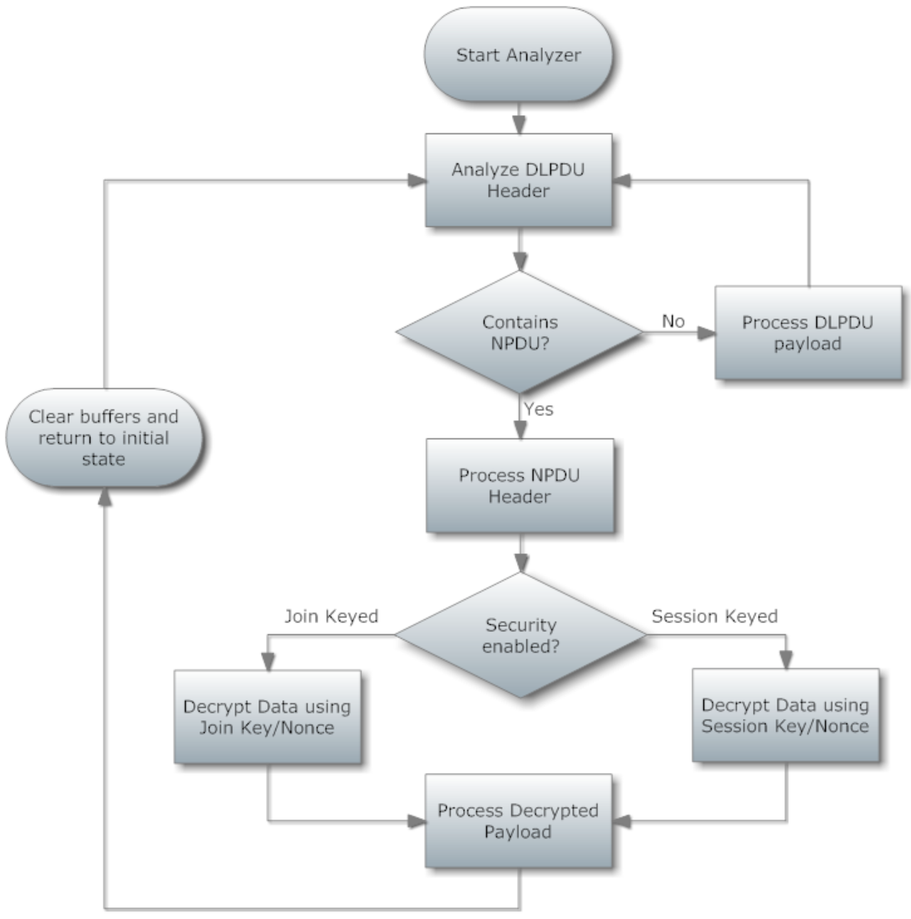

Start Analyzer
Analyze DLPDU Header
Contains NPDU?
No
Process DLPDU payload
Yes
Clear buffers and return to initial state
Process NPDU Header
Join Keyed
Security enabled?
Session Keyed
Decrypt Data using Join Key/Nonce
Decrypt Data using Session Key/Nonce
Process Decrypted Payload

Figure 3.5: WirelessHART Packet Analyzer Basic Algorithm

Since each of the layers of the *WirelessHART* stack is processed individually the packet analyzer can be easily extended without changing the former implementation. A successful processing of a packet requires only the join key. There are commercial *WirelessHART* packet analyzers however none of them allows to adapt its behavior in details like it is done here.

3.4 Extended Bridge

The Tmote Sky wireless sensor node is used as a hardware bridge in this thesis. Although, the Tmote Sky is limited in capabilities especially in its computational power and memory, we show that it possible to meet most of the *WirelessHART*-related requirements using this sensor board.

3.4.1 Time Synchronization with the WirelessHART clock

A generic Bridge operates mainly at the Physical and the Data-Link layers. *WirelessHART* defines a strict $10ms$ time slot and utilizes the TDMA technology to provide collision free and deterministic communications. One particularly critical requirement for a *WirelessHART*-enabled device is the time synchronization with the absolute clock. The *WirelessHART* Gateway is the source of the absolute time in a *WirelessHART* network, hence devices that wish to join a network must synchronize its time with the Gateway's clock. The *WirelessHART* Gateway periodically broadcasts Advertisement DLPDUs that are used, among other things, to maintain the time synchronization. The specific timing requirements inside a *WirelessHART* time slot is shown in the figure 3.6[18].

