VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Fang Shanshan

# Protocol Implementation Using TI's SimpliciTI Platform

Technology and Communication
2011

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Programme of Telecommunication

# ABSTRACT

| | |
|---|---|
| Author | Fang Shanshan |
| Title | Protocol Implementation Using TI's SimpliciTI Platform |
| Year | 2011 |
| Language | English |
| Pages | 41 |
| Name of Supervisor | Gao Chao |

SYNC2SINK is a global synchronization protocol used for Wireless Sensor Network. The objective of this thesis is to implement SYNC2SINK protocol on the CC2510-based data gathering sensor network and mastering the radio interface.

The project consists of two parts: framework design and protocol implementation. My work is to implement Data frame and Sync frame which designed by my teammate to our communication system by using SimpliciTI platform.

Our designed communication systems consist of three parts: SINK, Node, and Sniffer. The achievement of SINK part is to broadcast Sync frame at 2.4 GHz periodically and correctly received the data frame. The function of Node part is to hunt the Sync frame and send out the data frame through antenna. During this transmission the sniffer will capture and analysis all the frames from SINK and Node side.

This thesis report describes the design and implantation of this communication system, and also analyzes the entire system by calculating the Packet lost rate, transmission speed, etc.

After completed this project, I familiarized with embedded system programming, I understood the SYNC2SINK protocol, timing issue, collision avoidance. I believed I am able to build an own designed Wireless sensor network in practical application.

Keywords: Embedded system, SYNC2SINK protocol, Wireless sensor network

# CONTENT

# ABBREVIATIONS

| | |
|---|---|
| **AP** | Access Point |
| **API** | Application Program Interface |
| **DK** | Development Kit |
| **ED** | End Device |
| **ISR** | Interrupt Service Routine |
| **LED** | Light Emitted Diode |
| **MAC** | Medium Access Control |
| **MCU** | Micro Controller |
| **RTT** | Round Trip Time |
| **SMPL** | Sample Modular Protocol Library |
| **SYNC** | Synchronization |
| **SoC** | System on Chip |
| **TI** | Texas Instrument |
| **TDMA** | Time Division Multiple Access |
| **USB** | Universal Serial Bus |

# LIST OF FIGURES AND TABLES

# 1 INTRODUCTION

Today, engineers start to build our environment in a smart way; this smart environment represents the next evolutionary development step in industrial, building, transportation system, utilities automation. They collected information from multiple sensors of different modalities in distributed locations to develop and monitor the system. Through the success of Bluetooth, Zigbee, Wi-Fi, wireless sensor network becomes more and more widely used in communication area.

CC2510 manufactured by Texas instrument, it provides a wireless solution for WSN. It is designed to offer a wireless communications at 2.4GHz, with a bit rate up to 500k Baud. In order to minimize the cost, CC2510 has a highly integrated 8051-compatible microcontroller and up to 32kB of in-system programmable flash memory.

This project is aim to build a short range communication system which can be used for sensing task. It is interesting to see how SYNC2SINK performs when implemented on a radio platform without Medium Access Control (MAC) support.

To finish this project, there are three main tasks I need to complete: Firstly SYNC2SINK protocol is an essential method for the entire network; Secondly the embedded system devices need 8051 C language to let them running correctly; At last, I need to analyze and statistic the data frames I received to make sure the whole system is stable and has high efficiency transmission.

As a result, I successfully built a sensor network contains one Access Point (AP) and two End Devices (ED). These two end devices are able to switch between transmission mode and receiving mode veraciously.

This final thesis consists of 6 Chapters. The first Chapter is a briefly introduction of this project. At the second Chapter of my thesis I will introduce the embedded system, and WSN. I also describe hardware devices, and software which used in my project in Chapter 3, and then I will describe the details about my communica-

tion system in Chapter 4, this part includes the network design and protocol implementation. The transmission efficiency and packet sniffer result will be analyzed in Chapter 5 of my thesis. At end of thesis I plan to summary the information and knowledge that I gained from my final project in Chapter 6.

# 2 BACKGROUND

## 2.1 EMBEDDED SYSTEM

Embedded system is a "fully embedded within the controlled device, designed for specific applications dedicated computer system." According to the UK Institution of Electrical Engineer of the definition of embedded system, it use to control, monitor or auxiliary equipment, machinery or equipment for the operation of the factory.

Embedded systems are usually running with the specific requirements of the pre-defined tasks.

Embedded system due to specific task, designers can optimize it and reducing size for cost reduction. As embedded systems are usually carried out mass production, therefore, a single cost savings can be carried out with the production of hundreds of amplification.

## 2.2 WSN

Embedded wireless sensor networks, consisting of small, low-power devices integrating a modest amount of CPU, memory, and wireless communication.

