Implementation and Evaluation of Communication Tables and Link Scheduling for WirelessHART Devices

Esam Alkureishi
Master of Science Thesis 2010
Embedded System
Electrical Engineering
School of Engineering
Jönköping University
Implementation and Evaluation of Communication Tables and Link Scheduling for Wireless Devices

Esam Alkureishi
This thesis work is performed at Jönköping institute of Technology within the subject area Embedded System. The work is part of the university’s engineering degree. The author is responsible for the given opinion, conclusions and result.

Supervisor: Youzhi Xu
Examiner: Youzhi Xu
Credit points:
Date:
Archive number:
Abstract

WirelessHART is a wireless mesh network communication protocol that supports the requirement for the process automation applications. WirelessHART network contains many devices that perform a specific operation. Each device in the WirelessHART maintains two types of communication tables used for controlling communication between the devices and collecting statistics of these communications. One group of the tables is maintained in the network layer and the other one is maintained in the data link layer.

In this thesis we have concentrated on improvement and implementation of the communication tables for the network manager and other wireless devices in the data link layer of the WirelessHART networks. In addition, we have implemented link scheduling process for wireless devices when related communication tables are received from the network manager. Results in simulation show that our improvements make more effective power management for resource constrained wireless devices in WirelessHART networks.
Sammanfattning


Acknowledgement

In the beginning I would like to thank my examiner and supervisor professor Youzhi Xu for supporting and guiding to finished this thesis. He gave us a road map to start and finish this thesis.
I would like to thank master program coordinator Alf Johanson who gave me a chance to join with Embedded System Master Program and working on my thesis. I would like to thank Dr.Dong Yang who gave us great helpful suggestions. I would like to thank guest researcher Dr.Wei Shen who gave us a useful idea to implement the difficult tasks in this thesis.
I would like to thank all our teachers in the Embedded System Department who prepared us and gave us a useful idea during course study to make the thesis is easy in implementation.
Finally I would like to thank all my classmates who help me during course study and final thesis study.
# Table of contents

1. Introduction
   1.1 Background ................................................................. 1
   1.2 Purposes ........................................................................... 1
   1.3 Constraints ....................................................................... 2

2. Theoretical Background
   2.1 Wireless sensor network .................................................... 2
   2.2 IEE 802.15.4 ..................................................................... 3
   2.3 HART and WirelessHART ................................................... 3
   2.4 Embodiment ...................................................................... 7
   2.5 TinyOS ............................................................................. 8
      2.5.1 Programming structure ................................................. 9
      2.5.2 Components, interfaces, and wiring ............................... 10
      2.5.3 Wiring and callbacks .................................................... 11

3. Related Works .................................................................... 13
   3.1 Time Synchronized Mesh Protocol ...................................... 13
   3.2 Time Synchronized Channel Hopping .................................. 13
     3.2.1 TSCH’s Data-Link packets .......................................... 15
     3.2.2 TSCH’s Neighbors Table ............................................. 18
   3.3 Overview on TSCH stack ................................................... 18
     3.3.1 TSCHAppC Component ................................................. 18
     3.3.2 TSCH Component ....................................................... 18
     3.3.3 Reservation Component .............................................. 19
     3.3.4 CellUsage Component ................................................. 19
     3.3.5 KeepAlive Component ............................................... 20
     3.3.6 Global Time Component .............................................. 20
     3.3.7 Multiplex Time Component .......................................... 21
     3.3.8 TDMASlot Component ................................................ 21
     3.3.9 TSCHQueue Component ............................................. 21
     3.3.10 Advertise Component ............................................... 22
     3.3.11 Neighbors Component ............................................... 22
     3.3.12 Forwarding Component ............................................. 23
     3.3.13 ActiveMessageAddress Component ........................... 23

4. Improvement of communication tables for WirelessHART devices
   4.1 Communication Tables ....................................................... 25
      4.1.1 Superframe table ....................................................... 26
List of figures

Figure: 1. Typical Multi-Hop Wireless Sensor Network Architecture 2
Figure: 2. HART and WirelessHART OSI 7-Layer Model 4
Figure: 3. Graph routing 6
Figure: 4. Source routing 7
Figure: 5. Wireless HART mesh network 7
Figure: 6. WirelessHART Network of the our project 8
Figure: 7. PowerupC module in nesC 9
Figure: 8. PowerupAppC configuration in nesC 10
Figure: 9. Wiring diagram for Powerup application 10
Figure: 10. Powerup with blinking LED in nesC 12
Figure: 11. Powerup with blinking LED configuration 12
Figure: 12. Slot-Channel matrix for a network with 5 channels 13

Figure: 13. Schedule for 4 nodes A, B, C and D mesh network, slotframe is 10 slots 14

Figure: 14. cc2420 message-t 15
Figure: 15. cc2420 header structure of cc2420 15
Figure: 16. Advertise TSCH packet structure 16
Figure: 17. Data TSCH packet 16
Figure: 18. Reservation TSCH packet 17
Figure: 19. The TSCH packets type 17
Figure: 20. TSCH components stack 24
Figure: 21. WirelessHART Data Link Layer Architecture 25
Figure: 22. Communication Tables 26
Figure: 23. Overall Superframes for Network contain two devices with four times Slot for each superframe 27
Figure: 24. WirelessHART NPDU structure 31
Figure: 25. DLPDU packet Structure 33
Figure: 26. TSCH stack 35

Figure: 27. Structure of Link Table 36
Figure: 28. Structure of Superframe 37
Figure: 29. Network Manager Device sends Superframe and Link tables To Device 37
Figure: 30. Occupation Data field of message-t by Superframe and Link 38
Figure: 31. Construction of WirelessHART Network packet 38
Figure: 32. Address possibility of destination of WirelessHART Network packet 38
Figure: 33. Link Scheduling process 39,40
Figure: 34. TSCH neighbor. h file including WirelessHART Neighbor Table Entries 41
Figure: 35. Getting lastTimecommunicated and number of Packets transmitted entries of WirelessHART Neighbor Table 42
Figure: 36. Getting Neighbor ID entry of WirelessHART Neighbor Table 42
Figure: 37. Getting join priority entry 43
Figure: 38. Finding timesourceflag entry 44
Figure: 39. The process to find packet 45
Figure: 40. Finding the misackpkt entry of Neighbor Table 46
Figure: 41. Finding Boexp and BoCntr 47
Figure: 42. Getting status and time path failure timer entries of Neighbor 48

Figure: 43. Prepare pc to print from a mote 50
Figure: 44. Useful commands 51
List of Tables

Table: 1. Neighbor table entry of TSCH 18
Table2. Superframe contents 26
Table: 3. Link table 28
Table: 4. Graph Table 29
Table: 6. Neighbor Table Entries of WirelessHART 30
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HART</td>
<td>Highway Addressable Remote Transducer Protocol</td>
</tr>
<tr>
<td>TinyOS</td>
<td>Tiny Operating System</td>
</tr>
<tr>
<td>NesC</td>
<td>Language used with TinyOS</td>
</tr>
<tr>
<td>MAC</td>
<td>Multiple Access Control Layer</td>
</tr>
<tr>
<td>PHY</td>
<td>Physical Layer</td>
</tr>
<tr>
<td>NEK</td>
<td>Network Layer</td>
</tr>
<tr>
<td>TSCH</td>
<td>Time synchronization Channel Hopping</td>
</tr>
<tr>
<td>OSI model</td>
<td>Open Systems Interconnection model</td>
</tr>
<tr>
<td>TDMA</td>
<td>Time Division Multiple Access</td>
</tr>
<tr>
<td>Src</td>
<td>Source</td>
</tr>
<tr>
<td>Dest</td>
<td>Destination</td>
</tr>
<tr>
<td>DLPDU</td>
<td>Data Link Protocol Data Unit</td>
</tr>
<tr>
<td>Tx</td>
<td>Transmitter</td>
</tr>
<tr>
<td>Rx</td>
<td>Receiver</td>
</tr>
<tr>
<td>NPDU</td>
<td>Network Protocol Data Unit</td>
</tr>
<tr>
<td>ASN</td>
<td>Absolute Slot Number</td>
</tr>
<tr>
<td>XMIT</td>
<td>Transmitter</td>
</tr>
<tr>
<td>RECV</td>
<td>Receive</td>
</tr>
<tr>
<td>AM-TSCH-ADV</td>
<td>Advertise Time Synchronization Channel Hopping packet</td>
</tr>
<tr>
<td>AM-TSCH-DATA</td>
<td>Data Time Synchronization Channel Hopping packet</td>
</tr>
<tr>
<td>AM-TSCH-RES</td>
<td>Reservation Time Synchronization Channel Hopping packet</td>
</tr>
<tr>
<td>AM-TSCH-ACK</td>
<td>Acknowledgement Time Synchronization Channel Hopping packet</td>
</tr>
<tr>
<td>TSCHAppC</td>
<td>Time synchronization Channel Hopping Application Component</td>
</tr>
<tr>
<td>TSCHC</td>
<td>Time synchronization Channel Hopping Component</td>
</tr>
<tr>
<td>ReservationC</td>
<td>Reservation Component</td>
</tr>
<tr>
<td>CellUsageC</td>
<td>Cell Usage Component</td>
</tr>
<tr>
<td>KeepAliveC</td>
<td>Keep Alive Component</td>
</tr>
<tr>
<td>GlobalTimeC</td>
<td>Global Time Component</td>
</tr>
<tr>
<td>MultiplexC</td>
<td>Multiplexer Component</td>
</tr>
<tr>
<td>AdvertiseC</td>
<td>Advertise Component</td>
</tr>
<tr>
<td>NeighborsC</td>
<td>Neighbors Component</td>
</tr>
<tr>
<td>ForwardingC</td>
<td>Forwarding Component</td>
</tr>
<tr>
<td>ActiveMessage</td>
<td>Active Message Address Component</td>
</tr>
<tr>
<td>TSCHP</td>
<td>Synchronization Channel Hopping Private Component</td>
</tr>
</tbody>
</table>
1. Introduction

Wireless HART is an open wireless communication standard used for control application and process measurement. There are many standards that are available before Wireless HART is released, such as ZigBee and Bluetooth. They have many problems, such as lack security, they could not meet the industrial requirements and some of them do not provide a guarantee on end-to-end communication. ZigBee need to build in channel hopping and Bluetooth is limited, not scalable, and only supports star topology. The new Wireless HART is designed to solve these problems.

In this report we will analysis and describe in detail the communication tables that should be maintain by each device in the Wireless HART network. Then we will build a prototype to maintain the Network Layer tables, and also build prototype to implement the link scheduling by each device in the Wireless HART network. This report contains the analyzing the TSCH stack code.

1.1 Background

WirelessHART

Wireless sensor technology is widely used in many applications; it is used in telecommunication industry system which is referred to radio transmitter and receiver, remote controls, computer networks, network terminals and many others [1]. Since no technology is right for every application the Wireless HART technology is attempt to cover process monitoring and control applications, including, monitoring the process, equipment and environment.

Wireless HART technology support wired instrumentation and plants. It is backward compatibility; including the HART command structure and Device description language, makes it is easy to support wire and wireless device to use the same tools.

Wireless HART adopts device ”interchangeability”, to provide true interoperability, which means that the selection of the new Wireless HART devices that replace the old Wireless HART devices regardless of manufacture. This new replaced device can work with complete functionality without any losses in a system [2].

1.2 Purpose

The purpose of this thesis is implementation the Wireless HART network, maintain the superframe and link table by the network manager, sending the superframe and link table from the network manager to a device, maintain the
neighbor table and link scheduling process by the device. The second purpose is analyzing the TSCH code by using source insight tool.

1.3 Constrains

We will use two sensor to implement our tasks because the limitation of the economic resources. First sensor represents the Network Manager Device and gateway at the same time. The second sensor represents the device in the Wireless HART network.

2 Theoretical Background
2.1 Wireless sensor network

Wireless sensor networks consist of sensors, each sensor called node, as shown in Figure 1. The purpose of these sensors is monitoring the environment such as temperature, pressure, level liquid, sound and vibration.

