WirelessHART: Applying Wireless Technology in Real-Time Industrial Process Control

Jianping Song, Song Han, Aloysius K. Mok
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712, USA
{sjp, shan, mok}@cs.utexas.edu

Deji Chen, Mike Lucas, Mark Nixon
Emerson Process Management
12301 Research Blvd., Bldg. III
Austin, TX 78759, USA
{deji.chen, mike-1.lucas, mark.nixon}@emerson.com

Wally Pratt
HART Communication Foundation
9390 Research Blvd., Suite I-350
Austin, TX 78759, USA
wallyp@hartcomm.org

Abstract

Wireless technology has been regarded as a paradigm shifter in the process industry. The first open wireless communication standard specifically designed for process measurement and control applications, WirelessHART was officially released in September 2007 (as a part of the HART 7 Specification). WirelessHART is a secure and TDMA-based wireless mesh networking technology operating in the 2.4GHz ISM radio band. In this paper, we give an introduction to the architecture of WirelessHART and share our first-hand experience in building a prototype for this specification. We describe several challenges we had to tackle during the implementation, such as the design of the timer, network wide synchronization, communication security, reliable mesh networking, and the central network manager. For each challenge, we provide a detailed analysis and propose our solution. Based on the prototype implementation, a simple WirelessHART network has been built for the purpose of demonstration. The demonstration network in turn validates our design. To the best of our knowledge, this is the first reported effort to build a WirelessHART protocol stack.

1 Introduction

Wireless process control has been a popular topic recently in the field of industrial control [20, 15, 14]. Compared to traditional wired process control systems, their wireless counterparts have the potential to save costs and make installation easier. Also, wireless technologies open up the potential for new automation applications. Several industrial organizations, such as ISA [7], HART [3], WINA [8] and ZigBee [10], have been actively pushing the application of wireless technologies in industrial automation. As a milestone of such efforts, WirelessHART is ratified by the HART Communication Foundation in September 2007. WirelessHART is the first open wireless communication standard specifically designed for process measurement and control applications [3].

Before WirelessHART is released, there have been a few publicly available standards on office and manufacturing automation, such as ZigBee [10] and Bluetooth [2]. However, these technologies cannot meet the stringent requirements of industrial control. Compared with office applications, industrial applications have stricter timing requirement and higher security concern. For example, many monitoring applications are expected to retrieve updates from sensors every one second. Neither ZigBee nor Bluetooth makes any effort to provide a guarantee on end-to-end wireless communication delay. In addition, industrial environments are harsher for wireless applications in terms of interferences and obstacles than office environment. Some interferences may be persistent. ZigBee, without built-in channel hopping technique, would surely fail in such environments. Bluetooth assumes quasi-static star network, which is not scalable enough to be used in large process control systems.

The new WirelessHART is specifically targeted to solve these problems and provide a complete solution for process control applications. At the very bottom, it adopts IEEE 802.15.4-2006 [5] as the physical layer. On top of that, WirelessHART defines its own time-synchronized MAC layer. Some notable features of WirelessHART MAC include strict $10ms$ time slot, network wide time synchronization, channel hopping, channel blacklisting, and industry-standard AES-128 ciphers and keys. The network layer supports self-organizing and self-healing mesh networking techniques. In this way, messages can be routed around interferences and obstacles. WirelessHART also distinguishes itself from other public standards by maintaining a central network manager. The network manager is responsible for maintaining up-to-date routes and communication schedules for the network, thus guarantee the network performance.

In this paper we discuss how we developed a prototype

WirelessHART protocol stack. Based on the prototype, we build a three-node network for demonstration purposes. The goal of this paper is to introduce the WirelessHART architecture and to share our first-hand experiences on an implementation of the specification. The contributions of this paper are threefold:

- **Introduction of the architecture of WirelessHART.** We will highlight the features that make WirelessHART suitable for wireless process control.
- **Study of some challenging problems in WirelessHART implementation.** For practical concern, we need to implement the feature-rich WirelessHART on controllers with low processing power and limited resources. We identified and analyzed some challenges, such as time management, communication security, and mesh networking.
- **Sharing of some experiences and lessons learned during the implementation.** To the best of our knowledge, this effort is the first reported attempt to implement the newly approved WirelessHART standard. Those who want to build a full-featured WirelessHART stack should find our experiences helpful.

The remainder of this paper is structured as follows. In Section 2 we review some existing public standards in office and manufacturing automation. We describe the layered architecture of WirelessHART in Section 3. Section 4 presents some challenges and our proposed solutions. In Section 5, we validate our design by a demonstration WirelessHART network. We talk about the future work and conclude the paper in Section 6.

2 Background and Related works

Conceptually, WirelessHART networks are one special type of wireless sensor network. Although it bears many similarities with other wireless standards, such as Bluetooth [2], ZigBee [10], and Wi-Fi [4], WirelessHART differentiates itself from them in many other aspects.