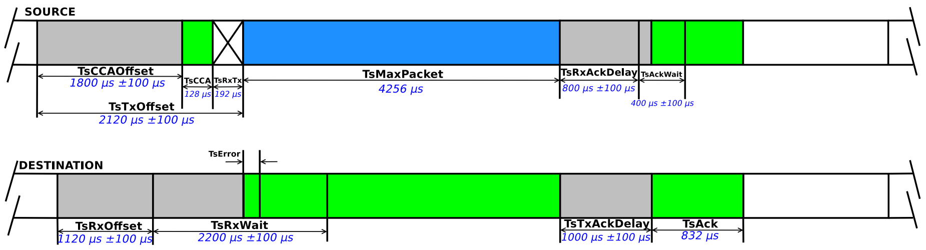


Figure 3.6: WirelessHART Slot Timing

The *WirelessHART* protocol requires the clock resolution of $1\mu s$ to meet the requirements within a *WirelessHART* time slot. The Tmote Sky is equipped with the 32768 Hz watch crystal, thus the maximum achievable clock resolution is around $30.5\mu s$.

In the beginning when a node enters a *WirelessHART* network it has no idea what current time is. To synchronize with the absolute clock a node reads an advertisement DLPDU and extracts the ASN value. A real time timer task is then used to initialize a slot counter.

Eventually, due to a limited resolution of the Tmote Sky clock, time will drift and the synchronization will be lost. To constantly maintain the synchronization with the ASN a node listens to the wireless medium and timestamps frame arrivals and then it calculates the difference between the estimated beginning of a time slot (measured with its own clocks) and its actual beginning.

Since the structure of a time slot is fixed, a node can calculate the beginning of a next time slot using formula:

$$T_{nextSlot} = arrival\_time + 10ms - TsTxOffset$$

Synchronization happens for every frame received and advertisement DLPDUs are used to verify that the ASN and the local slot counter are the same. In Section 4 we discuss stability of the time synchronization module.

3.4.2 SPI Access

A 4-wire Serial Peripheral Interface (SPI) Bus is used for the CC2420 RF transceiver configuration and data buffering. SPI is used for reading an incoming packet from the wireless medium, also for writing a packet to be transmitted and for CC2420 configuration, for instance, to select a security mode.

SPI speed is a limiting factor for most of the operations that use CC2420 RF transceiver. Early results showed that without SPI acceleration some critical tasks, such as reading bytes from a receiver buffer, take too much time. The decision has been made to increase the SPI speed in Contiki operating system for specific tasks. The SPI speed is increased for reading and writing data, and for configuring CC2420 transceiver.

3.4.3 WirelessHART Link-layer Acknowledgments

WirelessHART-enabled devices make use of DLL ACKs to provide reliable hop-to-hop communications. The *WirelessHART* frame transfer and its corresponding ACK occur within the same time slot (refer to the figure 3.6). A node has $1ms$ to prepare an ACK (TsTxAckDelay field) that includes switching radio transceiver from receive to transmit mode and constructing an ACK. The latency in a serial line between COOJA and an extended Bridge is too high to create DLL ACKs in software.

One possible solution to this problem is to use a hardware bridge for DLL ACKs. Essentially, time to prepare an ACK is the main bottleneck within a time slot. In order to prepare an ACK a node should read contents of a DLPDU header, construct an ACK DLPDU and authenticate it using the CCM security mode. In Section 4 we show the total timing for preparing an ACK DLPDU using the Tmote Sky and CC2420 radio transceiver.

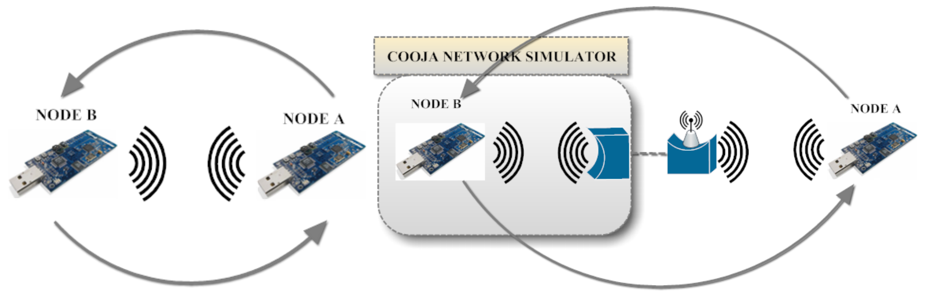
Chapter 4

Evaluation

The latency for relaying data introduced by a generic Bridge and the timing and stability of the *WirelessHART* Data-Link Layer for the Tmote Sky sensor node are discussed in this chapter.

4.1 Single-hop Round Trip Time Measurements

A Communication delay introduced by a serial line and hardware and software bridges is the most critical metric for a generic Bridge. A generic Bridge is represented by two nodes and packets that are relayed via a generic Bridge are expected to exhibit a higher delay.