![Wireless sensor network diagram]

It recently used in many applications such as military, civilian and industrial area. Each node in the network is provided with radio transceiver equipments, a small microcontroller and source voltage which usually a small battery [1]. The size of sensor or mote is different from little centimeter to mini centimeter, it depend on application that used for. The cost of the mote is varying from few dollars to few hundred dollars. A sensor in the network supports multi-hop routing algorithm which means that several sensors may forward packet data to base station.
In our project we use Telosb mote platform which represent a sensor node to perform our tasks. The computation power, cost and memory size of this mote are low.

2.2 IEE 802.15.4

IEE 802.15.4-2006 is a basic standard for WirelessHART specification which is released by IEE 802.15.4 working group. This standard define two layers, physical and media access control layer. WirelessHART is attempting to complete the network solution for other layers that not cover by this standard [5].

IEE standard 802.15.4 focus on low cost and low speed for this reason this standard in the area of wireless networks, home security, home automation, home network and so on. The physical layer (PHY) is responsible for data transmission. It offers of 20 kps (bitrates), a single channel in the frequency range 868-868.6 MHz. It offers 40 kps (bitrates), ten channels in the range 905-928 MHz, and 250 kps (bitrates), 16 channels in the 2.4 GHz ISM band 2.4-2.485 GHz with 5 MHz spacing the center frequencies. The total available channels are 27, but the MAC protocol use only one channel of these at a time [5].

The MAC Layer is responsible for network beacon, secure service, and control frames and guarantees timeslots. In our project we depend on MAC layer to store Superframe, Link table, Neighbor table and link scheduling which perform within this layer.

2.3 HART and WirelessHART

Process automation is that monitoring the values and qualities of outputs by using the computer technology. The sensors are playing main rule in this process since it collect different data, level liquid, temperature, pressure and so on, depending on application. Depending on the data has been collected it will be analyzed the right decision will be taken to improve the production.

This importance of these processes leads to investigate and release many communication protocols by many company and groups. In the beginning they use weird communication but many of them transfer to wireless since installing wireless sensors are cheaper than wire sensors. There are many protocols released to achieve process automation but we will describe just two of them, HART, WirelessHART.
HART communication protocol (Highway Addressable Remote Transducer protocol) is a bi-directional communication between a host application and intelligent field instruments. The host application includes diagnostic and parameter setting [9]. HART is organized around the ISO/OSI 7-layer model, as shown in Figure: 2, HART and WirelessHART are divided into layers [10].

**Physical Layer**

This layer based on the Bell 202 standard, using frequency shifting key (FSK) to communicate. The signal frequencies representing bit values of 0 and 1 are 200 and 1200 Hz respectively. This signal is compressed at low level 4-20mA analogue measurement signal without causing any interfaces with any analogue signal [2].

**Data Link Layer**

This layer defines a master-slave protocol, in normal situation, a field device in the network respond when it is spooking to. The slave represent any field device can respond only to message from master. HART support Burst-Mode transfer which allows a field device to be configured to transmit digital values. All HART hosts must support Burst-Mode, however, utilization of Burst-Mode depend on the application.
Network Layer

It is responsible for providing routing, transport service, end-end security and managing the sessions between two devices.

Transport Layer

It is responsible for ensuring end-end communication is successful.

Application Layer

It is responsible for defining the commands, responses, data types and status.

WirelessHART is a wireless mesh network protocol for process automation application. It is an extension of HART protocol. As we see in figure 4. The OSI Model protocol stack, it consists of 5-layers, Physical Layer, Data Link Layer, Network Layer, Transport Layer and Application Layer.

Physical Layer

IEE 802.15.4-2006 is a basic standard for WirelessHART specification, frequency of operation is 2.4 GHz. The operation frequency lies within ISM band which is unlicensed so the probability of interference is very high.

Data Link Layer

WirelessHART is used both channel hopping and Time Multiple Access (TDMA), since the WirelessHART is working in harsh environment. The main roles of these mechanisms are decreasing the interference with another network. We will describe in detail this layer in section [write no. of section] since our project is based on MAC layer properties.

Network Layer and Transport Layer

The purposes of these layers are providing and secure and reliable end-end communication for network.

Each WirelessHART device should be supported mesh network topology to forward packets to next hop in the network. WirelessHART define two types of routing, Graph routing and Source routing.
**Graph routing** is a directed list of paths that connect end points. The path in each graph is created by the Network Manager and downloaded to each network device individually.

Every graph in a network has unique Graph Id. The device places the Graph Id value in network header NPDU. Devices received the packet then forward it along the set of paths belonging to the Graph to the destination [11]. We will illustrate Graph routing in Figure 3. bellow:

![Figure: 3. Graph routing](image)

N3 communicates with N6 using Graph 1. To send packet on the graph, there are two probabilities N3 either forward it to N2 or N4. Both alternatives take several routes but they end with N6. If N3 would communicate with N5, It sends a packet on Graph 2, either through N2 or N4.

**Source routing** specifies a list of address a packet should take when travelling from the source device to the destination device. Source routing is unidirectional. Source routing can be used by Network devices that have been configured with source routes lists by the Network manager.

As shown in Figure 4, we consider that N2 wishes to send packet to N6. The routing table on N2 may contain <N1, N4 and N5 > as the source route for N6. In that case N2 will put a packet containing < N1, N4 and N5 > in the header (NPDU) and send the packet to N1. N 1, upon receiving the packet will send it to N4 after finding it in the header. N4 will send the packet to N5, then, N5 will send the packet to N6 (final destination) [11].
2.4 Embodiment

Wireless HART network consist of many elements which includes, Field Devices, Handheld, Getaway and Network Manager, as shown in Figure 5. Every element has specific function, Field Devices attach to the plant process and responsible for monitoring the environment. Handheld is portable Wireless HART computer, it performs many functions like; configure devices, calibration and diagnostic. Gateway connects host application to Devices. Network Manager configure network, schedule and manage communication between Wireless HART Devices [3].
In our thesis we use two sensors, one of them represents Network Manager, Gateway and Device at same time and the other represents Field Device as shown in Figure 6.

![Figure 6. WirelessHART Network of the our thesis](image)

### 2.5 TinyOS

A TinyOS is an open-source operating system designed for wireless embedded sensor network. It is component based-architecture. The component library of the TinyOS includes network protocols, distributed services, sensor drivers and data acquisition tools. It is event-driven execution model [3].

TinyOS are written in nesC, almost same C programming language but it is modify to meet the requirement of the wireless sensor networks which has limited memory. All TinyOS programming are maintain from software components. In order to connect the components we use the interface to do that. Each interface has functions; it may either event functions, or command functions, even may it has both types of functions.

TinyOS has a single stack that means all input / output operations that take a few hundreds microsecond are asynchronous and have callback, it allows a low software layer to call function or subroutine defined in a higher-level layer [4]. TinyOS use events feature to link all these callback. TinyOS supports tasks to implement more computations. Any TinyOS component can post this task in order to these tasks will schedule by the OS to run later.
2.5.1 Programming structure

Programming structure is the most essential and obvious difference between C and nesC. C programs are composed of variable, types, and functions defined in files that are compiled separately and then linked together. The nesC programs are built out of components that are connected (“wired”) by explicit program statements; the nesC compiler connects and compiles these components as a single unit [15].

The nesC program consist of two parts, first part is “module” contains the executable logic of nesC program. The other part is “configuration”, specifies how nesC program is connected to TinyOS's services [15].

In order to illustrate two concepts by taking a simple example program in nesC. The program is “Powerup” application that turns on one of the motes LEDs at boot, then goes to sleep.

```c
module PowerupC {
    uses interface Boot;
    uses interface Leds;
}

implementation {
    event void Boot.booted() {
        call Leds.led0On();
    }
}
```

*Figure 7. PowerupC module in nesC*

The first part of PowerupC is “module” as shown in figure 7. This code says that PowerupC interacts with the rest of the system via two interfaces, Boot and Leds, and provides an implementation for the booted event of the Boot interface that calls the led0On2 command of the Leds interface. The nesC programs are built out of components that implement a particular service (in the case of PowerupC, turning a LED on at boot-time). The interactions between components are specified by interfaces: the interface’s user makes requests (calls commands) on the interface’s provider; the provider makes callbacks (signals events) to the interface’s user [15].

Commands and events themselves are like regular functions (they can contain arbitrary C code); calling a command or signaling an event is just a function call. PowerupC is a user of both Boot and Leds; the booted event is a callback signaled when the system boots, while the led0On is a command requesting that LED 0 be turned on.
The second part of Powerup is PowerupAppC as shown in figure 8.

```c
configuration PowerupAppC [ ]
implementation [
    components MainC, LedsC, PowerupC;
    MainC.Boot -> PowerupC.Boot;
    PowerupC.Leds -> LedsC.Leds;
}
```

*Figure 8. PowerupAppC configuration in nesC*

This says that the PowerupAppC application is built out of three *components* (modules or configurations), MainC (system boot), LedsC (LED control), and PowerupC (our powerup module). PowerupAppC explicitly specifies the connections (or *wiring*) between the interfaces provided and used by these components. When MainC has finished booting the system it *signals* the booted event of its Boot interface, which is connected by the wiring in PowerupAppC to the booted event in PowerupC. This event then calls the led0On command of its Leds interface, which is again connected (*wired*) by PowerupAppC to the Leds interface provided by LedsC. Thus the call turns on LED 0. The resulting component diagram is shown in Figure 9 – this diagram was generated automatically from PowerupAppC by nesdoc, nesC’s documentation generation tool.

*Figure 9. Wiring diagram for Powerup application [15]*

### 2.5.2 Components, interfaces, and wiring

In nesC – components, interfaces, and wiring – all relate to naming and organizing a program’s elements (variables, functions, types, etc).
The nesC’s components provide a more systematic approach for organizing a program’s elements. A component (module or configuration) groups related functionality (a timer, a sensor, system boot) into a single unit, in a way that is very similar to a class in an object-oriented language. For instance, TinyOS represents its system services as separate components such as LedsC (LED control, seen above), ActiveMessageC (sending and receiving radio messages), etc. Only the service (component) name is global, the service’s operations are named in a per-component scope: ActiveMessageC. SplitControl starts and stops the radio; ActiveMessageC.AMSend sends a radio message, etc [15].

Interfaces bring further structure to components: components are normally specified in terms of the set of interfaces (Leds, Boot, SplitControl, and AMSend) that they provide and use, rather than directly in terms of the actual operations. Interfaces simplify and clarify code because, in practice, interactions between components follow standard patterns: many components want to control LEDs or send radio messages, many services need to be started or stopped, etc. Encouraging programmers to express their components in terms of common interfaces also promotes code reuse: expressing your new network protocol in terms of the AMSend message transmission interface means it can be used with existing applications, using AMSend in your application means that it can be used with any existing or future network protocol [15].

Rather than connect declarations to definitions with the same name, nesC programs use wiring to specify how components interact: PowerupAppC wired PowerupC’s Leds interface to that provided by the LedsC component, but a two-line change could switch that wiring to the NoLedsC component (which just does nothing):

```
components PowerupC, NoLedsC;
PowerupC.Leds -> NoLedsC.Leds;
```

### 2.5.3 Wiring and callbacks

Leaving the component connection decisions to the programmer does more than just simplify switching between multiple service implementations. It also provides an efficient mechanism for supporting callbacks, as we show through the example of timers. TinyOS provides a variable number of periodic or deadline timers; associated with each timer is a callback to a function that is executed each times the timer fires [15].

The nesC version of Blink is used interfaces and wiring to specify the connection between the timer and the application as shown figure 10:
module BlinkC {
    uses interface Boot;
    uses interface Timer;
    uses interface Leds;
}
implementation {
    event void Boot.booted() {
        call Timer.startPeriodic(250);
    }
    event void Timer.fired() {
        call Leds.led0Toggle();
    }
}

Figure 10. Powerup with blinking LED in

The BlinkC module starts the periodic 250 ms timer when it boots. The connection between the startPeriodic command that starts the timer and the fired event which blinks the LED is implicitly specified by having the command and event in the same interface:

interface Timer {
    command void startPeriodic(uint32_t interval);
    event void fired();
    ...
}

Finally, this Timer must be connected to a component that provides an actual timer. BlinkAppC wires BlinkC.Timer to a newly allocated timer MyTimer as shown in figure 11:

configuration BlinkAppC {
}
implementation {
    components MainC, LedsC, new TimerC() as MyTimer, BlinkC;
    BlinkC.Boot -> MainC.Boot;
    BlinkC.Leds -> LedsC.Leds;
    BlinkC.Timer -> MyTimer.Timer;
}

Figure 11. Powerup with blinking LED
The connection between the timer and the Blink application is specified at compile-time in BlinkAppC. This is saving RAM, and allows the nesC compiler to perform optimizations across callbacks [15].