## 2.3 Low-cost sensor

- General information:

  1. Cheap price
  2. At least 7 days work time
  3. Density: 20 per cubic meter

- Functionality: perception, numeracy, traffic capacity

  1. Perceptive one or multiple objects. e.g. macro mote, RF mote.

2. Perceptive rage: RF mote, 20 meter range; sensors, 2 axis magnetometers; 2 axis accelerometers; light, temperature, pressure; laser mote, 21 kilometers./1/

- Sink vs. sensor

1. Mobility: sensory generally does not move, but can be moved. Sensors in the ocean drifting with the ocean; Sink can be moved, e.g. mobile computer which carried by aircraft.

2. Sink node have more functions than sensor node, sink has no resource constraints, sink can communicate directly, it can install GPS and power generation system.

## 2.4 Purpose of Project

Project objects:

- Build up a demo system by using CC2510 platform
- SINK & Node programming
- SimpliciTI application
- Sniffer the transmission packets
- Calculate the Round Trip Time (RTT), packet loss rate, etc.

# 3 SYSTEM OVERVIEW

## 3.1 CC2510 Development Kits (DK)

The CC2510 is a true low-cost 2.4 GHz System-on-Chip (SoC) designed for low ower wireless applications. The CC2510 the excellent performance of the state-of-the-art RF transceiver CC2510 with an industry-standard enhanced 8051 Micro Controller (MCU), up to 32 kB of in-system programmable flash memory and 4 kB of RAM, and many other powerful features. The small 6×6 mm package makes it very suited for applications with size limitations. /2/

The CC2510 is highly suited for systems where very low power consumption is required. This is ensured by several advanced low-power operating modes. The CC2511Fx adds a full-speed Universal Serial Bus (USB) controller to the feature set of the CC2510Fx. Interfacing to a PC using the USB interface is quick and easy, and the high data rate (12 Mbps) of the USB interface avoids the bottlenecks of RS-232 or low-speed USB interfaces. Figure 1 shows us the Kit contents for CC2510-11DK.

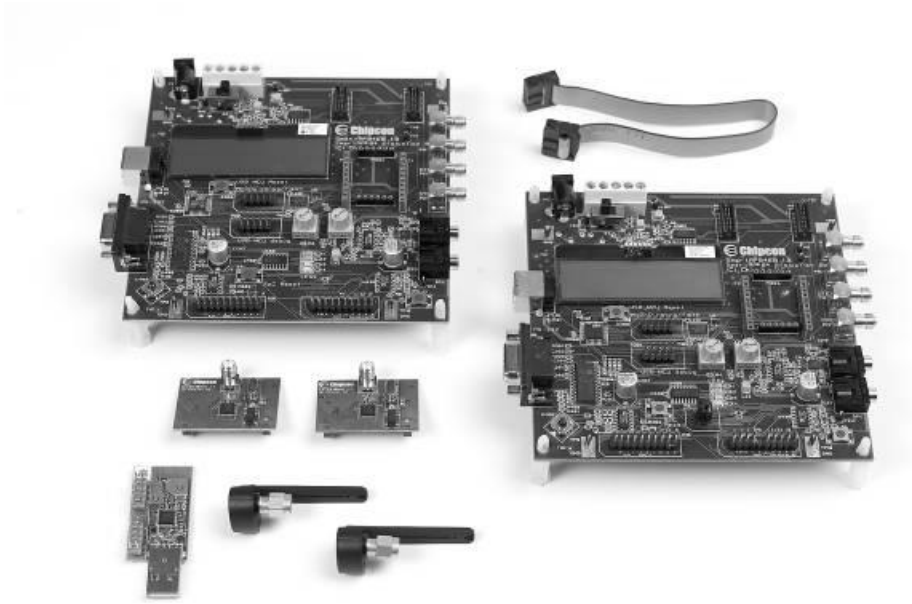

**Figure 1.** Kit contents for CC2510-11DK.

• 2 SmartRF04EB

• 2 CC2510EM

• 1 CC2511 USB Dongle

• 2 antennas

• 2 USB cables

• 1 USB extension cable /3/

## 3.2 IAR Embedded Workbench® for 8051

IAR Embedded Workbench with its optimizing C/C++ compiler provides extensive support for a wide range of 8051 devices. The optimizing compilers generate very compact and efficient code. Ready-made examples and code templates for your embedded project are included with the product. The standard edition of IAR Embedded Workbench also includes built-in plugins to different hardware debug systems. /4/ the screenshot of IAR workbench shows in Figure 2.