Wireless sensor network has received extensive study recently [13, 18, 17, 21, 22, 12]. Different from generic wireless sensor networks which assume that sensors are deployed randomly and abundantly, the deployment of WirelessHART network is deliberate and has only limited redundancy. In a generic sensor network, many sensors may be deployed in the same area and perform the same function. However, in a WirelessHART network, sensors are usually attached to field devices to collect specific environmental data, such as flow speeds, fluid levels, or temperatures. A reading from a sensor is not necessarily replaceable by that from the nearby sensors. More importantly, generic wireless sensor networks are self-configurable and have no strict requirements on timing and communication reliability. To meet the requirements of wireless industrial applications, WirelessHART uses a central network manager to provide routing and communication schedules. Thus WirelessHART is essentially a centralized wireless network.

WirelessHART, Bluetooth and ZigBee share a very obvious feature: they all operate in the unrestricted 2.4GHz ISM radio band, which is available nearly globally. On the other hand, they distinguish from each other in many other aspects. Both WirelessHART and Bluetooth support time slots and channel hopping. However, Bluetooth is targeted at Personal Area Networks (PAN), whose range is usually set to 10 meters. Furthermore, Bluetooth only supports star-type network topology, and one master can only have up to 7 slaves. These limitations make it awkward to apply Bluetooth in large industrial control systems. In contrast, WirelessHART supports mesh networking directly. The topology of a WirelessHART network can be a star, a cluster or a mesh, thus providing much better scalability.

Both WirelessHART and ZigBee are based on the IEEE 802.15.4 physical layer. While ZigBee uses the existing IEEE 802.15.4 MAC, WirelessHART goes one step further to define its own MAC protocol. WirelessHART introduces channel hopping and channel blacklisting into the MAC layer, while ZigBee can only utilize Direct Sequence Spread Spectrum (DSSS) provided by IEEE 802.15.4. Thus, if a noise is persistent, which is not unusual in industrial fields, the performance of a ZigBee network might degrade severely. By changing the communication channel pseudo-randomly, WirelessHART can limit the damage to minimum.

Just like ZigBee, Wi-Fi does not support channel hopping either. In addition, power consumption is not a concern for Wi-Fi. Thus, Wi-Fi is not a good fit for industrial environment as well.

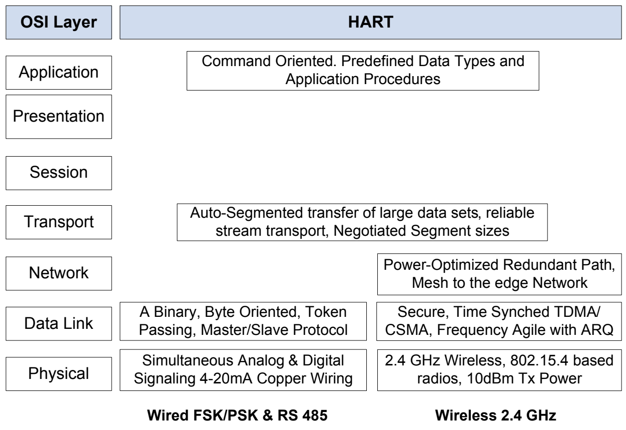
It is noteworthy that ISA SP100 [7] committee is also working on wireless standards for industrial applications. However, the standard is yet to be published.

3 WirelessHART Architecture

In order to make this paper self-contained, we elect to describe in this section the parts of the WirelessHART specification that are related to our work.

Figure 1 illustrates the architecture of the WirelessHART protocol stack according to the OSI 7-layer communication model. As shown in this figure, WirelessHART protocol stack includes five layers: physical layer, data link layer [1], network layer, transport layer and application layer. In addition, a central network manager [19] is introduced to manage the routing and arbitrate the communication schedule.

[1]In the rest of this paper, we use "data link layer" and "MAC layer" interchangeably.



OSI Layer	HART	
Application	Command Oriented. Predefined Data Types and Application Procedures	
Presentation		
Session		
Transport	Auto-Segmented transfer of large data sets, reliable stream transport, Negotiated Segment sizes	
Network		Power-Optimized Redundant Path, Mesh to the edge Network
Data Link	A Binary, Byte Oriented, Token Passing, Master/Slave Protocol	Secure, Time Synched TDMA/ CSMA, Frequency Agile with ARQ
Physical	Simultaneous Analog & Digital Signaling 4-20mA Copper Wiring	2.4 GHz Wireless, 802.15.4 based radios, 10dBm Tx Power
	Wired FSK/PSK & RS 485	Wireless 2.4 GHz

Figure 1. Architecture of HART Communication Protocol

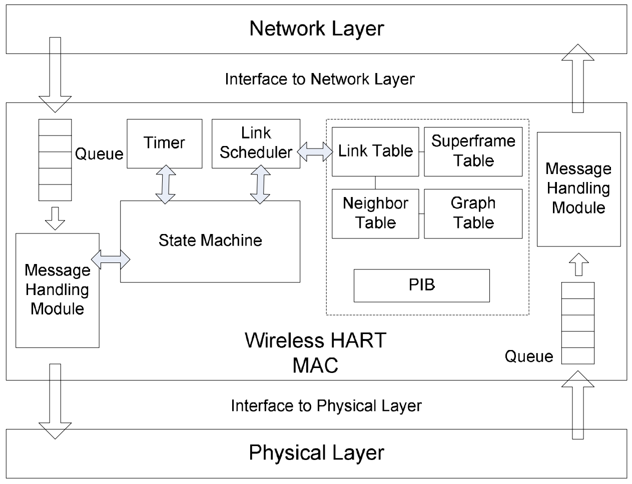


Figure 2. WirelessHART Data Link Layer Architecture

3.1 Physical layer

The WirelessHART physical layer is based mostly on the IEEE STD 802.15.4-2006 2.4GHz DSSS physical layer [5]. This layer defines radio characteristics, such as the signaling method, signal strength, and device sensitivity.