3 Related Works

3.1 Time Synchronized Mesh Protocol

Time Synchronized Mesh Protocol, TSMP, is a communication protocol for self-organized networks of wireless device. TSMP devices communicate during specific time slots and still synchronized with each other, same as TDM, Time-division multiplexing system. These properties allows the devices to save energy since the transmission/receiving data occur during the scheduling periods and the device is on just during these periods, as shown in Figure:12.

![Slot-Channel matrix for a network with 5 channels](image)

*Figure:1 2. Slot-Channel matrix for a network with 5 channels [7]*

TSMP adopts channel hopping which make it very reliable to operate in the noisy environment. The exchange of data packet in TSMP devices occur on different radio channels depending on time of transmission [6]. WirelessHART Transport and Application layers are based on TSMP protocol.

3.2 Time Synchronized Channel Hopping

Dust Company group have developed Time Synchronization Channel Hopping. It has two main properties Channel Hopping and Reservation.
The purpose of Channel Hopping is that reducing the interfaces of narrow-band and increasing reliability. TSCH timing is divided into slots, an Absolute Slot Number (ASN) is incremented at every time slot and shared by all nodes in the network. N slots form a slotframe, this slotframe repeats over time. The frequency to be used at every time slot is calculated by using the formula below:

\[
\text{Frequency} = (\text{ASN} + \text{ChannelOffset}) \mod 16
\]

![Figure: 13. Schedule for 4 nodes A, B, C and D mesh network, slotframe is 10 slots [8]](image)

ChannelOffset is a number between 0 and 15 which is assigned to each slot during reservation. During the reservation one slot of slotframe for node A may reserved to send data to B at a given ChannelOffset. A can thus send data to B at frequency calculated from formula above. ASN is incremented every time and never repeated so we ensure that packet data at different frequency [8].

The second property of TSCH is Reservation, which is a mechanism to avoid collision scheduling. Each cell in the slotframe has the following probability:

I. Not used.
II. DATA cells.
III. RES cells.
IV. ADV cell.

A DATA cell is used to send data from one node to another. RES cells is used to exchange DATA cell information [slot, ChannelOffset], Each DATA cell has RES cell. ADV cell is used for receiving and transmitting advertisement messages to allow to new device join a network.

Our prototype in this project is based on TSCH protocol open source code. We use this code for two reasons, it is free and has many common properties with WirelessHART such as channel hopping and mesh protocol.
3.2.1 Data-Link packets of TSCH

The Data-Link Packet of TSCH is based on cc2420 message-t structure which is included in TSCH folder under the path TSCH>stack>cc2420>cc2420.h. The cc2420 message-t consists of three main fields as shown in figure: 14. They are Header, Data and Metadata as shown in figure.

The Header field is included 15 bytes length and consist 8 sub-fields, they are defined in cc2420.h file as shown in figure: 15. The first fields is length (1 byte), which determines the total length of the packet. The fcf field (2 bytes) is defined the values of the Frame Control Field.

The dsn field (1 byte) is defined the values of the Data Sequence Number. Both destpan (2 bytes) and dest (2 bytes) contain the receiver of this packet, while the src (2 bytes) field contains the source of sender of this packet. The field type (1 byte) is payload of MAC layer and indicates the AM (ActiveMessage) identifier of a TinyOS packet [13]. The timestamp field (4 bytes) indicates the maintain time of this packet.

The type field is used to specify the type of TSCH packets, AM_TSCH_ADV, AM_TSCH_RES, AM_TSCH_DATA and AM_TSCH_ACK.

Figure: 14. cc2420 message-t

Figure: 15. cc2420 header structure of cc2420 message-t
The Data field of message-t is defaulted at 28 bytes length in the cc2420.h. The TSCH packets type is based on the content of Data field, which are AM_TSCH_ADV, AM_TSCH_RES, AM_TSCH_DATA and AM_TSCH_ACK. They are defined in TSCH.h file.

The AM_TSCH_ADV (advertise) packet is maintained by occupying Data field with height, Slot Offset, Channel Offset and Absolute Slot Number parameters. The Data field is constructed as AM_TSCH_ADV packet in the TSCH.h file as shown in figure: 16.

![Figure: 16.Advertise TSCH packet structure](image)

The purpose of AM_TSCH_ADV packet is invited new device to join with the network.

The AM_TSCH_DATA packet is maintained by occupying the Data field with value variable (4 bytes length) or any either variable that we intend store the data. The Data field is constructed as AM_TSCH_DATA packet in the TSCH.h file as shown in figure: 17. The purpose of this packet is transit the device's data to another device in the network.

![Figure: 17. Data TSCH packet structure](image)
The AM_TSCH_RES packet is maintained by occupying the Data field with three parameters, request, slot offset and channel offset. The Data field is constructed as AM_TSCH_RES packet in the TSCH.h file as shown in figure: 18. The purpose of this packet is reserved cell (link) to transmit/receive data.

```c
typedef nx_struct res_format_t {
    nxle_uint8_t request;
    nxle_uint16_t slotOffset;
    nxle_uint8_t channelOffset;
} res_format_t;
```

**Figure: 18. Reservation TSCH packet structure**

The AM_TSCH_ACK packet is maintained by nothing occupying the Data field. The purpose of this packet is confirming the data packets; non-ack packets are received to the intended destination.

The summarization of all TSCH packet types and the detail of message-t’s fields are illustrated in figure: 19.

**Figure: 19. The TSCH packets type**
3.2.2 Neighbors Table of TSCH

The Neighbors Table of TSCH consists of 8 entries as shown in table 1.

<table>
<thead>
<tr>
<th>Used</th>
<th>Neighbor ID</th>
<th>Height</th>
<th>Link quality</th>
<th>Confidence</th>
<th>numSent</th>
<th>numSentOK</th>
<th>LastUsedTimeStamp</th>
</tr>
</thead>
</table>

Table 1. Neighbor Table Entries of TSCH

The Used entry is used to check whether the new neighbor was in list Neighbor Table or not. Neighbor ID is the node address. The numSent is indicated how many packets is sent to this neighbor. The numSentOK is indicated how many ack packets is received from this neighbor. LastUsedTimeStamp is indicated the last time communicated with this neighbor.

3.3 Overview on TSCH stack

We will attempt to take an overview on Time Synchronization Channel Hopping source code and its components since our WirelessHART prototype implementation based on this code. Figure 20. Illustrate the components of TSCH and its interfaces [8]. This sub-section should be reading with the source code which is available [8].

The evaluation of each component is depending on private component, for example the AdvertiseC component is consist of AdvertiseP.nc file pulse AdvertiseC.nc file, the evaluation is depending on AdvertiseP.nc file since this file is containing the functions and tasks, so the reading of this component it should be with AdvertiseP.nc file, same recommendation when evaluation another components.

3.3.1 TSCHAppC COMPONENT (TSCHAppC.nc)

The main purpose of this component is that connect all TSCH components together to working as one unit but each of component has different functionality.

3.3.2 TSCH Component (TSCHC)

The main purpose of this component is that Receive/Transmit AM-TSCH-DATA packet type from/to another device. The receiving of message is by using Receive interface which is provided by the MultiplexC. The transmitting of message is by
using sendDATAMessage () task which is posted inside the block of the Notify interface block. In the figure, we do not see clearly TSCHC, so we use TSCHC instead of TSCHTest.

Inside the Notify interface block we see that all nodes that have not id=1 will be working as Transmitters and the node which has id=1 should work as Receiver. The transmitting of the message is beginning when pressing on the “user” button of telosB mote.

The reservation interfaces can be ignored or canceled since in WirelessHART there is no reservation mechanism and instead of that the device makes link scheduling operation.

The printDebug() task is responsible for print all tables of other components that connected with TSCHC like, GlobalTimeC, TSCHQueueC etc., since the TSCHC use the Debug Print interface and other components provide it, this allow to call print() function of the Debug Print interface at any place of the TSCHC(TSCHP.nc).

### 3.3.3 Reservation Component (ReservationC)

The main purpose of this component is reserved one of the slot in a superframe of a node for sending/receiving Data to/from another node. The second is that put AM-TSCH-RES packet in queue to TSCHQueueC.

The registerRES () function is used to initial ongoingRES table, which is contain many information about reservation cell such as slot offset, channel offset, type of cell and etc. The ongoingRES table is a queue/buffer for applying the reserving cell, when a message is processing and another message comes, a queue/buffer is needed to store the messages to process them one by one.

The NUMONGOINGRES variable is determining the size of queue/buffer. The NUMONGOINGRES is default at 3 which means 3 request messages can apply reserve cell at the same time.

The processOngoingRES () task is updated some entries of ongoingRES table and AM-TSCH-RES packet field (see sub-section), such as slot offset, channel offset depending on to-do possibility entry.

The Receive interface is used to receive the AM-TSCH-RES packet from another device. After receiving the AM-TSCH-RES packet the registerRES () function
will be called and depending on the request field of AM-TSCH-RES packet the ongoingRES table could be updated.

### 3.3.4 CellUsage Component (CellUsage C)

The purpose of this component is that classifying the type cells in a superframe of a device and determining the information that related with it. Each device has different type of cells, CELLTYPE-OFF, CELLTYPE-RXDATA, CELLTYPE-TXDATA, CELLTYPE-RXRES, CELLTYPE-TXRES, CELLTYPE-ADV and CELLTYPE-RESERVERED.

The CellUsageGet interface is used to get number of cells in a superframe of device, slot offset, channel offset, cell type and neighbor for this device that related this cell. Each cell in a superframe has cell table which contain additional information and statics that related with this cell (see sub-section).

The CellUsageSet interface and helper functions are used to fill cell table. The CellStats interface and cellStatsUpdate () task are used to update the cell table.

### 3.3.5 KeepAlive Component (KeepAliveC)

The purpose of this component is that generating and send KeepAive packet to another device when a device lost synchronization with another device.

The sendKeepAlive (); task is responsible for generating KeepAive packet which has the length field =0 since the payload of it is equal to zero. This task will be posted inside the block of the loosingSynch () of the interface GlobalSynch.

### 3.3.6 GlobalTime Component (GlobalTimeC)

The purpose of this component is that get the local and global time for the device and it is responsible for synchronization. We can use both global and local time to know when the any type of packet is received by using the functions of the GlobalTime and GlobalSynch interfaces.

In the Init interface we see all nodes take synchronization from node id=1, when this happen the LED number 2 will become on.

Inside subreceive interface when AM-TSCH-ADV packet type is received from another node except node id=1 the LED number 2 will become on indicating the neighbor is detected.
3.3.7 Mutiplex Component (MultiplexC)

The purpose of this component is received all types of packets and classified to deliver it to a specific component.

Inside ReceiveAll interface the different of packet are received as follow:

- When the AM-TSCH-ADV packet is received the two interfaces will be signaled, IndicateRx and ReceiveADV. IndicateRx will be get source (src), time stamp and receive signal strength indicator (rssi) of AM-TSCH-ADV packet (see sub-section). ReceiveADV will be delivered the AM-TSCH-ADV packet to NeighborC (Component).

- When the AM-TSCH-RES packet is received the two interfaces will be signaled, IndicateRx and ReceiveRES. IndicateRx will be get source (src), time stamp and receive signal strength indicator (rssi) of AM-TSCH-RES packet. ReceiveRES will be delivered the AM-TSCH-RES packet to ReservationC (Component).

- When the AM-TSCH-DATA packet is received the two interfaces will be signaled, IndicateRx and ReceiveDATA. IndicateRx will be get source (src), time stamp and receive signal strength indicator (rssi) of AM-TSCH-DATA packet. ReceiveDATA will be delivered the AM-TSCH-DATA packet to TSCHC (Component).

3.3.8 TDMASlot Component (TDMASlotC)

This component is not connected in TSCHAppC, so this component is not played any role. Instead of using it, they use cc2420 TdmaSlotC to send/receive a different type of packets via radio.