### 3.2.1 Key components

- Integrated development environment with project management tools and editor
- Highly optimizing C and C++ compiler for 8051
- Configuration files for 8051 devices from different manufacturers
- Run-time libraries with complete source code
- Relocating 8051 assembler

- Linker and librarian tools
- C-SPY debugger with 8051 simulator and support for RTOS-aware debugging on hardware
- ROM-monitor and JTAG drivers as well as source code and project for creating your own ROM-monitor driver
- Example projects for 8051 and code templates
- User and reference guides in PDF format
- Context-sensitive online help

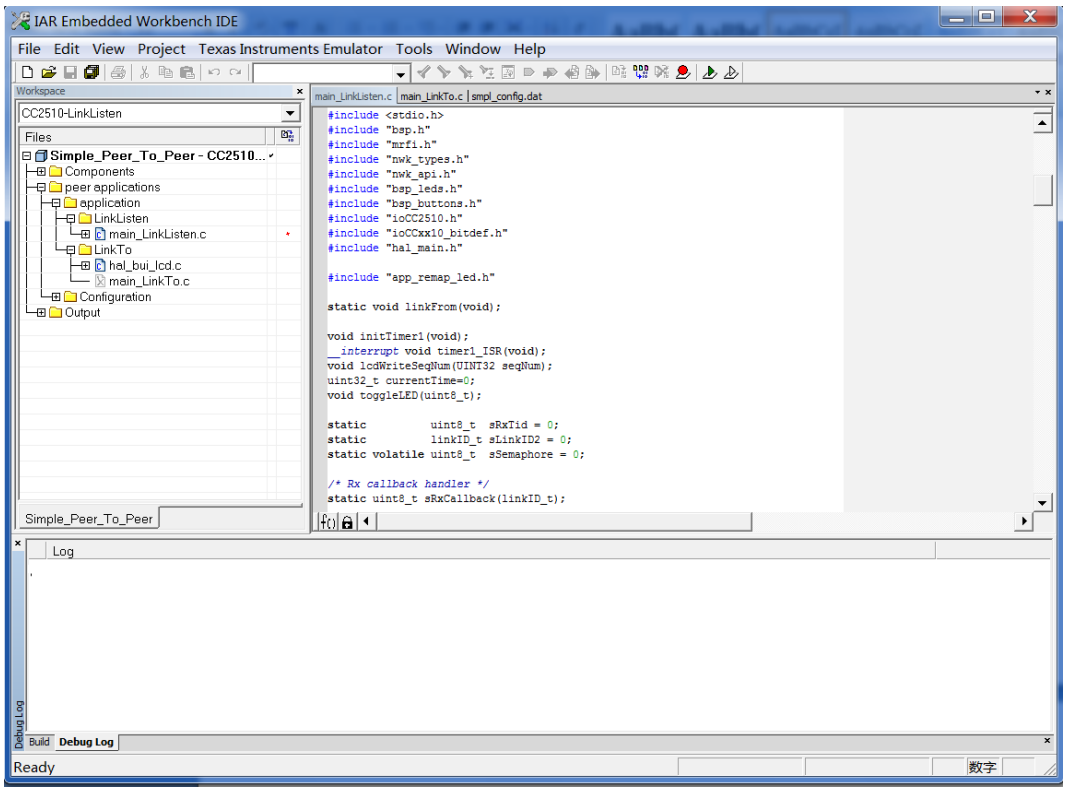


**Figure 2.** IAR Embedded Workbench.

## 3.3 SimpliciTI

SimpliciTI is a TI proprietary low-power RF network protocol. SimpliciTI has very low cost of memory; it only needs less than 8k flash memory and 1k ram

space depend on configuration. It supports 2 basic topologies: strictly peer-to-peer and a star topology. /5/

The protocol support is realized in a small number of API calls. These APIs support customer application peer-to-peer messaging. The association between two applications, called linking, is done at run time. The main purpose of this application is do messaging in the air.

SimpliciTI can provide many wireless solution to the customer it can support many Ems manufactured by Texas Instrument (TI) such as MSP430+CC110x/2500, CC1110/2510, CC1111/CC2511, CC2430, CC2520.

SimpliciTI support sleep mode for the devices in order to extend the working time. The key components of SimpliciTI are shown in Figure 3.

### 3.3.1 Key components

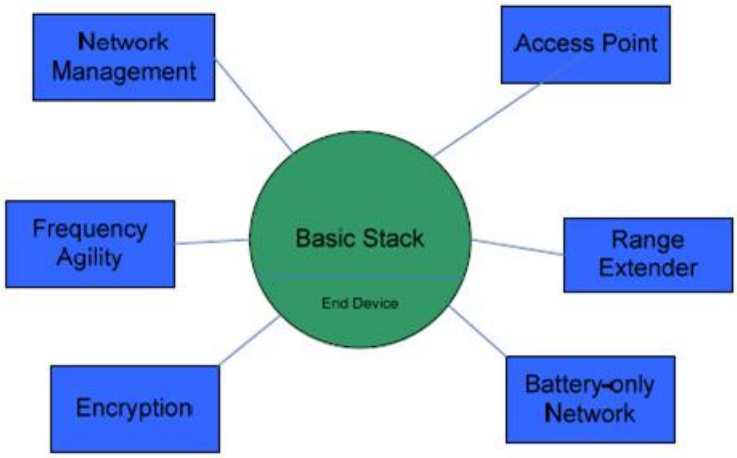


**Figure 3.** SimpliciTI modular components. /6/

## 3.4 TI SmartRF sniffer

The SmartRF™ Packet Sniffer is a PC software application used to display and store RF packets captured with a listening RF HW node. Various RF protocols are supported. The Packet Sniffer filters and decodes packets and displays them in a convenient way, with options for filtering and storage to a binary file format./7/ Figure 4 is the screenshot I captured for SmartRF packet sniffer.