Just as IEEE 802.15.4 [5], WirelessHART operates in the 2400-2483.5MHz license-free ISM band with a data rate of up to 250 kbits/s. Its channels are numbered from 11 to 26, with a 5MHz gap between two adjacent channels.

3.2 Data Link Layer

One distinct feature of WirelessHART is the time-synchronized data link layer. WirelessHART defines a strict 10ms time slot and utilizes TDMA technology to provide collision free and deterministic communications. The concept of *superframe* is introduced to group a sequence of consecutive time slots. Note a superframe is periodical, with the total length of the member slots as the period. All superframes in a WirelessHART network start from the ASN(absolution slot number) 0, the time when the network is first created. Each superframe then repeats itself along the time based on its period.

In WirelessHART, a transaction in a time slot is described by a vector:

{frame_id, index, type, src_addr, dst_addr, channel_offset}
where $frame\_id$ identifies the specific superframe; $index$ is the index of the slot in the superframe; $type$ indicates the type of the slot (transmit/receive/idle); src_add and dst_addr are the addresses of the source device and destination device, respectively; *channel_offset* provides the logical channel to be used in the transaction.

To fine-tune the channel usage, WirelessHART introduces the idea of *channel blacklisting*. Channels affected by consistent interferences could be put in the black list. In this way, the network administrator can disable the use of those channels in the black list totally.

To support channel hopping, each device maintains an active channel table. Due to channel blacklisting, the table may have less than 16 entries. For a given slot and channel offset, the actual channel is determined from the formula:
ActualChannel = (ChannelOffset + ASN) % NumChannels

The actual channel number is used as an index into the active channel table to get the physical channel number. Since the ASN is increasing constantly, the same channel offset may be mapped to different physical channels in different slots. Thus we provide channel diversity and enhance the communication reliability.

Figure 2 describes the overall design of the data dink layer which consists of six major modules as described in the follow subsections.

3.2.1 Interfaces

The interface between the MAC and PHY layer describes the service primitives provided by the physical layer, and the interface between the MAC and NETWORK layer defines the service primitives provided to the network layer.

3.2.2 Timer

Timer is a fundamental module in WirelessHART. It provides accurate timing to ensure the correct operating of the system. One significant challenge we met during the implementation is how to design the timer module and keep those 10ms time slots in synchronization. The specific timing requirement inside a WirelessHART time slot is depicted

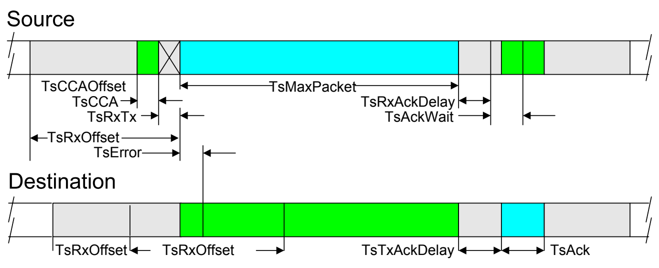


Figure 3. WirelessHART Slot Timing

in Figure 3 and the implementation issues are addressed in Section 4.

3.2.3 Communication Tables

Each network device maintains a collection of tables in the data link layer. The superframe table and link table store communication configurations created by the network manager; the neighbor table is a list of neighbor nodes that the device can reach directly and the graph table is used to collaborate with the network layer and record routing information.

3.2.4 Link Scheduler

The functionality of the link scheduler is to determine the next slot to be serviced based on the communication schedule in the superframe table and link table. The scheduler is complicated by such factors as transaction priorities, the link changes, and the enabling and disabling of superframes. Every event that can affect link scheduling will cause the link schedule to be re-assessed.

3.2.5 Message Handling Module

The message handling module buffers the packets from the network layer and physical layer separately.

3.2.6 State Machine

The state machine in the data link layer consists of three primary components: the TDMA state machine, the XMIT and RECV engines. The TDMA state machine is responsible for executing the transaction in a slot and adjusting the timer clock. The XMIT and RECV engine deal with the hardware directly, which send and receive a packet over the transceiver, respectively.

3.3 Network Layer and Transport Layer

The network layer and transport layer cooperate to provide secure and reliable end-to-end communication for network devices [2].

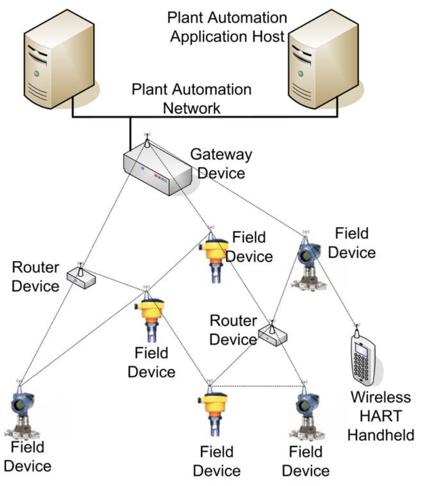


Figure 4. WirelessHART Mesh Networking

As shown in Figure 4, the basic elements of a typical WirelessHART network include: (1) **Field Devices** that are attached to the plant process, (2) **Handheld** which is a portable WirelessHART-enabled computer used to configure devices, run diagnostics, and perform calibrations, (3) A **gateway** that connects host applications with field devices, and (4) A **network manager** that is responsible for configuring the network, scheduling and managing communication between WirelessHART devices.