3.3.9 TSCHQueue Component (TSCHQueueC)

The purpose of this component is holding the data of an Active Messages as well as the address to be executed (transmitted). The first task of this component is putting the message will be sent in queue then send it one by one.
There are four types of packet messages that will be sent, AM-TSCH-DATA, AM-TSCH-RES, AM-TSCH-DATA, AM-TSCH-ADV and KeepAlive packet which is the same as AM-TSCH-DATA packet but it has not payload (see sub-section).

The interface SendDATA will be used to put the AM-TSCH-DATA packet in queue by calling the putInQueue () function.

The interface SendKA will be used to put the KeepAlive packet in queue by calling the putInQueue () function. The interface SendRES will be used to put the AM-TSCH-RES packet in queue by calling the putInQueue () function.

The interface SendADV will be used to put the AM-TSCH-ADV packet in queue by calling the putInQueue () function. In order to send the packets above they used for each type of packet a specific task to do that. The sendDoneTsakDATA () is responsible for send AM-TSCH-DATA packet. This task will post in informRequster () function. The sendDoneTsakKA () is responsible for send KeepAlive packet. This task will post in informRequster () function.

The sendDoneTsakRES () is responsible for send AM-TSCH-RES packet. This task will post in informRequster () function. The sendDoneTsakADV () is responsible for send AM-TSCH-ADV packet. This task will post in informRequster () function. The Dequeue interface will be used to execute all tasks above by calling informRequster () function which is included (posted) all tasks above.

### 3.3.10 Advertise Component (AdvertiseC)

The purpose of this component is built and broadcast AM-TSCH-ADV packet to any device in the network.

The buildAndSendAdv () task is used to maintain the AM-TSCH-ADV packet (see sub-section). The first step is filling the data field of this packet which are, slot offset, channel offset and height, then fill the some field of header which are, length, dest and type. Dest field should be Tos-BCAST-ADDR which means that the destination address is broadcast. The final step is send AM-TSCH-ADV packet to TSCHQueueC by using SimpleSend interface. The buildAndSendAdv () task is posted when Timer expire.

### 3.3.11 Neighbors Component (NeighborsC)

The purpose of this component is receiving the AM-TSCH-ADV packet from neighbor device through MultiplexC. The duty of this component is filled and
updates Neighbor Table Entries after receiving the AM-TSCH-ADV packet. Finally this component is updated some fields of different packet type (AM-TSCH-ADV, AM-TSCH-DATA, AM-TSCH-RES) that will be sent to TSCHQueueC such as destination in header of packet, ack and timestamp of metadata packet (see sub-section).

The SendDATA, SendADV, SendRES and SendKA interfaces are used to update and send AM-TSCH-DATA, AM-TSCH-ADV, AM-TSCH-RES, AM-TSCH-ADV (KeepAlive) packets respectively to TSCHQueueC to put them in queue. These interfaces is used updateStatsAtTx () function to update destination in header of a packet, ack and timestamp of a metadata packet.

The NeighborGet interface will be used to fill Neighbor Table Entries by calling different functions of this interface.

The ReceiveADV interface will be used to receive AM-TSCH-ADV packet from another neighbor device, when that happen the new neighbor will be add to list of the neighbor device and Neighbor Table entries of this new neighbor will be filled.

### 3.3.12 Forwarding Component (ForwardingC)

The purpose of this component is get AM-TSCH-DATA packet from TSCHC then forwards it to TSCHQueueC.

The SentFromUpper interface is used to get AM-TSCH-DATA packet from TSCHC, then forwards it to TSCHQueueC by using SentToLower interface which is calling signalSendDone task to do that.

### 3.3.13 ActiveMessageAddress Component (ActiveMessageAddressC)

The purpose of this component is stored the node’s active message address and group ID.
4 Improvement of communication tables for WirelessHART devices

There are many tasks that MAC protocol (Medium Access Control) supplied:

- Send the messages that received from Network Layer and listen to the messages from neighbors through Physical Layer.
- Slot time synchronization.
- Identify the device that should access the medium.

The Medium Access Control (MAC) sub-layer is responsible for sending the Data Link Protocol Data Unit Packet (DLPDU), that maintain in this sub-layer, across the link. To ensure that, MAC should achieve many operations, Communication Tables, State
Machine, Link Scheduling and Timer, we will explain in detail just Communication Tables and Link scheduling since they are related with our project. Figure 21. Illustrate MAC layer architecture.

4.1 Communication Tables

Each device maintains a collection of tables in the data link layer that control the communications performed by the device and collect statics on those communications, Figure 22. Illustrate these tables [10].

Figure: 21. WirelessHART Data Link Layer Architecture

Figure: 22. Communication Tables [10].

The tables controlling communication activities include:
4.1.1 Superframes Table

The purpose of this table is to configure the communication between the device and its neighbors. The superframe table consists of four fields, superframe ID, NumSlots indicating the size of superframe, Active flag indicating whether this superframe active or not and Links which contain the list of links in this superframe, see Table 2 [10].

<table>
<thead>
<tr>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsigned-8 SuperframeID</td>
<td>Unique identifier of the Superframe</td>
</tr>
<tr>
<td>Unsigned-16 Numslots</td>
<td>Number of slots in the Superframes</td>
</tr>
<tr>
<td>Bits-1 ActiveFlag</td>
<td>Flag indicating</td>
</tr>
<tr>
<td>Links [ ]</td>
<td>List of links in this superframe</td>
</tr>
</tbody>
</table>

**Table2. Superframe contents**

The Network Manager is responsible to generate and supply Superframes in the network, before generating the Superframes it derived suitable routes for network. Then Network Manager generating overall Superframe to network, after that it sub-scheduling the overall Superframes, and then each superframe will be delivered to each device, as shown in Figure 9. The superframe is fixed slots time, but it different in contention in order to satisfy the requirement of communication for each device.

As we see in the Figure 9. The network manager is determined the type of each cell (slot), which is corresponding the link Type (normal, broadcast, join, discovery) and the direction of each link which is either Transmit link (Tx) or Receive link (Rx) in advance in order make it easier for Link scheduling operation to do its task.

The Superframe table is interface with the Link table by LinkId as shown in Figure 23.

**Figure: 23. Overall Superframes for Network contain two devices with four time slot for each superframe**
4.1.2 Link Table

The purpose of this table is to configure the communication between the device and its neighbors. There are more than one link within a Superframe can be configured to specify the communication with a specific neighbor or broadcast (an unspecified) group of neighbors [10].

The link table consists of many fields, which are filled by Network Manager. See table 3.

<table>
<thead>
<tr>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkId</td>
<td>Identifier of the link</td>
</tr>
<tr>
<td>RefNeighborId</td>
<td>Reference to Neighbour table entry</td>
</tr>
<tr>
<td>LinkType</td>
<td>Type of link</td>
</tr>
<tr>
<td>TxLink</td>
<td>Direction of the link is Transmission</td>
</tr>
<tr>
<td>RxLink</td>
<td>Direction of the link is Receiving</td>
</tr>
<tr>
<td>SharedLink</td>
<td>Shared link indicator</td>
</tr>
<tr>
<td>Slot</td>
<td>Slot number</td>
</tr>
<tr>
<td>ChannelOffset</td>
<td>Frequency hopping</td>
</tr>
</tbody>
</table>

*Table: 3. Link table*

The LinkId is the unique identifier for the link; it is supplies by the Network Manager within a Superframe and also all entries in the table. The RefNeighborId is the reference to Neighbor table entry, i.e. a reference to a neighbor that is allowed to communicate with the device, it represents the interface entry with neighbor table. The LinkType indicates the type of link (normal, broadcast, join or discovery link). The TxLink when it set this mean the link as transmitter link. The RxLink when it set this mean the link as receiver link. The SharedLink when it set, indicates the link may is shared by multiple devices.

The Slot is the interface from the Superframe to the link table, which represent the slot within the Superframe, it represents in which slot number within specific a superframe the device will be communicated with neighbor. The ChannelOffset is the channel
offset of the frequency hopping, it represents in which channel number the device will be communicated with neighbor.

### 4.1.3 Neighbors Table

The purpose of Neighbor Table is that to help the route table to select right route through Graph Table. The Neighbor Table contains all devices that the device communicate with sharing link or the device may be communicated have been overheard. Since each link has a reference to one neighbor i.e. each link has one Neighbor Table Entry. The Neighbor Table Entry involves the statics and properties that regard the neighbor, as shown in table 4.

<table>
<thead>
<tr>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NeighbourUniqueID</td>
<td>Long address of the neighbour device</td>
</tr>
<tr>
<td>NeighbourNickname</td>
<td>Short address of the neighbour device</td>
</tr>
<tr>
<td>JoinPriority</td>
<td>Depended on the DLPDU content</td>
</tr>
<tr>
<td>TimesourceFlag</td>
<td>Flag indication</td>
</tr>
<tr>
<td>Status</td>
<td>Status information regarding this neighbour</td>
</tr>
<tr>
<td>BOExp</td>
<td>Back off exponent for shared link</td>
</tr>
<tr>
<td>BOCntr</td>
<td>Back off countdown for shared link</td>
</tr>
<tr>
<td>Time</td>
<td>Last time communicated with this neighbour</td>
</tr>
<tr>
<td>LastTimeCommunicated</td>
<td></td>
</tr>
<tr>
<td>Time PathFailureTimer</td>
<td>Cyclic path failure timer</td>
</tr>
<tr>
<td>AvgRSL</td>
<td>Average received signal level for packets received from neighbour</td>
</tr>
<tr>
<td>packetsTransmitted</td>
<td>Number of packets transmitted to the neighbour</td>
</tr>
<tr>
<td>MissedAckPackets</td>
<td>Number of packets for which an expected ACK was not received</td>
</tr>
<tr>
<td>PacketsReceived</td>
<td>Number of packets received from neighbour</td>
</tr>
<tr>
<td>BroadcastsReceived</td>
<td>Number of broadcast packets received from neighbour</td>
</tr>
</tbody>
</table>

*Table 4. Neighbor table entry*

TimesourceFlag entry is indicating if the device should take time synchronization from this neighbour. Time PathFailureTimer entry resets to pathfailinterval after each successful communication. Neighbor Table has two interfaces, one with Link Table which is NeighbourID as input from Neighbor Table to Link Table, and the other with the Graph Table which is destNickname as output from Neighbor Table to Graph Table [10].
Neighbor Table is different from other tables since it is not filled by Network Manager as others tables but it is filled when new neighbor’s device is detected (when device receive advertise massage).

4.1.4 Graph Table
The Graph Table maintains by Network Manager and delivers to store in Network Layer device’s and stamp it on Network Layer Protocol Data Unit packet (NPDU). NPDU packet coming from Network Layer to MAC layer can be used GraphId of Graph table to guide the DLPDU packet to final destination. Graphs are used to route messages from their source to their destination. A graph is a directed list of paths that connect two devices within the network. See table 5.

<table>
<thead>
<tr>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>graphId</td>
<td>Unique Graph Id</td>
</tr>
<tr>
<td>destUniqueID</td>
<td>Destination node’s address</td>
</tr>
<tr>
<td>destNickname</td>
<td>The short address of the neighbour</td>
</tr>
<tr>
<td>RefNeighbor</td>
<td>List of references to neighbors</td>
</tr>
</tbody>
</table>

*Table: 5. Graph Table*

The RefNeighbor represents the list of references to neighbors that are the next hop toward the destination. It identifies those devices that allow legally Data-Link destinations for the packet’s next-hop toward its final destination.

The Graph Table has two interfaces, one with the Neighbor Table which is destNickname as output from the Neighbor Table to Graph Table, the another interface is Graph ID with NPDU packet as output from the Graph Table to the NPDU packet [10].

4.2 Link Scheduler
The main functionality of link scheduler is determined the next slot to be serviced (receiving or transmitting slot) based on the communication schedule in superframe and link table. The scheduler is complicated since it is depended on many factors such as link changes, transaction priority, and
enabling and disabling of superframe. Every event that can affect link scheduling must result in the link schedule being re-assessed [12].

In case of transmitting packet link scheduling includes evaluating the packet come from Network Layer to MAC Layer which is waiting for propagation and determining the Absolute Slot Number (ASN) that should used to send the packet. In case of receiving packet all received links within a superframe should assessed to determine the first Absolute Slot Number (ASN) that can be used in attempting to receive the packet.