### 3.4.1 Hardware platform

The packet sniffer can be used with different HW platforms. The following HW can be used:

• CC2430DB

• SmartRF04EB + (CC2430EM, CC2530EM, CC1110EM or CC2510EM)

• SmartRF05EB + (CC2430EM, CC1110EM, CC2510EM, CC2520EM or CC2530).

• CC2531 USB Dongle.

• CC Debugger + SmartRFCCxx10TB

The applicable board must be connected to the PC through USB.

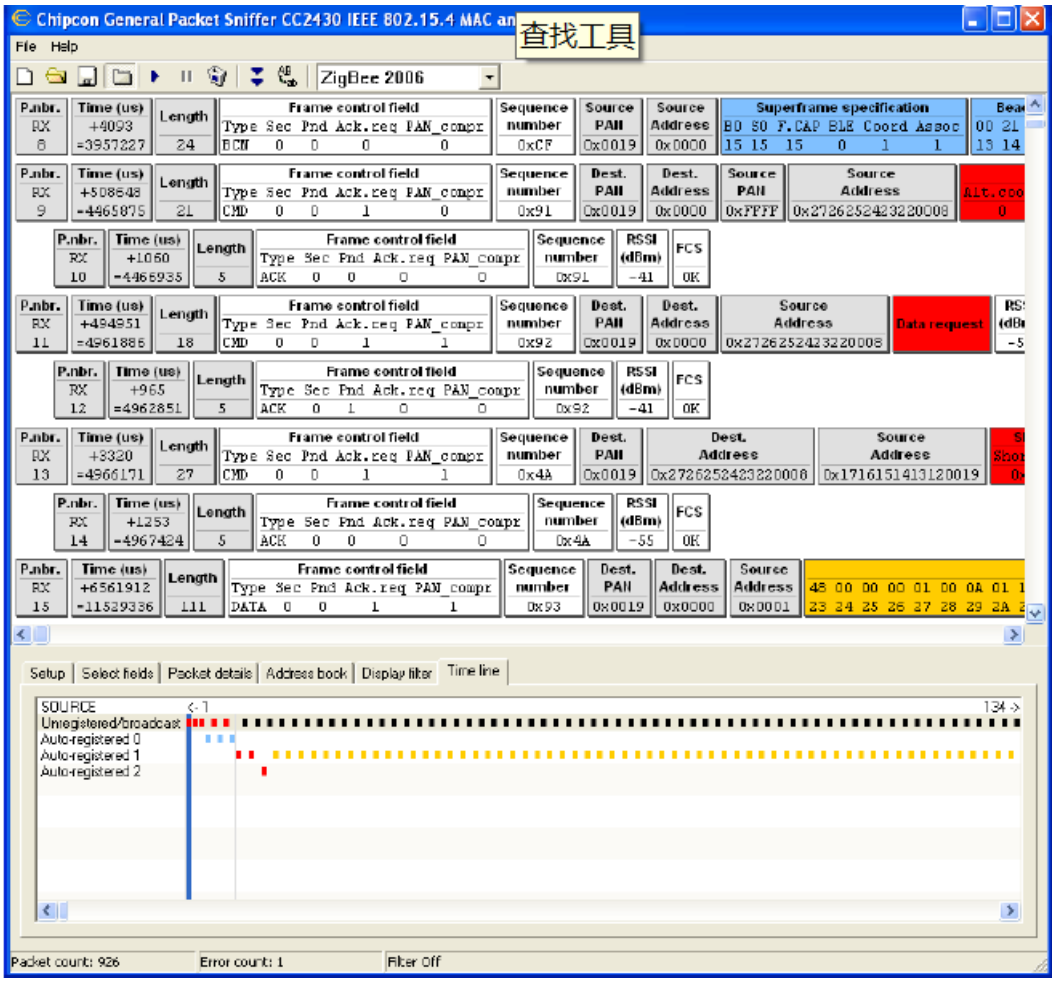


**Figure 4.** SmartRF packet sniffer screen shot.

# 4 IMPLEMEANTAION

In order to implement my project, I need to achieve two objects:

- Build network for the entire system which includes 1 sink, 2 nodes, and a sniffer.