To support the mesh communication technology, each WirelessHART device is required to be able to forward packets on behalf of other devices. There are two routing protocols defined in WirelessHART:

- **Graph Routing**: A graph is a collection of paths that connect network nodes. The paths in each graph is explicitly created by the network manager and downloaded to each individual network device. To send a packet, the source device writes a specific graph ID (determined by the destination) in the network header. All network devices on the way to the destination must be pre-configured with graph information that specifies the neighbors to which the packets may be forwarded.

- **Source Routing**: Source Routing is a supplement of the graph routing aiming at network diagnostics. To send a packet to its destination, the source device includes in the header an ordered list of devices through which the packet must travel. As the packet is routed, each routing device utilizes the next network device address in the list to determine the next hop until the destination device is reached.

[2]For simplicity, in the rest of the paper, we will not separate these two layers in the presentation.

3.4 Application Layer

The application layer is the topmost layer in WirelessHART. It defines various device commands, responses, data types and status reporting. In WirelessHART, the communication between the devices and gateway is based on commands and responses. The application layer is responsible for parsing the message content, extracting the command number, executing the specified command, and generating responses.

3.5 Security Architecture

WirelessHART is a secure network system. Both the MAC layer and network layer provide security services. The MAC layer provides hop-to-hop data integrity by using MIC. Both the sender and receiver use the CCM* mode together with AES-128 as the underlying block cypher to generate and compare the MIC.

The network layer employs various keys to provide confidentiality and data integrity for end-to-end connections. Four types of keys are defined in the security architecture:

- **Public Keys** which are used to generate MICs on the MAC layer by the joining devices.
- **Network Keys** which are shared by all network devices and used by existing devices in the network to generate MAC MIC's.
- **Join Keys** that are unique to each network device and is used during the joining process to authenticate the joining device with the network manager.
- **Session Keys** that are generated by the network manager and is unique for each end-to-end connection between two network devices. It provides end-to-end confidentiality and data integrity.

Figure 5 describes the usage of these keys under two different scenarios: 1) a new network device wants to join the network and 2) an existing network device is communicating with the network manager. In the first scenario, the joining device will use the public key to generate the MIC on MAC layer and use the join key to generate the network layer MIC and encrypt the join request. After the joining device is authenticated, the network manager will create a session key for the device and thus establish a secure session between them. In the second scenario, on the MAC layer, the DLPDU is authenticated with the network key; on the network layer, the packet is authenticated and encrypted by the session key.

4 Challenges and Solutions

As described in Section 3, WirelessHART includes some core modules, such as time management, mesh networking,

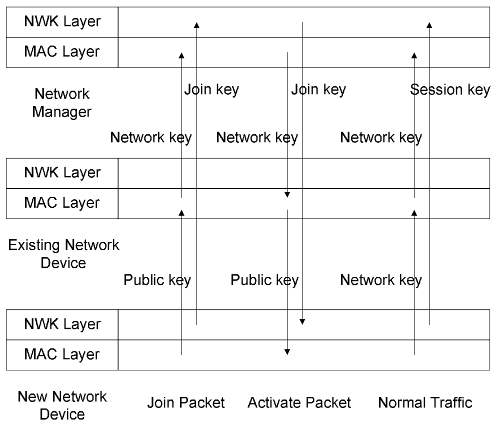


Figure 5. Keying Model

security and network management. It is a very challenging task to build such a prototype on a resource-limited hardware platform.

In the following subsections, we first introduce the hardware platform we use. Then we describe those challenges we met in the process of development and present our solutions.

4.1 Hardware Platform

We base our implementation on the MC1321x evaluation kit [1] provided by Freescale. This toolkit contains one 1321x-NCB (Network Coordinator Board) board, two 1321x-SRB (Sensor Reference Board) boards, and a USB Multilink BDM Programmer/Debugger. The only major difference between 1321x-NCB and 1321x-SRB is that 1321x-NCB has a programmable 2-line LCD for displaying messages. Other than that, the two boards share the following common features:

- 40 MHz 8-bit HCS08 MCU
- 2.4 GHz wireless transceiver compatible with the IEEE 802.15.4 standard
- Programmable 60 KB Flash and 4KB RAM memory
- Multiple 16-bit timers
- USB port to interface with PC
- 3-axis acceleration sensor and temperature sensor
- 4 LEDs and switches for demonstration, monitoring and control

Together with the toolkit, Freescale also provides a simple IEEE 802.15.4 physical layer library in ANSI C. Our task is to build a new WirelessHART protocol stack by using the physical layer library.

Implicitly, this toolkit imposes two restrictions on our implementation: code size and MCU speed. The maximum code size is limited by the flash size, which is 60 KB in this case. Also, the relatively low computation capability of HCS08 makes it difficult to meet the timing requirement inside a time slot. We describe our design to solve these problems in the following subsections.