(a) Direct Communications (b) Bridged Communications

Figure 4.1: Single-hop Latency Measurements

The figure 4.1 shows two experiment setups while measuring the single-hop latency introduced by a generic Bridge. In the figure 4.1b the Round Trip Time (RTT) is measured using a Generic Bridge as an intermediary device between COOJA network simulator and a real node. The Node A sends a packet that is relayed via a generic Bridge to the Node B which immediately responds with the same packet. Figure 4.1a shows the direct communications experiment setup performed exclusively with real nodes.

In order to get accurate RTT measurements the time when a packet is detected by the RF transceiver is saved and compared with the next time when a packet is detected. In addition to the transmission time of a packet payload, the IEEE 802.15.4 Physical Layer overhead has to be taken into account.

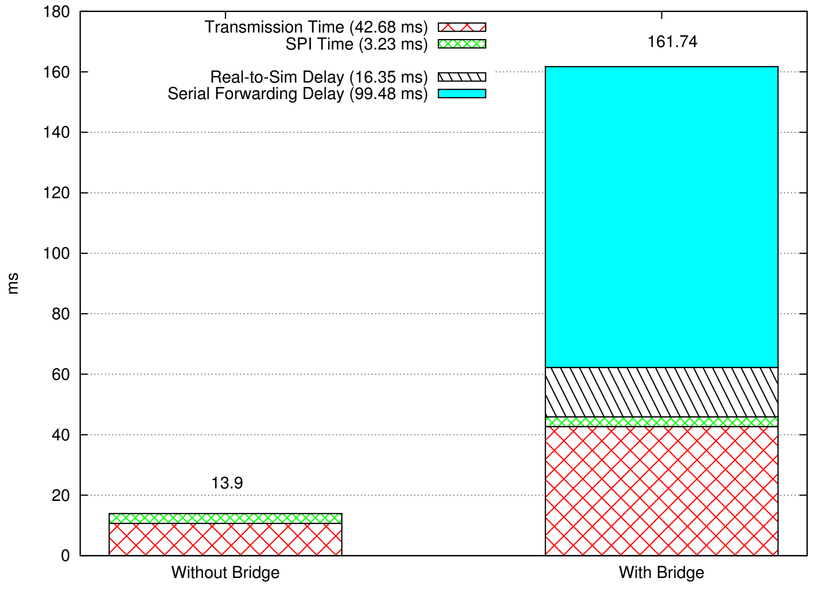


Figure 4.2: Round Trip Time Increase with a Generic Bridge

Measurements were performed over 100 packets with a packet size of 125 bytes. An RTT comparison between the two experiment setups (refer to the figure 4.1) is shown in the figure 4.2.

The IEEE 802.15.4 maximum transmission rate is 250 kbps, thus the transmission time for a 125 bytes packet is $4ms$. The transmission time is increased by $192\mu s$ due to the IEEE 802.15.4 Physical Layer overhead (preamble and SFD) and the length byte that is not a frame payload. The Physical Layer overhead is taken into consideration only for a response packet, whereas the length byte should be calculated for both packets. Hence, the total transmission time for the 125 bytes response packet is $4192\mu s$ and for the request packet is $4032\mu s$, and thus the RTT is $8224\mu s$. In [28] was shown that the actual single-hop maximum transmission rate is, in fact, lower than the theoretical maximum and is equal to 225 kbps and therefore the RTT for single-hop communications in figure 4.1a is approximately $9.14ms$.

	Measured Time	Expected Time
RTT (ms)	10.67	9.14

Table 4.1: RTT transmissions

Measurement results and expected values for the direct communications scenario are summarized in the table 4.1. The difference between the values is introduced due to, for instance, a serialization delay.

Bridged communication scenario (refer to the figure 4.1b) in the figure 4.2 exhibits 1163% higher RTT of which $16.35ms$ is a delay introduced by the Tmote Sky, specifically delay due to a data forwarding to a serial line. Also, since in this setup there are 4 nodes the transmission time is 4 times higher. The other factor that influences the communication delay is a Serial Forwarder application delay.

Although it is possible to optimize the performance of a generic Bridge, no acceleration technique will allow to achieve the communication delay similar to the delay of a direct communications scenario. The *WirelessHART* request/response transaction occurs in a $10ms$ time span, obviously with the currently achievable speed it is not possible to satisfy these requirements. Fortunately, *WirelessHART* does not impose such strict demands on end-to-end communications which occur based on a network schedule and per slot basis. The use of an extended Bridge for a request/response transaction within $10ms$ can help to overcome this problem.

4.2 Multi-hop Round Trip Time Measurements

The figure 4.3 illustrates an experiment setup used to measure the multi-hop communication delay. Four real Tmote Sky sensor nodes are connected with four emulated nodes via a Generic Bridge. All of the nodes run the same application constructing a typical mesh network. We start by measuring the multi-hop round trip time. We also analyze the difference in the RTT by substituting the real nodes with the emulated images in the simulation framework. As we do not have the *WirelessHART* stack, we do not use *WirelessHART*-enabled devices, however the overall performance is expected to be similar.