Regardless transmitting or receiving packet the first Absolute Slot Number (ASN) should be scheduled for servicing (transmitting or receiving slot).

### 4.2.1 Servicing Transmit Links

The packets received from the Network Layer, Network Protocol Data Unit packet (NPDU) ,are scheduled for a slot based on the destination address type (NPDU), see Figure 24. The type of destination address determines the type of link type associated with the ASN.

<table>
<thead>
<tr>
<th>control</th>
<th>TTL</th>
<th>Graph</th>
<th>Destination</th>
<th>Source</th>
<th>Expanded Routing</th>
<th>Payload from Transport layer</th>
</tr>
</thead>
</table>

**Figure: 24. WirelessHART NPDU structure**

For each type of destination address the set of transmit links must be determined [10].

- If Destination address type Graph: For a graph-routed destination, the set of links involve all transmit links for all neighbors in the graph.
- If Destination address type broadcast: the set of links shall be all broadcast links for the designated Superframe ID.
- If Destination address type Neighbor: when a single neighbor is specified as the destination, the set of links shall be all transmit links to that single neighbor.
- If Destination address type Proxy: then set of links shall be all transmit links of type Join.

After setting of suitable transmitting links the packet that coming from Network Layer should be determined, the set of associated slots of
transmitting links should be determined also. The intersection between the slots of transmitting links and waiting packet for propagation determines the next slot to be scheduled for transmitting a packet [10].

4.2.2 Servicing Receive Links

All receive links within the active superframe should be scheduled. All receive link should be serviced. After generating the order list links, the earliest slot is selected and the link within that slot with lowest superframe ID number becomes the member for servicing because it already oldest slot within the oldest superframe ID.

4.3 Timer

The main purpose of timer in WirelessHART is providing accurate timing to ensure the correct operating of the system.

4.4 Message Handling Module

The purpose of Message Handling Module is to buffer the packets from physical layer and network layer.

4.5 State Machine

State machine control the receiving and propagation of packets through the MAC sub-layer. It gives the order to both, Timer to synchronization and link scheduler to start scheduling operation. The state machine consist of TDMA state machine, XMIT and RECV engines. The TDMA state machine executes the transaction in a slot and adjusting timer clock. The XMIT and RECV are responsible for sending and receiving a packet [12].

4.6 Data-Link packets (DLPDUs)

The format of the Data-Link packet (DLPDU) of Wireless HART consists of the following field [10]:

- A single byte set to ox41.
• A 1-byte is address specifier.
• The 1-byte Sequence Number.
• The 2-byte Network ID.
• Destination and Source Addresses either of which can be 2 or 8-bytes long.
• A 1-byte DLPDU specifier.
• The DLL payload.
• A 4-byte keyed Message Integrity Code (MIC), and
• A 2-byte ITU-T CRC16.
Figure 25 illustrates the basic PhPDU and DLPDU structure.

The overall packet length is 127 bytes.
There are four types of DLPDU Wireless HART. The least significant 3 bits of DLPDU Specifier indicate the type of DLPDU being communicated and the purpose of the (optional) DLL payload. There are five DLPDU types [10]:

• DATA DLPDU: the Source Address and Destination Address field of this DLPDU type is regarded the Network Layer of both source and destination devices. The packet payload generated from the Network Layer of hope’s source device and is passed to the destination device’s Network Layer. This type of packet is containing network and device data in transit to their final destination.
• Keep-Alive DLPDU: It is used for network maintenance, for network time synchronization, for assess communication with a neighbor (confirm connectivity) and for neighbor discovery. This packet’s payload is empty (equal to zero). This packet is not broadcast since it would be sent to specific neighbor.

• Advertise DLPDU: It is used to invite new devices into the network. When a device wishes to join a network, it listens to Advertise DLPDU packet and then uses the information in this packet to synchronize with the network and initiate the join process. The Advertise DLPDU packets contain network information, Absolute Slot Number (ASN), the join control information and the security levels. The Advertise DLPDU Destination Address field should be broadcast.

• Disconnect DLPDU: It is used to inform the neighboring devices that the device is leaving the network. The payload of this packet is empty.

• ACK DLPDU: It is transmitted by a device in response to receipt of non-broadcast and non-ACK DLPDU.

5. Implementation

The implementation of WirelessHART stack of this thesis is in tynos 2.1.0. In this section the implementation of WirelessHART stack is described. An overview of architecture will be taken. Finally the tasks of this thesis are explained.

Since the WirelessHART is released recently, the closed open source is very limited. Thomas W. [8] has developed TSCH prototype which is very close to WirelessHART protocol. Our tasks implementation is based on TSCH source code.

The implementation and testing our prototype is implemented on TelosB [14] mote since this type of mote is supported Channel Hopping.

5.1 Architecture

The TSCH stack components is corresponding 3-Lyers of WirelessHART OSI 7-layer Model (see subsection 2.3). They are physical Layer (PHY), Multiple Access Control Layer (MAC) and Application Layer as shown in figure: 26.
The Application Layer here is playing limited role, it just defines device data type. The MAC Layer is playing major role to implement our tasks, communications table, Link Scheduling, send/receive different packets type.

5.2 Network Manager Device

The implementation of Network Manager of WirelessHART is essentially since the some tasks implementation of this thesis is based on it. The Network Manager is responsible for create superframe table and link table. After creating of superframe table and link table the Network Manager will be sent them to a specific device. We use Network Manager Device as Device sometimes.

The first step is how to specify the Network Manger based on TSCH code? Depending on the analyzing of TSCH components (see subsection 3.3) just node address id=1, is receiver node for data packets and the other node's id address is sending data packets. So the Network Manager is every node has not address id =1.

The specifying of node's id address is during the installation and compiling the TSCH code on Telosb mote by using the following commands:

```
make telosb install,2,bsl, /dev/ttyUSBx
```

The number 2 is indicated node’s id address and here the device is represented transmitter of data packet and we use it as Network Manager.

5.3 Device

The Device in our implementation is also essentially to implement some of our tasks. The device in our work is responsible for receiving both superframe table and link table from the Network Manager. After receiving the superframe table and link table the device starting link scheduling process and maintain Neighbor Table. The specifying of a node as device by type the following command during installation and compiling the TSCH code on Telosb mote:
The number 1 in the above command is indicated that the device is working as receiver of data packet.

5.4 Superframe Table and Link Table

The Superframe Table and Link Table are maintain in Network Manager Device that is built in subsection 5.2. The initial step is define both Superframe Table and Link Table Entries in TSCH.h file as shown in figure: 27 and figure: 28.

We choose the TSCH component to perform Superframe Table and Link Table task (see subsection 3.3.2).

```c
typedef nx_struct linktable_t {
    nxle_uint8_t neighbour;
    nx_bool Txlink;
    nx_bool Rxlink;
    nx_bool sharedlink;
    nxle_uint8_t slot;
    nxle_uint8_t channelofset;
    nxle_uint8_t linktype;
} linktable_t;
```

**Figure: 27. Structure of Link Table**

```c
typedef nx_struct superframe_t {
    nx_bool activeflag;
    nxle_uint8_t sfld;
    nxle_uint16_t numslots;
    linktable_t* links[4];
} superframe_t;
```

**Figure: 28. Structure of Superframe Table**

The second step is selecting appropriate TSCH packet type to send both Superframe Table and Link Table to the Device node (see figure: 29) that built in subsection 5.3 since if we define new TSCH packet type we define it in many components of TSCH stack and it take many time to do that.
We select AM-TSCH-DATA packet (see subsection 3.2.1) type to send Superframe Table and Link Table, so Data field of message-t (see subsection 3.2.1), will occupy by Superframe and Link tables instead of value as shown in figure:30.

The third step is defaulted values for Superframe Table and Link Table entries. The fourth step is sent the AM-TSCH-DATA which is contain Superframe Table and Link Table entries to the Device by using the sendDATAMessage() task (see subsection 5.2).

### 5.5 Link Scheduling

The Link Scheduling process is implemented on Device node. The Link Scheduling starting after receives Superframe and Link tables from Network Manager Device. The Link Scheduling depends on NEK packet that is coming from Network Layer in case of transmitting any type of packets to another Device in the network (see subsections 4.2.1 and 4.2.2).
The first step is maintaining WirelessHART Network packet, so we construct the Network packet as shown in figure: 31 and define destinationAddress variable as shown in figure: 32 in the TSCH.H of Device file.

We choose the TSCH component to perform Link Scheduling process (see subsection 5.2).

```c
typedef nx_struct network_header {
    nxle_uint8_t control;
    nxle_uint8_t ttl;//Time To Live hop counter
    nxle_uint16_t ASNShort;
    nxle_uint16_t graphID;
    nxle_uint16_t destAddress;
    nxle_uint16_t srcAddress;
} network_header_t;
```

**Figure: 31. Construction of WirelessHART Network**

```c
enum {
    DEST_NEIGHBOR,
    DEST_GRAPH,
    DEST_PROXY,
    DEST_BROADCAST
};
```

**Figure: 32. Address possibility of destination of WirelessHART**

After defining of WirelessHART Network packet we define a startlinkscheduling () task to perform the Link Scheduling, this task is posted inside of Receive interface function.

The flow chart of figure: 33 is summarize Link Scheduling process that is performed by the startlinkscheduling () task.
Figure: 33 Link Scheduling process
Figure: 33 Link Scheduling process
5.6 Neighbor Table

We use Device that is built in subsection 6.3 to implement the WirelessHART Neighbor Table. By using the TSCH component (see subsection 3.2.2) of the Device to implement WirelessHART Neighbor Table.

First step is that extending the TSCH Neighbor Table Entries (see subsection 3.2.2) in neighbors.h file to including WirelessHART Neighbor Table Entries (see subsection 4.1.3) as shown in figure: 34.

```c
typedef struct neighborEntry_t {
    bool used; uint16_t id; uint8_t height;
    uint8_t linkQuality; uint8_t confidence; uint32_t numSent;
    uint32_t numSentOK; uint32_t lastHeardTimestamp;
    // WirelessHART Neighbor Entries
    uint16_t nickname; uint8_t joinPriority; bool timesourceFlag;
    bool status;//changed to bool
    uint8_t boexp; uint8_t bocnt; uint16_t avgrs; uint32_t pktx;
    uint16_t misackpkt; uint16_t pktrx; uint16_t broadcastpktrx; uint8_t timepathf;
    uint32_t lastTimecommunicated; uint8_t type;//just for join priority
} neighborEntry_t;
```

Figure: 34. TSCH neighbor. h file including WirelessHART Neighbor Table Entries
The common entries between TSCH Neighbor Table and WirelessHART Neighbor Table are three entries. The id, lastHeardTimeStamp and numSent of TSCH Neighbor Table are corresponding Neighbor ID, lastTimecommunicated and packetsTransmitted entries of WirelessHART Neighbor Table.

These three entries of TSCH Neighbor Table are getting in the NeighborsC (see subsection 3.3.10), so in this case we can call the interfaces whose responsible to generate these three entries in the TSCHC of the Device as shown in figure: 35. and figure:36.

```c
neighbors[i].lastTimecommunicated = call NeighourGet getLastHeard(neighbors[i].id);
neighbors[i].pkttx = call NeighourGet getNumSent(neighbors[i].id);
```

*Figure: 35. Getting lastTimecommunicated and number of Packets transmitted entries of WirelessHART Neighbor Table*

```c
neighbors[i].id = ((cc2420_header_t*)msg->header)->src;
```

*Figure: 36. Getting Neighbor ID entry of WirelessHART Neighbor Table*

The Neighbor ID will be getting inside Receive interface function of TSCHC as shown in figure: 36. The lastTimecommunicated and packets transmitted (pkttx) entries are getting by calling their interfaces functions as shown in figure 30, inside the printable () task that generated in TSCHC of the Device to print WirelessHART Neighbor Table.

The rest WirelessHART Neighbor Table Entries will be getting by generating new task inside TSCHC of the Device. The joinpriority entry will be getting by using Joinpriorityfunction () task which is responsible for getting the join priority as shown in figure: 37. This task will be posted inside Receive interface function of the Device.