4.2 Timer and Timer Interrupts

WirelessHART has very stringent timing requirements on each network device. A $10ms$ time slot is further sliced into several time intervals, each of which ranges from $100\mu s$ to $4.5ms$. For example, as shown in Figure 3, a receiver must start listening *TsRxOffset* time units after the beginning of a time slot. In addition, A receiver must acknowledge a packet within *TsMaxPacket+TsTxAckDelay* time units after the arrival of the first bit of the packet. Some of the time intervals are very short. For instance, *TsCCA*, the CCA detection time, is defined to be $128\mu s$.

Also note that an ACK DLPDU is required to carry a 2-byte time adjustment field measured in microseconds. Thus, the timer used in WirelessHART MAC must be precise enough to count in microseconds.

During each time interval defined in a time slot, a node can either be idle or perform some tasks if necessary. Some of those tasks may be very time consuming. For example, when a node receives a DLPDU, it has to first verify the MIC (message integrity code) and then prepare the corresponding acknowledge. Ideally, those tasks should be finished within the designated time interval. However, in practice, the execution may take a longer time and by the time those tasks are completed, the predefined next time interval already starts. In this case, the subsequent tasks will be in serious troubles. Consequently, we can not wait till the end of all tasks in current interval to set the next timer. Instead, we decide to make the timer WirelessHART -aware and use the timer module to start/stop the tasks.

We use a separate 16-bit TPM (Timer/Pulse Width Modulator Module) module to implement the timer. The TPM module's input clock is set to the bus clock (16MHz). By changing the internal prescaler of the TPM module, we can change the clock frequency of the timer as follows:

$$f_{timerclock} = \frac{f_{busclock}}{prescaler}$$

Currently, the prescaler is set to 16. As a result, each tick of the timer is $1\mu s$, which is precise enough to meet the requirement of WirelessHART MAC. The TPM module contains one free running counter and one comparison counter. Whenever the free running counter equals the comparison counter, a timer interrupt is triggered.

By adjusting the comparison counter and maintaining some internal data structures, the timer module can simulate several software timers. The caller of the timer module indicates what type the next time slot would be. Then the timer module generates a sequence of timeout events in the slot based on the given time slot type (transmit/receive/idle). As an example, if current slot is a *receive* slot, the timer would first generate an interrupt at the start of the slot. Then, *TsRxOffset* time units later, it would automatically generate another interrupt to the MAC, informing the MAC to put the transceiver in the listening mode. Conceptually, an interrupt handler is composed of two parts: synchronous part and asynchronous part. The synchronous part resides in the interrupt handler, whereas the asynchronous part is included in the MAC state machine. Time critical and light-weight jobs are put in the synchronous part, and less time critical and computation intensive jobs are put in the asynchronous part. For the second interrupt in the example above, the interrupt handler only needs to set the mode of the transceiver and change some internal states, which can all be put in the synchronous part and leave the asynchronous part empty. However, an asynchronous part is needed when some time-consuming job is incurred, such as data encryption and decryption. In this case, at the very end of the interrupt handler, a specific event is sent to the MAC state machine to signal the execution of the asynchronous part. The interrupt handler finishes immediately after that.

4.3 Synchronization

As time is divided into time slots and transactions within a time slot follow specific timing requirements, it is crucial that nodes in the network are kept in synchronization.

When it joins a WirelessHART network initially, a node has no idea what current time is. Fortunately, for each incoming DLPDU, a node records the time when the DLPDU's first bit arrives. Because of the strict time slot structure, a node can derive the start of the next time slot from the DLPDU arrival time according to the following formula:

$$T_{\text{next\_slot}} = \text{arrival\_time} + 10ms - \text{TsTxOffset}$$

Synchronization happens not only in the join process, but also during a node's normal operations. A receiving node always compares the start time of the incoming DLPDU and the expected arrival time measured in its own clock. The difference is the drift between their clocks. The receiver includes the difference in the time adjustment field of the corresponding ACK packet. Each node is designated a time source node. Whenever a node receives an ACK from its time source, it will adjust its clock based on the time adjustment field. If the sender is the time source of the receiver, the receiver adjusts its clock directly from the time difference value.

4.4 State Machine Design

The major part of WirelessHART MAC layer is a complicated state machine. Each run of the state machine contains three steps:

1. Call the link scheduler to determine the next slot to be serviced.
2. Receive the "time slot start" event from the timer and increment the ASN by 1.
3. When it is time to service the given time slot derived in step 1), execute the associated transaction.

Most of the code in the state machine deals with executing a transaction. We define six states in the state machine:

- **Join:** In this state, the device is not authorized by the network manager yet. After successfully joining the network, it enters the Idle state.
- **Idle:** When the device successfully joins the network or finishes transmiting/receiving a packet, it enters this state.
- **Talk:** When ready to transmit a packet, the state machine enters this state and calls the XMIT engine.
- **Wait ACK:** After a non-broadcast DLPDU is transmitted successfully, the state machine reaches this state.
- **Listen:** In this state, the state machine calls the RECV engine to wait for an incoming DLPDU.
- **Answer:** In this state, the state machine constructs and sends out an ACK DLPDU corresponding to the DLPDU received in the previous Listen state.