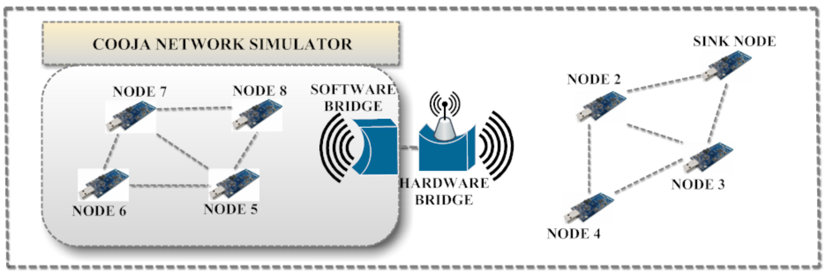


Figure 4.3: Multi-hop Latency Measurements

The table 4.2 summarizes measured statistics for several experimental setups.

Number of Real Nodes	4	3	2	1
Number of Simulated Nodes	4	5	6	7
RTT (ms)	319	324	322	326

Table 4.2: Multi-hop Round Trip Times

In this experiment the packet is forwarded via all of the nodes until it comes back to the sink node - node which is the source of the initial packet. The RTT is measured at the sink node which is a real hardware sensor node.

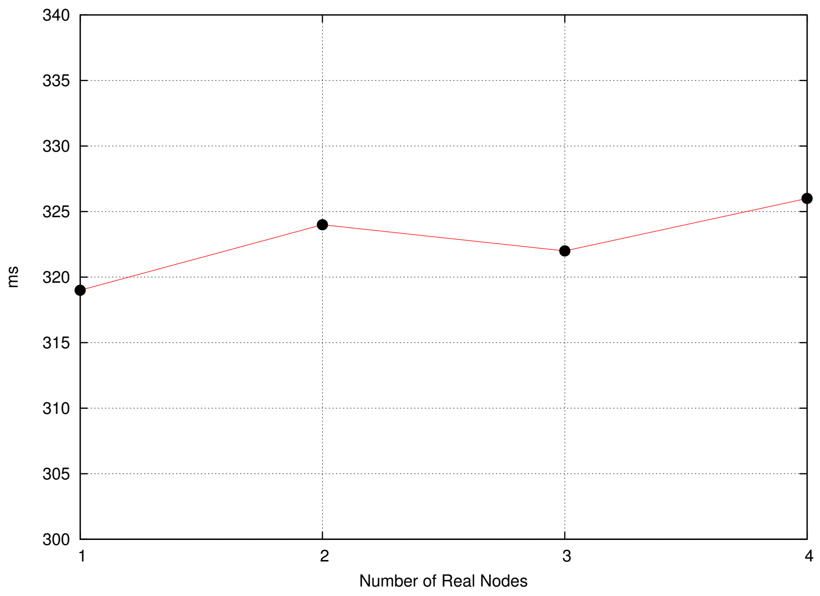


Figure 4.4: RTT Deviation in A Mesh Network

The multi-hop RTT is shown in the figure 4.4. Since emulated nodes precisely reflect their hardware copies, the RTT remains approximately the same regardless of the amount of real nodes.

Our hybrid simulation framework introduces a certain variation in the RTT and this is an another reason to use an extended bridge for time critical operations of the *WirelessHART* protocol.

4.3 A WirelessHART Network Traffic Analysis

The *WirelessHART* protocol analyzer is the primary tool for evaluating a *WirelessHART* network traffic. Physical interfaces for capturing *WirelessHART* packets are represented by the RF transceivers of the Tmote Sky sensor nodes each tuned to a specific frequency, based on the Network Manager configuration. Instead of using channel hopping and thus decreasing the reliability of the analyzer packet capture rate, we use a number of radios each listening at the specific channel. This approach do not require to adjust the operation of the analyzer to the frequency hopping scheme and is more reliable since it is guaranteed that every packet will be captured.

4.4 WirelessHART ACK Timing

Upon successful reception of a frame (last byte of a frame has arrived to a destination device) a node must construct (if a frame passes security and error checks) an ACK and respond within the time specified by a TsTxACK Delay (refer to the figure 3.6).

In order to successfully acknowledge a frame at the DLL a node must be able to prepare an ACK within approximately 1000 μs (not including an error tolerance of $100\mu s$). The ACK preparation consists of reading a part of an incoming DLPDU header (up to 22 bytes if both addresses are 8 bytes), writing bytes to a FIFO queue (up to 20 bytes) and performing an in-line authentication of a message in the FIFO queue. Since an ACK is the DLL PDU it does not contain the Network Layer overhead, and therefore an encrypted payload is absent in the ACK DLPDU. It is assumed that any received *WirelessHART* frame passes an authentication check, thus a MIC is not verified.