The average receive signal level (avgrsl) entry will be getting by using the IndicateRx interface, since it not using in the TSCHC, the declaration and connection of this interface should be performed. Then we can implement IndicateRx interface function in TSCHC of the Device. Inside the IndicateRx interface function block the avgrsl will be getting by using the following formula:

\[ \text{avgrsl} = \frac{\text{rssi}}{\text{avgrslDamp}} \]
Where

- \( \text{rssi} \) is the receive signal strength indicator.
- \( \text{avgrslDamp} \) is the damp factor and defaulted at 64.

The Time Source Flag (\( \text{timesourceflag} \)) entry is indicating if device should take time synchronization from this neighbor. In our case all nodes should take synchronization from node id=1, our Device has id=1 and it receive advertise packet from Network Manager Device, so it does not take time synchronization, implementing this process is inside the Receive interface function of TSCHC of the Device as shown in figure :35.

The packet receive (\( \text{pktrx} \)) entry of Neighbor Table is finding by implement the process inside Receive interface function block of Device's TSCHC as shown in figure :36.

Figure: 37. Getting join priority entry of WirelessHART Neighbor Table
The misacknowledgement packet (misackpkt) is found by using the following process inside the TSCHC of the Device as shown in figure: 37.

Figure: 35. Finding timesourceflag entry
Figure: 36. The process to find packet Rx entry
The finding beacon off exponent (BoExp) and beacon off counter (BoCntr) entries are regarding the shared link, since we have not shared link in our case so both of them are equal to zero. The following figure: 38. is summarizing how to get them. This process is implemented by using the boexpboctr() task which is posted inside the Receive interface of Device's TSCHC as shown in figure.

Figure: 37. Finding the misackpkt entry of Neighbor Table

The finding beacon off exponent (BoExp) and beacon off counter (BoCntr) entries are regarding the shared link, since we have not shared link in our case so both of them are equal to zero. The following figure: 38. is summarizing how to get them. This process is implemented by using the boexpboctr() task which is posted inside the Receive interface of Device's TSCHC as shown in figure.
Figure: 36. Finding BoExp and BoCntr entries
The status and time path failure timer entries are given by using the process as shown in figure: 39. We do not need to use new task to implement this process we just use the same timer that used in TSCHC of Device to get the status entries. We changed data type status entry from integer to Boolean since it indicates either successful communication or not as shown in figure.

Figure: 39. Getting status and time path failure timer entries of Neighbor Table
6. Results

6.1 Network Manager Device

After compiling and downloading Network Manager Device’s code (see Appendix) on a mote, it should take synchronization from Device before sending the superframe and link table message to Device. The message below is appearing on Network Manager Device’s simulation screen, which indicating Network Manager Device is not getting synchronization from Device yet:

| TSCH ReservationUpdate.addSlot to 1 returned=1 |
| ReservationP.nc DEBUG node 1 not reachable (yet?) |

When the message below is appearing on Network Manager Device’s simulation screen, it is indicating Network Manager Device has getting synchronization from Device and it is ready to send superframe and link table:

| TSCH ReservationUpdate.addSlot to 1 returned=0 |
| TSCHP ReservationUpdate.done add-1 neighbor-1 error-0 |
| TSCH ------------ READY -------------------------- |
| TSCH ------------ READY -------------------------- |
| TSCH ------------ READY -------------------------- |
| DEBUG: Not able to start radio on time |
| TSCH ------------ READY -------------------------- |
| TSCH ------------ READY -------------------------- |

6.2 Device and Neighbor table

When Network Manager Device has getting synchronization from Device and it is ready to send superframe and link table, we press on “user” button of Network Manager Device’s mote to send superframe and link table to Device.

When we are compiling and downloading Device's code (see Appendix) on a mote, it maintains Neighbor table after it detects Network Manager Device and the Neighbor table bellow is appearing on simulation screen's Device:
7. Conclusions and Future work

7.1 Conclusions

The first step that we should do is testing TSCH code by using new version of virtual machine and Xubuntu2.1, using old or corrupt version lead to many mistakes and worse the time.

In order to get java support to view print statement on pc screen we should do the following procedure as shown in figure: 40.

The WirelessHART Neighbor Table consist of 14 entries, 3 of them are common with TSCH Neighbor Table. In order to get rest entries we attempt to expand the Neighbor Get interface of NeighborsC by adding new command functions to this library but this way is not corrected.

The TSCH stack is designed in order the mote that has id=1 as receiver of DATA packet and the others mote that has other number id as transmitter.

The removing printDebug() task from Notify interface block of TSCHC component and post it inside the Receive interface function lead to print all table components if the Advertise packet is received.
There are many useful commands which are summarizing in figure: 41.

```
1) compile and install on telosb mote
   make telosb install,1 bsl,/dev/ttyUSB0

2) get print from mote to pc
   java net.tinyos.tools.Print2Client -comm serial0/dev/ttyUSB0:telosb

3) get usb no. and know if the mote connect or not
   motolist
```

**Figure: 41. Useful commands**

The Network Manager of WirelessHART is responsible for generating a superframe for a specific device in the network. The superframe contains different type of links and it is specified either transmit link or receive link in advance. Each link is a function of slot number and number of channel. Every communication tables of WirelessHART except Neighbor Table are created by the Network Manager then delivered to a specific device in the network. The Neighbor Table is maintaining by the device itself when the device is received Advertise packet.

The expand function of any interface is not correct way, so instead of that we rename the interface or add new component to the stack.

### 7.2 Future Work

The TSCH stack is could be used as WirelessHART Network Manager or Device. The using of TSCH stack as Device is required remove ReservationC and CellUsageC from the stack and add new component to do the link scheduling process but the using of TSCH stack as WirelessHART Network manager is required remove ReservationC and modify CellUsageC to specify transmit link or receive link.
The defining any of WirelessHART DLPDUs type is possible by occupying Data field of message-t by the contents of any DLPDUs type of WirelessHART.

References

Appendix

Device TSCMP

```c
#include "TSCMP.h"
#include "Neighbors.h"
#include "UserButton.h"

module TSCMP {
    //general
    provides interface init as softwareInit;
    uses interface boot;
    //from above
    uses interface leftover;
    uses interface Send;
    uses interface Neighbor;
    uses interface Receive;
    uses interface ReservationUpdate;
    uses interface OrdCntr;
    uses interface ActiveMessageAddress;
    uses interface Leds;
    uses interface indicatorR;;

    //debug
    uses interface Random;
    uses interface PrintPackets;
    uses interface NotifyButtonState;
    uses interface Timer as Timer;
    uses interface HeliMap400GeneralIO as Port94;
    uses interface DebugPrint as PrintReservation;
    uses interface DebugPrint as PrintDebug;
    uses interface DebugPrint as PrintNeighbor;
    uses interface DebugPrint as PrintNeighbor;
    uses interface DebugPrint as PrintQueue;
    uses interface DebugPrint as PrintMessage;
    uses interface DebugPrint as PrintStatistics;
    uses interface DebugPrint as PrintCharacteristics;
    uses interface DebugPrint as PrintGlobalTime;
    }

implementation {
    message & queuedPkt[QUEUEDLENGTH];
    bool queuedPkt[QUEUEDLENGTH];
    uint8_t neighbor_key[8];
    uint8_t printDebugCounter = 0;
    uint8_t debugMessageCounter = 0;
    uint8_t reservationCounter = 0;
    bool blinkBool;

    state_t state[7];
    }  //*******************************************************************************/

    //exam superframe
    uint8_t sfic;
    uint8_t numslots;
    uint8_t * slots = {0};
    bool activeFlag;
    //link scheduling
    devicemode_t mode;
    uint8_t linksetlist[4] = {0}; //to store valid slots of links in link
    scheduling
    network_header_t * nhvhs://vel
    neighbors_t neighbors[MAXNUMNEIGHBORS];
```
```c
void getNearestHop();
void checkReservedCells();
void sendDAMAmessage();
void addReservation();
void removeReservation();
void printStats();
void printDebug();
void blinkLeds();
void startLinkScheduling(); // esemextension
void printTable(); // esemextension
void joinPriorityFunction(); // esam
void boxpctr(); // esam
void pathfailstatus(); // esam
void startLinkScheduling() {
    uint8_t i;
    uint8_t slotTxPkt;
    uint8_t slotRxPkt;
    nwkaddr->GestAddress; // DEST_NEIGHBOR; // Testing, give a
default value.

    switch (mode) {
    case radio:
        if (activeFlag==1) {
            switch (nwkhdr->destiAddress) {
                case DEST_NEIGHBOR:
                case DEST_GRAPH:
                    for (i=0; i<numslots; i++) {
                        if (links[i].link==TRUE &
                            links[i].linktype==normal) {
                            linkslot[i]=links[i].slot;
                            slotTxPkt=(call Random.randl6()&linkslot[i]);
                        }
                    }

                //esam post Txpkt task here, attempt later

                //post testingLinkscheResult();//printf delay for test
                break;
                case DEST_PROXY:
                    for (i=0; i<numslots; i++) {
                        if (links[i].link==TRUE &
                            links[i].linktype==join) {
                            linkslot[i]=links[i].slot;
                            slotRxPkt=(call Random.randl6()&linkslot[i]);
                        }
                    }

                //esam post Txpkt task here, attempt later

                //post testingLinkscheResult();//printf delay for test
                break;
                case DEST_BROADCAST:
                    for (i=0; i<numslots; i++) {
                        if (links[i].link==TRUE &
```
links[1].linktype==broadcast) {
    linksslot[1]=links[1].slot);
    slotTmpt=[call Random.rand16()&linksslot[1]];
}

//exam post Tpkt task here, attempt later
//post testing(linkscheResult();//printf delay for test
break;
}
}
case listen:
    if(activeflag==1){
        for (i=0;i<numslots;i++) {
            if (links[i].R.xlink==TRUE ) { // links[i].R.xlink==TRUE
                slotTmpt[i]= links[i].slot; //select oldest slot
                break;
            }
        }
    }
}

}

}


task void printTable() {
    uint8_t i;

    printf("TSCHPer. 100 neighbors[]-\n");
    for (i=0;i<MAXNUYNEBNIGHORS;i++) {
        if (neighbors[i].used==TRUE){
            neighbors[i].numSent = call NeighborGet.getNumSent(neighbors[i].id);
            neighbors[i].pktx = call NeighborGet.getNumSent(neighbors[i].id);

            if (call ActiveMessageAddress.unkAddress()==1){//get timesrc flag
                neighbors[i].timesrcflag=FALSE;
            }else{
                neighbors[i].timesrcflag=TRUE;
            }

            printf(" neighbor id=%d",neighbors[i].id); //get id from Receive
            interface function
            printf(" nickname=*",neighbors[i].id); //get nickname
            printf(" PacketsTransmited=",neighbors[i].pktx);
            printf(" TimeLastCommunicated=",neighbors[i].lastTimecommunicated);
            printf(" PacketsReceived=",neighbors[i].timesrcflag);
            printf(" PacketsReceived =\n",neighbors[i].pktx);

            }else{

            }

    }

54
printf(“MissAckDucketsReceived =%lu”,neighbors[i].missAckPkt);
printf(“BroadcastPacketsReceived =%lu”,neighbors[i].broadcastPkt);
printf(“BonExp =%lu”,neighbors[i].boExp);
printf(“BonCnt =%lu”,neighbors[i].boCnt);
printf(“AvgRSL =%lu”,neighbors[i].avgRSL);//From indicateRx

printf(“timepathf =%lu”,neighbors[i].timepathf);
if(neighbors[i].timepathf==0)(printf(“ status = TRUE\n”));
else(printf(“ status = FALSE\n”));
}
}

printfln();
//end printTable

/**********************get join priority task,exam
//task void joinPriorityFunction()
{ uint32 t =0;
  for (t=0;t<MAXNUMNEIGHBORS;t++){
    switch(neighbors[t].type){
    case AM_TYPE_SCH_ADV:
      neighbors[t].joinPriority=2;
      //return neighbors[t].joinPriority;
      break;
    case AM_TYPE_SCH_DIS :
      //neighbors[t].joinPriority=0;
      //return neighbors[t].joinPriority;
      break;
    case AM_TYPE_SCH_DATA :
      neighbors[t].joinPriority=0;
      //return neighbors[t].joinPriority;
      break;
    case AM_TYPE_SCH_ACK :
      neighbors[t].joinPriority=1;
      //return neighbors[t].joinPriority;
      break;
    }
  }
}