Based on these internal states and the incoming event type, the state machine knows what task to execute. For example, when it receives a "time slot start" event, it will first increase the ASN by 1. Then, if current slot is a *transmit* slot, it sets the transceiver to transmit mode and enters into the "Talk" state.

4.5 Network Data Model Design

Similar to the MAC layer, the network layer maintains a set of tables, including the session table, transport table, and route table.

The session table is central in the design as all the end-to-end communication in WirelessHART is built upon the concept of secure session. A session establishes a secure data pipe between the device and one of its correspondent devices. The transport table is used to support end-to-end acknowledged transactions with automatic retries. It uses a MASTER bit to identify whether the device is a MASTER or a SLAVE. Along with the corresponding sequence number, this table also buffers the payload of the last request (in MASTER mode) or response (in SLAVE mode). Thus it allows the device to resend the request or response when the retry timer expires.

For each destination, there can be more than one entries in the route table with different graph IDs. When generating a network layer packet, WirelessHART has to consult the route table, superframe table, and graph table together to determine the routing information to be used. For certain destination, there can also exist a source route, which is mainly used for network diagnostics.

With these well-organized communication tables and graph/source routing protocols, WirelessHART supports various network topologies and provides reliable end-to-end communications.

4.6 Security

In the MAC layer, WirelessHART provides data authentication service. The authentication service uses CCM* mode(Counter with CBC-MAC (corrected)) [16] with AES-128 [11] as the underlying block cypher. CCM* needs 4 byte-strings as parameters (a, m, N, K). As the DLPDU is not encrypted, the second parameter m is empty, while a includes the DLPDU header and payload.

The key K is 16-byte long. The value of K depends on the current status of the node. If a node is joining a network or broadcasting a network advertisement, the well-known key $0\text{x}7777\,772E\,6861\,7274\,636F\,6D6D\,2E6F\,7267$ is used. In all other situations, the network key assigned by the network manager is used.

The nounce N is 13-byte long and is the concatenation of the absolute slot number and the source address. The first 5 bytes are always the absolute slot number. If the source address of the DLPDU is a long address (EUI-64 address), the source address is filled into the remaining 8 bytes of the nounce. Otherwise, the short source address(2 bytes) is put right next to the slot number, with the rest 6 bytes filled with 0.

The sender and receiver of the DLPDU both call the *CCM** function with the same input: the DLPDU header and payload. After receiving a DLPDU, the receiver compares the returned MIC with the MIC in the original message. If they match, the message is authenticated. Otherwise the message is invalid and discarded.

From the perspective of a receiver, it must run *CCM** on the received message and on the corresponding ACK message within $TsTxAckDelay(1ms)$. This is a very challenging task for low power processors.

We grabbed a complete *CCM** program from [6]. This program only encrypts/decrypts one block of data, which is 16 bytes. There are three major underlying functions in the program: $aes\_set\_key()$, $aes\_encrypt()$, and $aes\_decrypt()$. We measured the execution time of these three functions on the hardware platform described in previous section. On average, $aes\_set\_key()$ takes $1341\mu s$, $aes\_encrypt()$ takes $1335\mu s$, and $aes\_decrypt()$ takes $1581\mu s$. The numbers are just too big for WirelessHART. It should also be noted that those functions only works on 16 bytes, while a WirelessHART DLPDU can be as long as 121 bytes.

Then we tested the CCM* program on the new and more powerful FreeScale EVBQE128 kit. The run time for the three major functions $ccm\_init\_and\_key()$, $ccm\_encrypt\_message()$, $ccm\_decrypt\_message()$ are $77\mu s$, $795\mu s$, $818\mu s$, respectively. Obviously, without any tweak, the EVBQE128 toolkit would not meet the requirement of WirelessHART either.

We propose to use a hardware accelerator to speed up the encryption/decryptoin process. Upon our request, Freescale is developing a new toolkit with on-board hardware accelerator. Freescale claims that the encoding of 16 bytes of data would be finished in 13 system clocks with the new toolkit. However, even with the new toolkit, we may still have problems. Except for the $aes\_encrypt()$ function, $ccm\_encrypt\_message()$ and $ccm\_decrypt\_message()$ also call some other helper functions. Of the $795\mu s$ and $818\mu s$ run times of the two functions, the helper functions take about $123\mu s$ and $148\mu s$, respectively. These times can not be improved by the hardware accelerator.

In order to further speed up the encryption/decryption process, we propose to execute *CCM** incrementally. Originally, *CCM** is not designed to support stream processing. However, as WirelessHART DLPDU is not encrypted, the message length is indicated in the DLPDU header. Thus, we can run *CCM** on an incoming message as soon as every 16 bytes are received. Given the relatively slow data transmission rate (250kbps), we may only need to process one block of data in the $TsTXAckDelay$ period, regardless of the message length. In this way, we can meet the stringent timing requirements of WirelessHART.

WirelessHART also provides built-in security support in the network layer. It also uses *CCM** mode. As there is no time slot concept in the network layer, we do not have problems with message encryption/decryption in this layer.

4.7 Network Management

According to WirelessHART, two most important functions of a network manager are generating routes and communication schedules. Although WirelessHART specifies what a network manager should do in various cases, it leaves out all details. That is, the network manager has the freedom to choose how to fulfil its tasks. For example, when generating routes, the network manager can aim to either balance the load on network nodes or minimize the average network latency. Depending on the metrics the manager tries to optimize, the resulting routes can be very different.