Measurement *WirelessHART*	Average time $[\mu s]$[2]
Reading from FIFO	289
Writing to FIFO	174.56
Authentication[3]	186.767
ACK total time	663.76

Table 4.3: Hardware ACK timing

The table 4.3 shows measured values to prepare an ACK on the Tmote Sky sensor node and the figure 4.5 provides a visual comparison between these values.

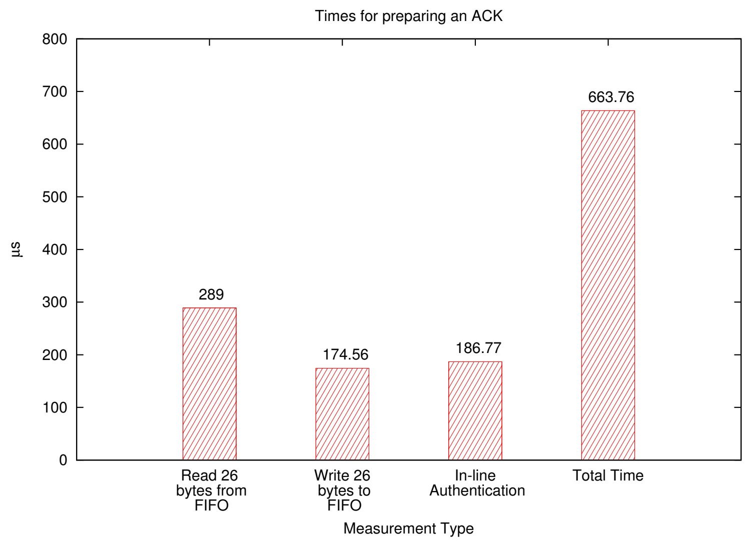


Figure 4.5: ACK Creation Time Expectancy

The TsTxAckDelay (refer to the figure 3.6) include the transmission time of an ACK. The reception of an ACK starts from locating the Start Of the Message (SOM) which begins directly after the synchronization header.

The SHR is always transmitted first for any frame employing the IEEE 802.15.4 Physical Layer. The SHR consists of the preamble and the SFD fields. In addition, every time before the SHR transmission sender waits for the time equal to 12 symbols. After byte to symbol mapping 5 bytes of the SHR result

[2]Measured over 100 samples without accounting for an absolute error. The absolute error for an individual measurement is $\triangle X = X_0$ - X, where $\triangle X \cong 30.5175\ \mu s$.

[3]Not including the transmission time for an authenticated message.

in 10 symbols, hence the total number of symbols before the SOM is detected equals to 22 symbols. The 2450 MHz Physical Layer employs a 62.5 ksymbol/s rate which equals to 16 μs per symbol, and therefore the total (theoretical) duration before the SOM detection is equal to 352 μs.

Measurement	Average Time [μs]	Average Time (boosted SPI) [μs]
SHR+12 symbols	335.7	335.7
TsTxAckDelay[4]	1007	925.3

Table 4.4: Sending node (Network Manager side)

The Table 4.4 illustrates the duration of several states within an individual *WirelessHART* transaction. Due to the Tmote Sky limited timer resolution an error of 1/32768 s (30.5175 μs) can be introduced, however with the current precision it is still possible to achieve sensible results.

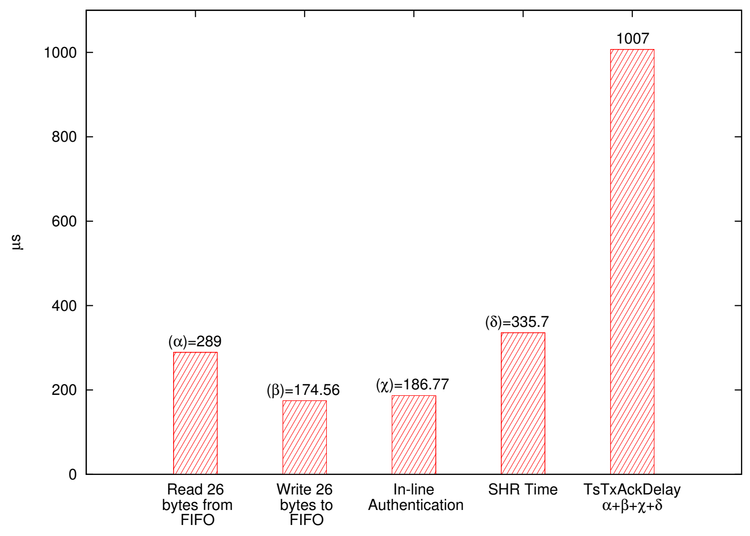


Figure 4.6: ACK Total Time without SPI Acceleration

In figures 4.6 and 4.7 the total time when reading 26 bytes from the RX FIFO for delivering an ACK is shown. The figure 4.6 shows the time without the acceleration of an SPI and figure 4.7 shows the time with the acceleration of an SPI.