- Use SimpliciTI application to implement my protocol stack. The timer and framework which designed by my teammate are also need to be modify, after that the program is able to drive the devices.

## 4.1 Network design

There are two types of devices in our system: 3EDs and 1 AP, Figure 5 shows us the connection between these devices.

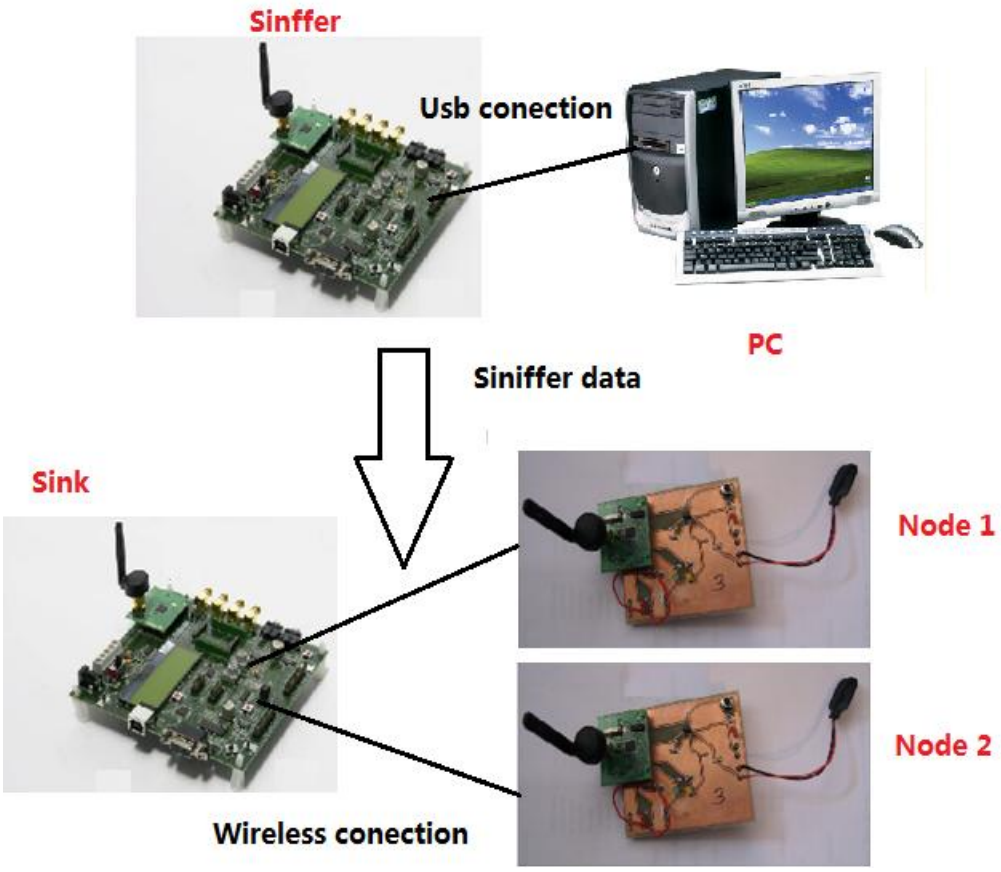


**Figure 5.** Connection overview.

## 4.2 Hardware Setting

- Plug CC2510EM into SmartRF04EB used as SINK.
- Plug CC2510EM into Tan's PCB board used as Nodes.
- Plug CC2510EM into SmartRF04EB used as Packet sniffer. Connect SmartRF04EB to a PC with a USB cable.
- Connect Antennas to all CC2510EM in order to provide radio support for the system.

- 9-Volts battery can provide power supply to SINK and Nodes. The packet sniffer is able to get the power supply from the USB cable directly. Figure 6 show us the battery we are used in our project.


**Figure 6.** 9-volt Power supply.

## 4.3 Software Application

The main Evaluation board of this project is CC2510EM; CC2510 includes an 8-bit CPU core which is an enhanced version of the industry standard 8051 core, so all the code we need to implement is 8051 C languages. By using the IAR Embedded Workbench, some of the header file I can use them directly, this saves a lot of work for the programming part.

There are two main parts in this WSN node and sink, Both of them belong to End device. So the system I designed must follow this session figure. Figure 7 shows us the way two EDs communicate.