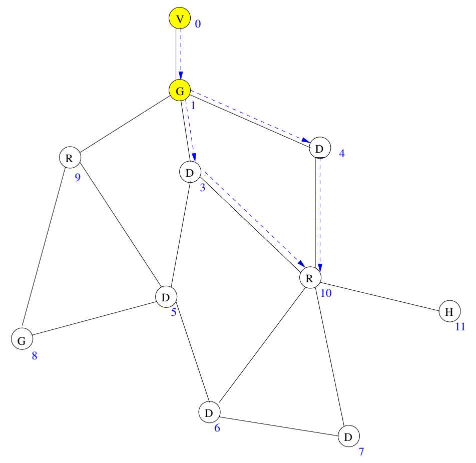


Figure 6. Topology of the WirelessHART Network in Fig. 4

For the purpose of proof of concept, we implemented a simple network manager. We illustrate the design of the network manager by an example. The topology of the example network in Figure. 4 is shown in Figure. 6, where the number below each node is the node's identification number and there are two paths from the gateway to device 10. The burst mode data rate from each device is summarized as follows:

Device ID	3	4	5	6	7	8
Burst Mode Rate (seconds)	8	32	4	64	16	64

4.7.1 Generating Routes

Based on the link information provided by each node, the network manager needs to create the overall network graph. In this process, the manager follows several rules in the specified order:

(a) Minimize the number of hops.

(b) Route through powered devices if they are available.

(c) Use signal strength to select the best paths to neighbors.

(d) Use a combination of weighted signal strengths to select between alternative routes.

(e) Prune the number of neighbors to 4 or less.

After generating the overall network graph, the manager has to create the follow graphs:

- An graph describing paths from each network device to the gateway
- A broadcast graph from the gateway downward to each device
- Graphs from the gateway to each device

4.7.2 Generating Communication Schedules

The network manager has to assign time slots to each device according to the graphs derived above. The strategies we take are summarized below:

(a) The network management superframe has priority over data superframes.

(b) Traverse the graph by breath-first search, starting from the gateway, number the devices as $N_0, N_1, \ldots, N_n$.

(c) Every device needs to have a slot for a keep-alive message. The keep alive timer is 60 seconds.

(d) For join requests, from the furthest devices, allocate one link for each en-route network device to the gateway (No redundancy provided)

(e) For join responses, traverse the graph by breath-first search, allocating one link for each en-route network device from the gateway to end network device.

(f) For data requests, the allocation is similar to join requests. An additional slot is allocated in each en-route device for retransmission.

(g) If there is an alternative path, allocate a slot for it.

The resulting schedule is shown in Figure. 7. The upper part of the figure shows the overall slot allocations divided into logical channels. The bottom portion of the schedule describes the transmit slots and receive slots for each device.

After the overall schedule is generated, the network manager splits the schedule into sub-schedule for each device and distributes them to the corresponding devices.

5 A WirelessHART Network Demonstration

Based on the WirelessHART prototype stack, we continued to develop some applications on the stack, with the aim of building a demonstration network with the Freescale 1321xEvk toolkit. The application is written in ANSI C.

The demonstration network, shown in Fig. 8, contains one gateway and two devices, which are referred to as *Device 1* and *Device 2*. In this network, *Device 1* designates the Gateway as its time source, and is the time source of the *Device 2*. The gateway and Device 2 both maintain a 4-bit counter, with the gateway counting upward and Device 2

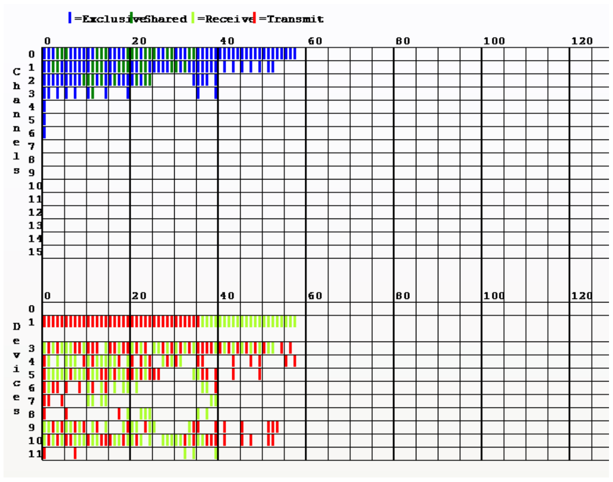


Figure 7. The Overall Schedule for the Sample Network

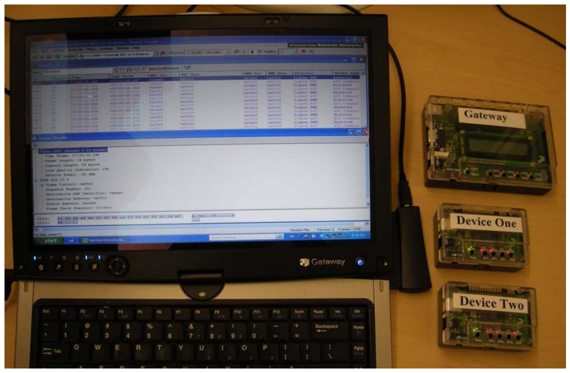


Figure 8. The Demonstration Network

counting downward. They exchange the counter values via Device 1 and show the received value on the LEDs. The first two LEDs on Device 1 display the lower two bits of the gateway's counter and the remaining two LEDs on Device 1 shows the lower two bits of Device 2's counter. In this way, we can check the communication status between any two devices simply from the LEDs.