In the worst case scenario the total time for preparing and sending an ACK using the Tmote Sky node is 1007 μs. In fact the Network Manager is likely to accept an ACK arriving after 1007 μs since it has a certain error variation tolerance. Nevertheless, although 1007 μs is almost equal to the allowed maximum value, it is unnecessary to read 26 bytes to prepare an ACK. In most situations it will be enough to read 10 bytes from an RX FIFO, hence the total time will be reduced by approximately 150-180 μs resulting in around 800 μs. Finally, most of the bytes of an ACK frame can be preloaded into a TX FIFO in advance (for instance during reception of a frame) since they

[4]Including Synchronization Header duration and 12 symbols waiting time.

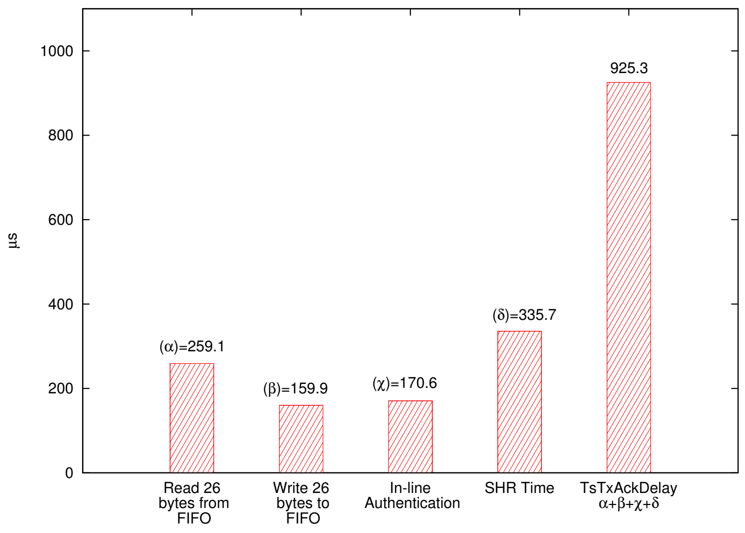


Figure 4.7: ACK Total Time with SPI Acceleration

are generally known (i.e. Network ID, DLPDU Specifier, First Byte, Source Address and even Destination Address) which basically will decrease a FIFO writing time to 30-50 μs.

4.5 The Time Synchronization Stabilitiy

The time synchronization with the Gateway ASN requires very stable and precise clocks. The Tmote Sky sensor node resolution upper bound is around $30\mu s$ whereas the *WirelessHART* TDMA specification states a resolution of $1\mu s$. Nevertheless, here we test our clock module implementation with regards to the clock drift and the clock offset. The clock drift indicates the frequency of local clock's change over time and the clock offset is the difference of the local clock from the real time (ASN).

Over the 10 minutes time span (60 000 time slots) conducted experiments show 100% reliability of the clock, that is the clock drift is zero. The clock offset, however, is variable as it is expected due to 16 KHz clocks employed in the Tmote Sky sensor node.

Figure 4.8 shows the local clock deviation from the real time. Clocks are synchronized upon reception of a packet and a shift error is calculated. Time synchronization is the most critical task and therefore the synchronization and error calculation is performed in the interrupt - immediately after the last byte of a packet is received.

The comparison is made between the local clocks and the time between packet receptions. The local clocks do not allow for an absolute precision necessary for the *WirelessHART* protocol, thus essentially a clock offset is introduced.

Obtained results indicate that the absolute clock offset does not exceed $100\mu s$ with the 95% probability, however this synchronization algorithm is stable only