//exam get hloop-connector task
//task void hloopconnect()
{ uint32 t =0;
  uint32 j =0;
  message_t * msg;
  for (t=0;t<MAXNUMNEIGHBORS;t++){
    if (neighbors[t].id==((cc2410_header_t)msg->header)->src){
      for (j=0;j<numNodes;j++){
        if (links[j].sharedLink==false) {
          neighbors[i].boExp=0;
          neighbors[i].boCnt=0;
        }
      }
    }
  }
}

};//finished task

//path failure time and status task start
//task void pathFailStatus(){
uint8_t i=0;
uint32_t pathfailinterval=250;
message_t msg;

for (i=0; i<2*MAXNUMNEIGHBORS; i++) {
    if (neighbors[i].id==((cc2420_header_t*)msg->header)->src) {
        call Timer_startPeriodic(pathfailinterval);
    }
}

} // path failure time and status task finished

task void getNextHop() {
    //nextHop = call NeighborSet.minHeight();
    nextHop_exists = TRUE;
    nextHop.neighbor = 1;
    if (!nextHop_exists) {
        printf("\n\TSCH ERROR no minHeight destination\n");
        return;
    }
    post checkReservedCells();
}

task void checkReservedCells() {
    //if (call ReservationUpdate.numSlots(nextHop.neighbor)<numreservations &&
    //  ((call Random.rand16())%100)+(call NeighborSet.getNumNeighbors()))<100) {
    if (call ReservationUpdate.numSlots(nextHop.neighbor)<numreservations &&
        ((call Random.rand16())%100<50)) {
        post addReservation();
        return;
    }
    if (call ReservationUpdate.numSlots(nextHop.neighbor)<numreservations) {
        post removeReservation();
        return;
    }
    if (call ReservationUpdate.numSlots(nextHop.neighbor)==numreservations) {
        printf("\n\TSCH ================= \n");
        return;
    }
}

task void addReservation() {
    printf("\n\n\TSCH ReservationUpdate.addSlot to \nreturned=\%i\", nextHop.neighbor,
        (call ReservationUpdate.addSlot(nextHop.neighbor)));
}

task void removeReservation() {
    printf("\n\TSCH ReservationUpdate.removeSlot to \nreturned=\%i\", nextHop.neighbor,
        (call ReservationUpdate.removeSlot(nextHop.neighbor)));
}

task void sendDataMessage() {
    error_t sendError;
    uint8_t i=0;
    while (i<QUEUELENGTH) {
        if (queuedPktUsed[i]==FALSE) {
            queuedPktUsed[i]=TRUE;
        }
    }
```c
((cc2420_header_t*)queuedPkt[i].header)->dest=nextHop.neighbor;
((data_format_t*)queuedPkt[i].data)->value=stavaluecounter;
sendError = CallSend.send(queuedPkt[i], allself(data_format_t));
if (sendError==FAIL) {
    queueasserted[i]=true;
    //printf("\n=TSCHP Packet %i Send sent error=%i", i, sendError);
    stavaluecounter++;
    return;
} i++;

printf("\n=TSCHP ERROR no free queuedPkt cell");printflush();
}

void printStats()
{
    uint8_t i;
    for (i=0;i<7;i++) {
        printf("%lu
",stats[i].neighbor.stats[i].first_received,stats[i].last_received,stats[i].num_received);
    }
}

void printDebug()
{
    switch (printDebugCounter) {
        /* case 0: call PrintGlobalTime.print();break;
        case 1: call PrintTSCHQueue.print();break;
        case 2: call PrintForwarding.print();break;
        case 3: call PrintTWASlot.print();break;
        case 4: call PrintAdvertise.print();break;
        case 5: call PrintReservation.print();break;
        case 6: call PrintNeighbors.print();break;
        case 7: call PrintCellUsage.print();break;*/
        default: printDebugCounter=0;break;
    }
}

void blinkLeds()
{
    call Leds.led0Toggle();
    call Leds.led1Toggle();
    call Leds.led2Toggle();
    if (blinkBool==TRUE) {
        post blinkLeds();
        blinkBool=FALSE;
    }else {
        blinkBool=TRUE;
    }
}

// helper function for check neighbor,usage
bool isNeighborFunction(uint8_t neighbor) {
    uint8_t i=0;
    for (i=0;i<MAXNUMNEIGHBORS;i++)
        if (neighbors[i].used==TRUE && neighbors[i].id==neighbor) {
            return TRUE;
        }
    return FALSE;
}
```
/** interfaces */

//Boot
event void Boot.booted() {
  uint8_t i;
  printf("\nBooted!\n"); printf("\n");
  blinkBoil = TRUE;
  call StdControl.start();
  call Notify.enable();
  call Ports34.makeOutput();
  stats[0].neighbor=1;
  stats[1].neighbor=2;
  stats[2].neighbor=3;
  stats[3].neighbor=4;
  stats[4].neighbor=5;
  stats[5].neighbor=6;
  stats[6].neighbor=7;
  for (i=0;i<7;i++) {
    stats[i].first_received=0;
    stats[i].last_received=0;
    stats[i].num_received=0;
  }
  if (!((call ActiveMessageAddress.ownAddress())==1)) {
    call Timer.startPeriodic(10000);
  }
}

//SoftwareInit
command export_t SoftwareInit.init() {
  uint8_t i;
  for (i=0;i<QUEUELENGTH;i++) {
    queuedPtctUsed[i]=FALSE;
  }
  return SUCCESS;
}

//Timer
event void Timer.fired() {
  uint8_t i=0;
  message_t msg;
  post getNextHop();
  //exam

  for (i=0;i<MAXNUMNEIGHBORS;i++) {
    if (neighbors[i].id==((cc2430_header_t*)msg)->src) {
      neighbors[i].status=TRUE;
      neighbors[i].timepath=0;
    }
  }
}

//Notify
event void Notify.notify(button_state_t val) {
  if (val==TRUE) {
    if ((call ActiveMessageAddress.ownAddress())==1) {
      post printStats();
      // post printDebug(); //exam
    } else {
      post sendData();
    }
  }
}
event void Send.sendDone(message_t *msg, error_t error) {
  uint8_t i=0;
  while (i<QUEUELNGTH){
    if(squeuedPkt[i]==msg) {
      squeuedPkt[i]=0;
      Send.sendDone msg=cp error=i
      post sendDATMessage(); // creates infinite stream of packets
      i++;
    }
  }
  printf(\n"\n-- TSC# ERROR: I did not send msg=\p",msg);printfflush();
}

event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
  uint8_t i=0;
  neighbors[i].pktrx=0;
  neighbors[i].misspkt=0;
  neighbors[i].broadcastpktrx=0;
  neighbors[i].pktrx = call NeighborGet.getNumSent(neighbors[i].id); // to calculate misspkt
  // Rx adv pkt and get some neighbor table entry, exam
  if (i=NeighborFunction(\((cc2420_header_t*)msg->header)\->src)==FALSE){
    while(i=MAXNUMNEIGHBORS){ // add to neighbor list
      if (neighbors[i].used=FALSE){
        neighbors[i].used=TRUE;
        neighbors[i].id=\((cc2420_header_t*)msg->header\)->src; // get
        neighbors[i].type=\((cc2420_header_t*)msg->header\)->type; // get
        neighbors[i].linkQuality=0;
        neighbors[i].confidence=0;
        neighbors[i].numSent=0;
        neighbors[i].numSentOK=0;
        break;
      }
    }
  if (i=MAXNUMNEIGHBORS) { // no available entry
    return msg;
  }
  // Rx adv pkt finished, Neighbor table entry
  // get packet received entry, exam sta:
  for (i=0;i<MAXNUMNEIGHBORS; i++){
    if (neighbors[i].id=\((cc2420_header_t*)msg->header\)->src){
      if (neighbors[i].type=AMSCHADV || AMSCH_DATA){
        neighbors[i].pktrx++;
      }
    }
  }
}

59
//finished receive number pkt

//get misack pkt, start exam

for (i=0;i<MAXNUMNEIGHBORS;i++) {
  if ((neighbors[i].id==((ccl420_header_t*)msg->header)->src)
    if (((ccl420_metadata_t*)msg->metadata)->ack==false)
      neighbors[i].misackpkt++;

  neighbors[i].misackpkt=neighbors[i].pkttx-neighbors[i].misackpkt;
}

//get misackpkt finished

//get broadcast rx start

for (i=0;i<MAXNUMNEIGHBORS;i++) {
  if ((neighbors[i].id==((ccl420_header_t*)msg->header)->src)
    if (neighbors[i].type==AM_TECH_ADV)
      neighbors[i].broadcastpkttx++;
}

//get broadcast rx finished

//GET ReExp and ReCntx start

post bconcntl();

//get status and pathFailure
post pathfailurestatus();

/**************************exam finished neighbor table***************************/

post printf("\n=TSCHD Received msg=\n rssi=%d",msg,((ccl420_metadata_t*)msg->metadata->rssi));
//call PrintPacket.print(msg);printfflush();
post blinkLED();

post printDebug(); //exam
post printTable(); //exam
post joinPriorityFunction(); //exam
/* while (i<7) */
  if (((ccl420_header_t*)msg->header)->src==states[i].neighbor)
    if (states[i].first_received==false)
      states[i].first_received = ((data_format_t*)msg->data)->value;
    states[i].last_received = ((data_format_t*)msg->data)->value;
    states[i].num_received++;
    return msg;
}
/* */

/**************************ScanSuperframe***************************/

((superframe_t*)msg->data)->sfid=sfid;
((superframe_t*)msg->data)->numlots=nuelots;
((superframe_t*)msg->data)->activeflag=activeflag;

60
const sccdflink_t sccdflink_init(sccdflink_t *s)
{
    // initialize fields
    /*
     * This function initializes structures allocated by the
     * memory manager. dialogpar_t and sccdflink_t structure
     * are examples. Use this function in application.
     */
    s->linkID = 0;
    s->linkQuality = 0;
    s->status = DialogActive;
    s->inputPort = 0;
    s->outputPort = 0;
    return s;
}

int main()
{
    // main function to test the implementation
    sccdflink_t s;
    s = sccdflink_init(&s);
    return 0;
}
neighbors[i].linkQuality=stamp_link;
if (neighbors[i].confidence==356){
    neighbors[i].confidence++;
}
break;
}
i++;
}

//ReservationUpdate
event void ReservationUpdate.done(bool add, Uint6_t neighbor, Error_t error_t

//ActiveMessageAddress
event async event void ActiveMessageAddress.changed();

//PrintDebug
event void PrintTSCQmem.done() {printDebugCounter++;print printDebug();}
event void PrintForwarding.done() {printDebugCounter++;print printDebug();}
event void PrintTIDASlot.done() {printDebugCounter++;print printDebug();}
event void PrintAdvertise.done() {printDebugCounter++;print printDebug();}
event void PrintGlobalTime.done() {printDebugCounter++;print printDebug();}
event void PrintReservation.done() {printDebugCounter++;print printDebug();}
event void PrintNeighbors.done() {printDebugCounter++;print printDebug();}
event void PrintCellUsage.done() {printDebugCounter++;print printDebug();}
Device TSC.h

ifndef TSC_H
#define TSC_H
#include "printf.h"

//main parameters
enum {
    LENGTHCELLFRAME = 101,
    CUERULENGTH = 5,
    TVRETRIES = 5,
    MAXNUMUNBANDEDOM = 10,
    NUMONGOINGRES = 3,
    NUMRESRETries = 3,
    ADVSLOTOFFSET=0,
    NUMRESLOTOFFSET=0,
    DELAY_REMOVE_FROZEN_RESERVATIONS=30000//in ms
};

//Slot Properties
enum {
    STARTTIME = 132, //4ms [initial 3ms)
    CHAN_SEL_TIME = 66, //2ms [initial 1ms)
    DATA_TIME = 264, //8ms (initial 8ms)
    ACK_TIME = 95, //3ms [initial 3ms)
    GUARD_TIME = 66, //2ms [initial 2ms

    IS_TIME = 99, //3ms [initial 2ms)
    SLOTSIZE = (CHAN_SEL_TIME + 2*GUARD_TIME + DATA_TIME + 2*GUARD_TIME +
                ACK_TIME + IS_TIME)
};