We also define an overall superframe for the network, shown in Table 1. Basically, this superframe regulates the time sequence of all communications. That is, the gateway first transmits a packet to Device 2 via Device 1. Device 2 sends out its counter value after it receives the packet from the gateway.

Since WirelessHART works on top of 802.15.4 physical layer and adopts 802.15.4 DLPDU format, we can capture the communication details on packet level by using a hardware sniffer and 802.15.4 protocol analyzer. These two components are also shown in Fig. 8. With the help of the analyzer, we have verified the following points:

(a) A device can synchronize to its time source within 3

Table 1. The superframe configuration

Time Slot	Transaction
0	Gateway → Device 1
1	Device 1 → Device 2
2	Device 2 → Device 1
3	Device 1 → Gateway

time slots.

(b) A DLPDU is always immediately acknowledged in the same time slot.

(c) Both the gateway and Device 2 get the other party's counter value via Device 1. That is, Device 1 is forwarding data packets for the two devices.

This demonstration proves that our design is feasible and the implementation works. More details on the demo can be found at [9].

6 Discussion and Conclusion

WirelessHART is the first open wireless standard for industrial process control. To meet the strict real-time requirements of process control, WirelessHART specifies many challenging features. In this paper, we briefly introduce the new WirelessHART standard, focusing on the overall architecture. Then we present some challenges in implementing this specification, such as timer design, synchronization and security. We proposed some novel ways to tackle the challenges. Preliminary results from our prototype are very encouraging and also revealing. Based on experiments with the hardware, we determine that there is a serious need for using AES hardware accelerator to realize the $10ms$ time slot.

WirelessHART is a layered and unique protocol stack. We have thus far focused on implementing the major features of WirelessHART. For future work, we will build a full-featured WirelessHART prototype. For example, there is much work to be done on the transport layer and application layer.

Another avenue of future work is the network manager. The network manager is the central control unit of a WirelessHART network. When deriving the routing table and communication schedules, the network manager can choose to optimize several metrics, such as energy consumption, average end-to-end latency. We believe the scheduling algorithms can be vastly different depending on the optimization goal. For now, our network manager is a proof of concept and far from perfect.

Beyond that, we are also interested in the co-existence issues between WirelessHART, ZigBee and Bluetooth. As they all work on the 2.4G ISM band, it would be interesting to see how they would react in a mixed environment.

References

[1] 1321xEVK Product Summary. `http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=1321xEVK%`.

[2] Bluetooth. `http://www.bluetooth.com`.

[3] HART Communication. `http://www.hartcomm2.org/index.html`.

[4] IEEE 802.11 Task Group. `http://grouper.ieee.org/groups/802/11/`.

[5] IEEE 802.15.4 WPAN Task Group. `http://www.ieee802.org/15/pub/TG4.html`.

[6] Implementation of aes in c/c++ and assembler. `http://fp.gladman.plus.com/cryptography_technology/rijndael/index.htm`.

[7] ISA100: Wireless Systems for Automation. `http://www.isa.org/MSTemplate.cfm?MicrositeID=1134&CommitteeID=6891`.

[8] WINA. `http://www.wina.org/`.

[9] WirelessHART Demo. `http://www.cs.utexas.edu/~cdj/WhEpmDemo.mpg`.

[10] ZigBee Alliance. `http://www.zigbee.org`.

[11] F. P. 197. *Advanced Encryption Standard (AES)*. U.S. DoC/NIST, November 2001.

[12] K. Akkaya and M. Younis. A survey of routing protocols in wireless sensor networks, 2005.

[13] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. In *IEEE Communication Magazine*, August 2002.

[14] T. Blevins, G. McMillan, W. Wojsznis, and M. Brown. *Advanced Control Unleashed: Plant Performance Management for Optimum Benefit*. ISA Press, 2002.

[15] D. Caro. *Wireless Networks for Industrial Automation*. ISA Press, 2004.

[16] M. Dworkin. *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*. NIST Special Publication 800-38C, May 2004.

[17] J. Edgar H. Callaway and E. H. Callaway. *Wireless Sensor Networks: Architectures and Protocols*. CRC Press, August 2003.

[18] R. B. Jose A. Gutierrez, Edgar H. Callaway. *IEEE 802.15.4 Low-Rate Wireless Personal Area Networks: Enabling Wireless Sensor Networks*. IEEE, April 2003.

[19] J. Song, S. Han, A. K. Mok, D. Chen, M. Lucas, and M. Nixon. A study of process data transmission scheduling in wireless mesh networks. In *ISA EXPO Technical Conference*, Oct. 2007.

[20] J. Song, A. K. Mok, D. Chen, M. Nixon, T. Blevins, and W. Wojsznis. Improving pid control with unreliable communications. In *ISA EXPO Technical Conference*, 2006.

[21] J. Stankovic, T. Abdelzaher, C. Lu, L. Sha, and J. Hou. Real-time communication and coordination in embedded sensor networks, 2003.

[22] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks, 2002.