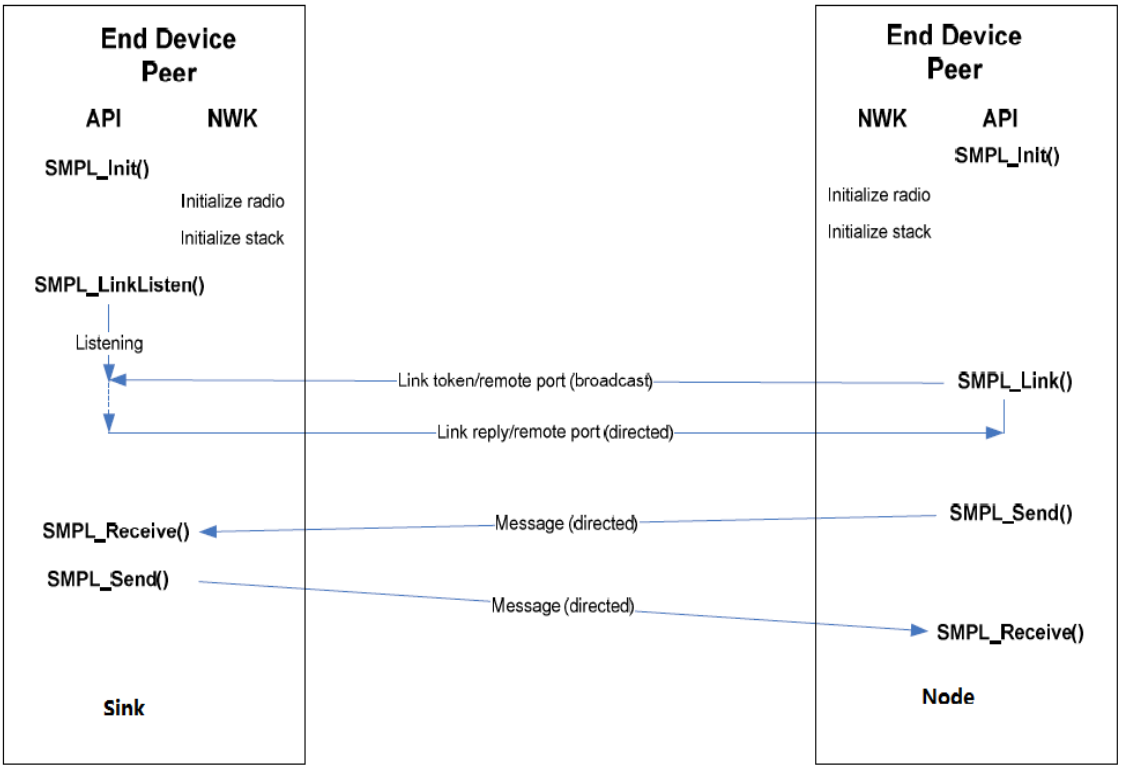


**Figure 7.** Peer to Peer session.

There are 6 types of sources and headers I used in this project all the sources and headers will be classify in Table 1; each of them has unique function to drive the devices:

| | |
|---|---|
| **Components** | Code packet for SimpliciTI |
| **Configurations** | Network configuration of SimpliciTI |
| **HAL** | Hardware management |
| **HAL_BUI** | API for SmartRF04EB |
| **ioCC2510** | Input and Output controlled for CC2510 |

**Table 1.** Sources and headers.

### 4.3.1 SimpliciTI application

SimpliciTI is a RF application which provide by TI. After implement the coding part of SimpliciTI we can do the messaging between sink and nodes.

#### 4.3.1.1 Common Data Types

The first part of implement SimpliciTI is to get familiar the common data types, I list all the data types I used in this application in Table 2.

| Data type | Description |
|---|---|
| typedef signed char int8_t | 8 bits signed integer number |
| typedef signed short int16_t | 16 bits signed integer number |
| typedef signed long int32_t | 32 bits signed integer number |
| typedef unsigned char uint8_t | 8 bits unsigned integer number |
| typedef unsigned short uint16_t | 16 bits unsigned integer number |
| typedef unsigned long uint32_t | 32 bits unsigned integer number |
| typedef unsigned char linkID_t | Singed character for LinkID only |
| typedef enum smplStatus smplStatus_t | enumerated types for smpl statues |

**Table 2.** SimpliciTI common data types.

#### 4.3.1.2 Statues

The following statues values are used in various Sample Modular Protocol Library (SMPL) function. All these statues are belonging to the type **typedef enum smplStatus smplStatus_t** which I mentioned above. I will specify the relevant returns individually for each Application Program Interface (API) symbol

in the following part of this section. I give a clearly description of each return types in Table 3.

| **Name** | **Description** |
|---|---|
| SMPL_SUCCESS | Operation successful. |
| SMPL_TIMEOUT | A synchronous invocation timed out. |
| SMPL_BAD_PARAM | Bad parameter value in call. |
| SMPL_NOMEM | No memory available. Object depends on API |
| SMPL_NO_FRAME | No frame available in input frame queue. |
| SMPL_NO_LINK | No reply received to Link frame sent. |
| SMPL_NO_JOIN | No reply received to Join frame sent. |
| SMPL_NO_CHANNEL | Channel scan did not result in response on at least 1 channel. |
| SMPL_NO_PEER_UNLINK | Peer could not delete connection. Returned in reply message to unlink request. |
| SMPL_TX_CCA_FAIL | Frame transmits failed because of CCA failure. |
| SMPL_NO_PAYLOAD | Frame received but with no application payload. |
| SMPL_NO_AP_ADDRESS | Should have previously gleaned an Access Point address but we have none. |