//Frame Properties
enum {
    NUM_CHAN = 16,
    PROBFNC仉 = 1
};

//cell types
enum {
    CELLTYPE_UGS,
    CELLTYPE_RXDATA,
    CELLTYPE_TXDATA,
    CELLTYPE_RXRES,
    CELLTYPE_TXRES,
    CELLTYPE_ADV,
    CELLTYPE_RESERVED//this is used as a temporary state during the reservation
process
};

//RES commands
enum {
    RES_REQ_CELL,
    RES_FREE_CELL,
    RES_SUCCESS,
    RES_FAIL
};

63
// packet types
enum {
    AM_TSCH_ADV  = 0,
    AM_TSCH_RES  = 1,
    AM_TSCH_DATA = 10,
    AM_TSCH_ACK  = 11
};

// enum enum declaration
// slam slam, network header.
typedef nx_struct network_header {
    nle_u16_t control;
    nle_u32_t cct;  // Time To Live hop counter
    nle_u16_t tsf;
    nle_u32_t graphId;
    nle_u16_t destAddress;
    nle_u16_t srcAddress;
} network_header_t;

// destination of NEAGENT types enum, for link scheduling
enum {
    BEST_NEIGHBOR,
    BEST_GRAPH,
    BEST_PROXY,
    BEST_BROADCAST
};

// linktype enum
enum {
    TRANSPORT,
    NORMAL
};

// mode of device
enum {
    LISTEN,
    RADIO
};

---------
// superFrame packet formats (enum)
// linktable packet enum, modified 2010/08/23
typedef nx_struct linktable_t {
    nle_u8_t neighbour;
    nx_bool_t rlink;
    nx_bool_t imlink;
    nle_u16_t rlink;
    nle_u16_t channelOffset;
    nle_u16_t t linktype;
} linktable_t;

typedef nx_struct superframe_t {
    nx_bool_t activeflag;
    nle_u16_t sfid;
    nle_u16_t seqNo;
    linktable_t links[4];
} superframe_t;

---------
// packet formats
typedef nx_struct data_format_t {
    nle_u16_t value;
    nle_u16_t valuel;
} data_format_t;

// then follows the sequence of traversed nodes

// data_format_t;
typedef nx_struct adv_format_t {
    nxle_uint32_t height;
    nxle_uint16_t slotOffset;
    nxle_uint8_t channelOffset;
    nxle_uint8_t amp;
} adv_format_t;

typedef nx_struct res_format_t {
    nxle_uint8_t request;
    nxle_uint16_t slotOffset;
    nxle_uint8_t channelOffset;
} res_format_t;

typedef uint32_t cellType_t;
typedef uint16_t slotOffset_t;
typedef uint8_t channelOffset_t;

typedef struct slotChannel_t {
    bool exists;
    slotOffset_t slotOffset;
    channelOffset_t channelOffset;
} slotChannel_t;

typedef struct stats_t {
    uint16_t neighbor;
    uint32_t first_received;
    uint32_t last_received;
    uint32_t num_received;
} stats_t;

typedef struct heightNeighbor_t {
    bool exists;
    uint8_t height;
    uint16_t neighbor;
} heightNeighbor_t;

#endif

65
Device Neighbot.h file

#ifndef NEIGHBORS_H
#define NEIGHBORS_H

//neighbor table entry definition
typedef struct neighborEntry_t {
    bool _used;
    uint16_t id;
    uint32_t height;
    uint8_t linkQuality;
    uint8_t confidence;
    uint32_t numSent;
    uint32_t numSentOK;
    uint32_t lastHeardTimestamp;
    //exam
    uint16_t nickname;
    uint32_t joinPriority;
    bool timesourceFlag;
    uint32_t status;
    uint32_t bexp;
    uint32_t bontrs;
    uint16_t avgrel;
    uint32_t pkttx;
    uint16_t misackpkt;
    uint16_t pkttx;
    uint16_t broadcast_pkttx;
    uint8_t type; //just for join priority
} neighborEntry_t;

#endif
Network Manager TSCHP

#include "TSCH.h"
#include "UserButton.h"

module TSCHP {
    //general
    provides interface Init as SoftwareInit;
    uses interface Boot;
    //from above
    uses interface Advertise;
    uses interface Send;
    uses interface NeighborGet;
    uses interface Receive;
    uses interface ReservationUpdate;
    uses interface StdControl;
    uses interface ActiveMessageAddress;
    uses interface Leds;
    //debug
    uses interface Random;
    uses interface PrintPacket;
    uses interface Notify<button_status_t>;
    uses interface Timer<TMilli> as Timer;
    uses interface HComb4G30GeneralIO as Port34;
    uses interface DebugPrint as PrintReservation;
    uses interface DebugPrint as PrintNeighbors;
    uses interface DebugPrint as PrintTSCHQueue;
    uses interface DebugPrint as PrintSIMAList;
    uses interface DebugPrint as PrintAdvertise;
    uses interface DebugPrint as PrintForwarding;
    uses interface DebugPrint as PrintGlobalTime;
    uses interface DebugPrint as PrintTable2; //exam
}

implementation {

    message_t queuedPkt[QUEUELENGTH];
    bool queuedPktUsed[QUEUELENGTH];
    heightNeighbor_t nextHop;
    uin32_t printDebugCounter = 0;
    uin32_t dataValueCounter = 0;
    uin32_t numReservations = 1;
    bool BlinkBooFy;
    stats_t stats[7];
    //****************************Exam Declarations******************************/
    //exam superframe
    uin32_t sfid;
    uin32_t numSfcts;
    linkTable_t* links[4];
    bool activeFlag;

    //**************************** helper functions
    //******************************/

    task void getNextHop();
    task void checkReservedCells();
    task void sendDATAMessage();
    task void addReservation();
    task void removeReservation(),
    task void printState();
    task void printDebug();
    task void blinkLeds();

    task void getNextHop() {
        //nextHop = call NeighborGet.minHeight();
        nextHop.exists = TRUE;
    }
}

67
nextHop.neighbor = 1;
if (nextHop.object==FALSE) {
    printf("\n==TSCH ERROR no minHeight destination")
    printf flush();
    return;
}
post checkReservedCells();
}

void checkReservedCells() {
// if (call ReservationUpdate.getNumslots(nextHop.neighbor) < numreservations
// 46 (call Random.rand1()) % 100((2+(call NeighborGet.getNumNeighbors())))) < 100)) {
if (call ReservationUpdate.getNumSlots(nextHop.neighbor) < numreservations 44
((call Random.rand1()) % 100 < 80)) {
    post addReservation();
    return;
}
if (call ReservationUpdate.getNumSlots(nextHop.neighbor) > numreservations) {
    post removeReservation();
    return;
}
if (call ReservationUpdate.getNumSlots(nextHop.neighbor) == numreservations) {
    printf("\nTSCH ================ READY
____________________ \n"");
    return;
}

void addReservation() {
    printf("\nTSCH ReservationUpdate.addSlot to %i returned=%i", nextHop.neighbor,
            (call ReservationUpdate.addSlot(nextHop.neighbor)))
    ;
}

void removeReservation() {
    printf("\nTSCH ReservationUpdate.removeFrom %i returned=%i", nextHop.neighbor,
            (call ReservationUpdate.removeFrom(nextHop.neighbor)))
    ;
}

void sendDATAMessage() {
    error = sendError;
    uint32_t isw;
    while (i<QUEUELENGTH) {
        if (queuedPtr[i].object==FALSE) {
            queuedPtr[i].object=TRUE;
            (cc2420_header_t*)queuedPtr[i].data)->dest=nextHop.neighbor;
            // (data_format_t*)queuedPtr[i].data)->value=dataValueCounter;
            // (data_format_t*)queuedPtr[i].data)->value=dataValueCounter;
            // (superframe_t*)queuedPtr[i].data)->ef=ef;
            // (superframe_t*)queuedPtr[i].data)->numslots=numslots;
            // (superframe_t*)queuedPtr[i].data)->activeFlag=activeFlag;
            // (superframe_t*)queuedPtr[i].data)->links[0].neighbour=links[0].neighbour:
            // (superframe_t*)queuedPtr[i].data)->links[0].Tlink=links[0].Tlink;
            // (superframe_t*)queuedPtr[i].data)->links[0].Rlink=links[0].Rlink;


68
```c
70

```
```
stats[1].first_received=0; 
stats[1].last_received=0; 
stats[1].num_received=0; 
if (((call ActiveMessageAddress.amAddress())!=1) 
{ 
call Timer.startPeriodic(10000); 
} /* Initialize Superframe and link table here */. 
sfid=1; numslots=10; activateflag=1;

links[0].neighbour=3; links[0].Tlink=1; links[0].Flink=0; links[0].sharedlink=0 
} 
links[0].slot=2; links[0].channeloffset=4; links[0].linktype=normal;

links[1].neighbour=2; links[1].Tlink=1; links[1].Flink=0; links[1].sharedlink=0 
} 
links[1].slot=2; links[1].channeloffset=4; links[1].linktype=normal;

links[2].neighbour=2; links[2].Tlink=1; links[2].Flink=0; links[2].sharedlink=0 
} 
links[2].slot=2; links[2].channeloffset=4; links[2].linktype=normal;

links[3].neighbour=2; links[3].Tlink=1; links[3].Flink=0; links[3].sharedlink=0 
} 
links[3].slot=2; links[3].channeloffset=4; links[3].linktype=normal;

} /* SoftwareInit */ 
command error_t SoftwareInit.init() { 
for (i=0;i<QUEUELENGTH;i++) { 
queuedPktUsed[i]=FALSE; 
} 
return SUCCESS; 
}

//Timer 
event void Timer_fired() 
{ 
printNameFlag(); 
}

//Notify 
event void Notify.notify(button state t val) 
{ 
if (val==TRUE) 
{ 
if (call ActiveMessageAddress.amAddress()--1) {//since it now 
node2, both post printState();//post printState()and printDebug() does not 
implemented post printDebug(); 
} else 
{ 
post sendDATAMessage();//this task only working 
post printDebug();//working for device 
} 
}
event void SendDone(message_t *msg, error_t error) {
    uint8_t i = 0;
    while (i<QUEUELNGTH) {
        if (queuedPkt[i]=msg) {
            queueDone[i]=TRUE;
        }
        SendDone(msg->tp, error=i,
            ack=i".msg, error, ((cc2420_metadata_t*)msg->metadata)->ack); printf("%s
        post sendDATAMsg((i); //creates infinite stream of packets
        return;
    }
    post blinkLeds();
    printf("\n=TSCHP ERROR I did not send msg="tp", msg); printf("%s
}

event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len) {
    uint8_t i = 0;
    //printf("\n=TSCHP Receive.received msg="tp
    rssi=i".msg, ((cc2420_metadata_t*)msg->metadata)->rssi);
    //call PrintPacket.print(msg); printf("%s
    post blinkLeds();
    while (i<7) {
        if (((cc2420_header_t*)msg->headers)"src"=stats[i].neighbor) {
            if (stats[i].first_received==0) {
                stats[i].first_received = ((data_format_t*)msg->data)->value;
            } if (stats[i].last_received = ((data format t*)msg->data)->value;
        } if (stats[i].num_received++;
            return msg;
        }
    post blinkLeds();
    printf("\n=TSCHP received message from unknown neighbor
        ", ((cc2420_header_t*)msg->header)->src); printf("%s
    return msg;
}

event void ReservationUpdate.done(bool add, uint8_t neighbor, error_t error) {
    printf("\n=TSCHP ReservationUpdate.done add="tp neighbor="tp
        ", add, neighbor, error);
    printf("%s
    post getNextHop();
}

async event void ActiveMessageAddress.changed() {
}

event void PrintTSCHQueue.done() { printDebugCounter++; post printDebug();
}

event void PrintForwarding.done() { printDebugCounter++; post printDebug();
}

event void PrintDS1slot.done() { printDebugCounter++; post printDebug();
}

event void PrintAdvertisement.done() { printDebugCounter++; post printDebug();
}

event void PrintGlobalTime.done() { printDebugCounter++; post printDebug();
}

event void PrintReservation.done() { printDebugCounter++; post printDebug();
}