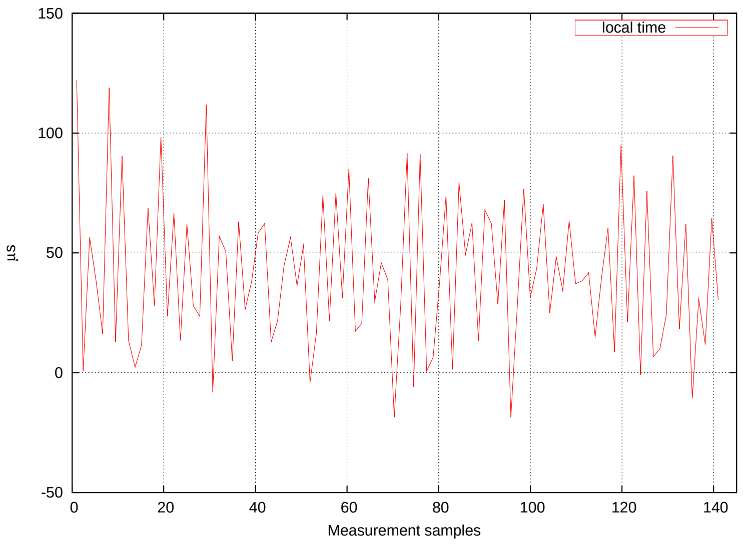


Figure 4.8: Local Clock Deviation from The Real Time

if messages are sent frequently. With the longer time between two consecutive messages at the same channel the clock offset becomes higher. Fortunately, *WirelessHART* Advertisement DLPDUs are sent periodically and on a frequent basis (at least one Advertisement DLPDU per second), therefore it is a valid approach to synchronize the clocks.

Chapter 5

Related Work

A hybrid simulation framework for WSNs has been discussed by Lalomnia et al. [24] based on TOSSIM [29] WSN simulator. In addition, the authors discuss the timing issues using simulated and real nodes and introduce a concept of a shadow node which is essentially a generic Bridge. This report considers similar issues for WSN hybrid simulations targeting cross platform implementation in COOJA with some support of Contiki operating system.

In Wen et al. [30] an example scenario for WSN augmented reality is presented. The authors introduce some relevant issues in WSN such as routing and media access with regards to augmented reality. In this report mainly latency problems are discussed and higher level issues were not in interest.

WirelessHART has been thoroughly studied by Kim et al. [31]. This paper describes, among other things, implementation and design issues when implementing *WirelessHART* architecture.

Many of the related security issues in *WirelessHART* was studied in Raza et al. [21]. The authors introduce pros and cons of *WirelessHART* security scheme alongside with a prototype implementation of a security manager.

Song et al. [22] and Gustafson [23] propose prototype architectures for *WirelessHART*. In the first case the authors implement a fairly complete *WirelessHART* architecture targeting such critical parts such as synchronization, time keeping, routing and scheduling. In the later case, the author mainly focuses on higher level issues such as the Network Layer and the Transport Layer services including reliable data delivery through the mesh network and handling of commands. None of the mentioned papers, however, specifically targets security implementation issues in *WirelessHART* as even specification remains independent from it. In this thesis, we look closely at the security implementation issues and the corresponding timing problems both in real devices and in augmented reality. We have also introduced time synchronization algorithms and related MAC problems for *WirelessHART*.

Time keeping problems in WSN were studied in [32] where a concept of Reference Broadcasts for network time synchronization is proposed and a competing scheme is introduced in [33]. In this thesis we evaluate time synchronization with respect to well established parameters such as a clock drift and clock offset. Main advantage of the current scheme is that no special software protocol is needed to keep network-wide synchronization, since it is aligned with the absolute network clock.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Hybrid simulations present a feasible tool for evaluation of *WirelessHART* networks, and thus critical *WirelessHART* tasks can be decoupled from less crucial routines. Time limited operations at the Physical and Data Link layers are performed synchronously whereas less critical operations are processed asynchronously. The latency when performing hybrid simulations is too high for the *WirelessHART* protocol, however resource constrained hardware platform is suitable for performing most of the time critical *WirelessHART* tasks and the power unconfined machine can be used to perform time unlimited operations. Hardware acknowledgments can be used to maintain communication with the *WirelessHART* Network Manager, thus allowing for stable network operation. The $923.5 \mu s$ needed to prepare a *WirelessHART* ACK is enough to be within the time bounds required by the *WirelessHART* specification (refer to the figure 3.6). Finally, time synchronization module shows no major break aways and the clock offset is small enough to be within the time slot boundaries (refer to the figure 4.8).

6.2 Future Work

This thesis work is one of the first steps towards implementation of *WirelessHART* simulations. Fields such as partial cross platform implementation of *WirelessHART* architecture and simulation scalability issues are outside the scope of this thesis.

Although real time hybrid simulations for WSNs is a desirable approach, scalability problems are likely to be encountered. Another possibility is to perform simulations in a non real time which will be considered for the future work. In order to foster further development of a simulation framework for *WirelessHART*, a stable, cross platform and preferably an open source implementation of the *WirelessHART* Network Manager, the Gateway and a Field Device is necessary. Future efforts will be dedicated towards the implementation of the Network Manager architecture and related *WirelessHART* components.

References

[1] HART Communication Foundation, *The HART Protocol - A Solution Enabling Technology*, February 2004.

[2] "Hart communication foundation." `http://www.hartcomm.org`.