**Table 3.** API function Returns.

#### 4.3.1.3 SimpliciTI API

Then I am able to use the API in SimpliciTI application. There are five types of interfaces for APIs includes initialization interface, connection interface, data interface, IOCTL interface and call back interface./8/

.These section shows the programming interfaces I used in this project:

- **BSP_Init()**

| | |
|---|---|
| **Discretion** | Not strictly part of the SimpliciTI API this call initializes the specific target hardware. It should be invoked before the SMPL_Init() call. |
| **Prototype** | Void BSP_Init(void) |
| **Parameters** | None |
| **Return** | None |

- **SMPL_Init()**

| | |
|---|---|
| **Discretion** | This function initializes the radio and the SimpliciTI protocol stack. It must be called once when the software system is started and before any other function in the SimpliciTI API is called. |
| **Prototype** | smplStatus_t SMPL__Init(uint8_t (*)(linkID_t)) |

| | |
|---|---|
| **Parameters** | The argument is a pointer to a function and causes the supplied function to be registered as the callback function for the device. Since the initialization is called only once the callback serves all logical End Devices on the platform.<br><br>The function prototype is: uint8_t sCallBack(linkID_t).<br><br>The function is invoked in the frame-receive ISR thread so it runs in the interrupt context. |
| **Return** | SMPL_SUCCESS Initialization successful.<br><br>SMPL_NO_JOIN No Join reply. Access Point possibly not yet up. Not an error if no Access Point in topology<br><br>SMPL_NO_CHANNEL Only if Frequency Agility enabled. Channel scan failed. Access Point possibly not yet up. |

- **SMPL_Link()**

| | |
|---|---|
| **Discretion** | This call sends a broadcast link frame and waits for a reply. Upon receiving a reply a connection is established between the two peers and a Link ID is assigned to be used by the application as a handle to the connection.<br><br>This call will wait for a reply but will return if it does not receive one within a timeout period so it is not a strictly blocking call. The amount of time it waits is scaled based on frame length and data rate and is automatically determined during initialization.<br><br>This call can be invoked multiple times to establish multiple logical connections. The peers may be on the same or differ- |

| | |
|---|---|
| | ent devices than previous connections. |
| **Prototype** | smplStatus_t SMPL_Link(linkID_t *lid) |
| **Parameters** | The parameter is a pointer to a Link ID. If the call succeeds the value pointed to will be valid. It is then to be used in<br><br>Subsequent APIs to refer to the specific peer. |
| **Return** | SMPL_SUCCESS Link successful.<br><br>SMPL_NO_LINK No Link reply received during wait window.<br><br>SMPL_NOMEM No room to allocate local Rx port, no more room in Connection Table, or no room in output frame queue.<br><br>SMPL_TX_CCA_FAIL Could not send Link frame. |

- **SMPL_LinkListen()**

| | |
|---|---|
| **Discretion** | This call will listen for a broadcast Link frame. Upon receiving one it will send a reply directly to the sender.<br><br>This call is a modified blocking call. It will block “for a while” as described by the following constant set in the nwk_api.c source file:<br><br>The application can implement a recovery strategy if the listen times out. This includes establishing another listen window. Note that there is a race condition in that if the listen call is invoked upon a timeout it is possible that a link frame arrives during the short time the listener is not listening. |

| | |
|---|---|
| **Prototype** | smplStatus_t SMPL_LinkListen(linkID_T *lid) |
| **Parameters** | The parameter is a pointer to a Link ID. If the call succeeds the value pointed to will be valid. It is then to be used in subsequent APIs to refer to the specific peer. |
| **Return** | SMPL_SUCCESS Link successful.<br><br>SMPL_TIMEOUT No link frame received during listen interval. Link ID not valid. |

- **SMPL_Send()**

| | |
|---|---|
| **Discretion** | This function sends application data to a peer. The network code takes care of properly conditioning the radio for the transaction. Upon completion of this call the radio will be in the same state it was before the call was made. The application is under no obligation to condition the radio. |
| **Prototype** | void SMPL_Send(linkID_t lid, uint8_t *msg, uint8_t len)PA |
| **Parameters** | lid -Link ID of peer to which to send the message.<br><br>msg -Pointer to message buffer.<br><br>len -Length of message. This can be 0. It is legal to send a frame with no application payload. |
| **Return** | SMPL_SUCCESS Transmission successful.<br><br>SMPL_BAD_PARAM No valid Connection Table entry for Link ID; data in Connection Table entry bad; no message or message too long. |

| | |
|---|---|
| | SMPL_NOMEM No room in output frame queue.<br><br>SMPL_TX_CCA_FAIL CCA failure. Message not sent. |