[3] HART Communication Foundation, *HART Communication Protocol Specification*, March 2009.

[4] "The tmote sky / telosb platform." `http://www.sics.se/contiki/platforms/the-telos-sky-platform.html`, March 2007.

[5] A. Dunkels, B. Grőnvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, (Tampa, Florida, USA), November 2004.

[6] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, "Protothreads: Simplifying event-driven programming of memory-constrained embedded systems," in *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)*, (Boulder, Colorado, USA), 2006.

[7] "Moteiv corporation. tmote sky - ultra low power ieee 802.15.4 compliant wireless sensor module," November 2006.

[8] A. Dunkels, "Rime — a lightweight layered communication stack forsensor network," in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, (Delft, The Netherlands), January 2007.

[9] A. Dunkels, "Tcp/ip for 8-bit architectures," in *Proceedings of the Swedish National Computer Networking Workshop*, (Arlanda stad, Sweden,), 2003.

[10] M. Durvy, J. Abeill, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels, "Making sensor networks ipv6 ready," in *Proceedings of the Sixth ACM Conference on Networked Embedded Sensor Systems (ACM SenSys 2008), poster session*, (Raleigh, North Carolina, USA), November 2008.

[11] F. Ősterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*, (Tampa, Florida, USA), November 2006.

[12] J. Polastre, R. Szewczyk, and D. E. Culler., “Telos: enablingultra-low power wireless research,” in *IPSN*, pp. 364–369, IEEE, 2005.

[13] “Msp430 - mixed signal microcontroller.” Texas Instruments, 2004.

[14] “Chipcon: Cc2420 802.15.4 compliant radio.” `http://www.chipcon.com`, 2007.

[15] Dust Networks, *SmartMesh IA-510 D2510 Manager Guide*, 040-0060 rev. 1 ed., April 2008.

[16] HART Communication Foundation, *2.4GHz DSSS O-QPSK Physical Layer Specification*, September 2007.

[17] IEEE Computer Society, *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, September 2006.

[18] HART Communication Foundation, *TDMA Data Link Layer Specification*, May 2008.

[19] HART Communication Foundation, *Network Management Specification*, March 2009.

[20] HART Communication Foundation, *Command Summary Specification*, July 2007.

[21] S. Raza, A. Slabbert, T. Voigt, and K. landernås, “Security considerations for the wirelesshart protocol,” in *Proceedings of the 14th IEEE international conference on Emerging technologies and factory automation*, 2009.

[22] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, M. Nixon, , and W. Pratt, “Wirelesshart: Applying wireless technology in real-time industrial process control,” *Real-Time and Embedded Technology and Applications Symposium, IEEE*, p. 0:377–386, 2008.

[23] D. Gustafsson, “Wirelesshart - implementation and evaluation on wireless sensors,” Master’s thesis, Royal Institute of Technology, 2009.

[24] A. Lalomia, G. L. Re, and M. Ortolani, “A hybrid framework for soft real-time wsn simulation,” *International Symposium on Distributed Simulation and Real Time Applications, IEEE/ACM*, 2009.

[25] “Dust networks.” `http://www.dustnetworks.com`.

[26] “Rxtx - a native library providing serial and parallel communication for the java development toolkit (jdk).” `http://www.rxtx.org`.

[27] M. Dworkin, “Recommendation for block cipher modes of operation: The ccm mode for authentication and confidentiality,” tech. rep., National Institute of Standards and Technology, May 2004.

[28] F. Ősterlind and A. Dunkels, “Approaching the maximum 802.15.4 multi-hop throughput,” tech. rep., Swedish Institute of Computer Science, March 2008.

[29] P. Levis, N. Lee, M. Welsh, and D. Culler, “Tossim: Accurate and scalable simulation of entire tinyos applications,” *SenSys, Los Angeles, California, USA.*, November 2003.

[30] Y. Wen, W. Zhang, R. Wolski, and N. Chohan, “Simulation-based augmented reality for sensor network development,” *SenSys, Sydney, Australia.*, November 2007.

[31] A. N. Kim, F. Hekland, S. Petersen, and P. Doyle, “When hart goes wireless: Understanding and implementing the wirelesshart standard,” in *ETFA*, pp. 899–907, 2008.

[32] J. Elson, L. Girod, and D. Estrin, “Fine-grained network time synchronization using reference broadcasts,” in *Proceedings of the Fifth Symphosium on Operating Systems Design and Implementation (OSDI 2002)*, (Boston, MA), 2002.

[33] S. Ganeriwal, R. Kumar, and M. B. Srivastava, “Timing-sync protocol for sensor networks,” in *Proceedings of ACM Conference on Embedded Networked Sensor Systems SENSYS.*, 2003.