- **SMPL_Receive()**

| | |
|---|---|
| **Discretion** | This function checks the input frame queue for any frames received from a specific peer.<br><br>Unless the device is a polling device this call does not activate the radio or change the radio's state to receive. It only checks to see if a frame has already been received on the specified connection.<br><br>If the device is a polling device as specified in the device configuration file, the network layer will take care of the radio state to enable the device to send the polling request and receive the reply. In this case conditioning the radio is not the responsibility of the application.<br><br>If more than one frame is available for the specified peer they are returned in first-in-first-out order. Thus it takes multiple calls to retrieve multiple frames. |
| **Prototype** | smplStatus_t SMPL_Receive(linkID_t lid, uint8_t *msg, uint8_t *len) |
| **Parameters** | lid - Check for messages from the peer specified by this Link ID.<br><br>msg - Pointer to message buffer to populate with received message. |

| | |
|---|---|
| | len - Pointer to location in which to save length of received message. |
| **Return** | SMPL_SUCCESS Frame for the Link ID found. Contents of ’msg‘ and ’len ‘ are valid.<br><br>SMPL_BAD_PARAM No valid Connection Table entry for Link ID; data in Connection Table entry<br><br>bad.<br><br>SMPL_NO_FRAME No frame available.<br><br>SMPL_NO_PAYLOAD Frame received with no payload. Not necessarily an error and could be deduced<br><br>by application because the returned length will be 0.<br><br>SMPL_TIMEOUT Polling Device: No reply from Access Point.<br><br>SMPL_NO_AP_ADDRESS Polling Device: Access Point address not known.<br><br>SMPL_TX_CCA_FAIL Polling Device: Could not send data request to Access Point<br><br>SMPL_NOMEM Polling Device: No memory in output frame queue<br><br>SMPL_NO_CHANNEL Polling Device: Frequency Agility enabled and could not find channel. |

- **SMPL_Ioctl()**

| | |
|---|---|
| **Discretion** | This is the single format taken by all ioctl calls |
| **Prototype** | smplStatus_t SMPL_Ioctl(ioctlObject_t obj, ioctlAction_t act, void *val) |
| **Parameters** | obj -Object of the action requested.<br><br>act -Action requested for the specified object.<br><br>val -Pointer to parameter information. May be input or output depending on action. May also<br>be null if object/action combination requires no parametric information. |
| **Return** | SMPL_SUCCESS Operation successful<br><br>SMPL_BAD_PARM ioctl object ot ioctl action illegal |

- **CallBack()**

| | |
|---|---|
| **Discretion** | The callback (if registered) is invoked in the receive Interrupt Service Routine (ISR) thread when the frame received contains a valid application<br><br>destination address |
| **Prototype** | uint8_t sCallBack(linkID_t) |
| **Parameters** | The parameter in the callback when invoked will be populated with the Link ID of the received frame. This is the way the callback can tell which peer has sent a frame and possibly requires service. The special Link ID<br><br>SMPL_LINKID_USER_UUD is a valid parameter value in |

| | |
|---|---|
| | this context.<br><br>A call to SMPL_Receive() using the supplied Link ID is guaranteed to succeed. This is the only means by which the frame can be retrieved. |
| **Return** | 0 received frame is left in the input frame queue for later retrieval in the user thread. This<br><br>is the recommended procedure. The callback can simply set a flag or otherwise store the information about the<br><br>waiting frame. The actual reference to SMPL_Receive() should be done in the user thread.<br><br>Non-zero the frame resource is released for reuse immediately. This implies that the callback has<br><br>extracted all valid information it requires |

#### 4.3.1.4 Other API

By using SimpliciTI, the project is able to transfer messages between SINK and Nodes, but it is still not good enough to achieve the objects of this project, because I need to analyze the efficiency of this communication system.

Timer1 which controlled the broadcasting time is also needed but this part has been completed by my teammate already; I also create a LCD initialization program to print the transmission period of time on LCD screen.

- **initTimer1()**

| | |
|---|---|
| **Discretion** | The ISR of timer1, an interrupt is generated every milli-seconds. |
| **Prototype** | void initTimer1(void) |
| **Parameters** | |
| **Return** | None |

- **halBuiInitLcd()**

| | |
|---|---|
| **Discretion** | This function |
| **Prototype** | void lcdWriteSeqNum(UINT32 seqNum); |
| **Parameters** | |
| **Return** | None |

4.3.1.5 System overview

At last all the programming tasks are finished. We need to implement this code in CC2510EM. The two flow charts below will show the working procedure of this embedded system, Figure 8 is the flow chart for SINK and Figure 9 is the flow chart for Node.

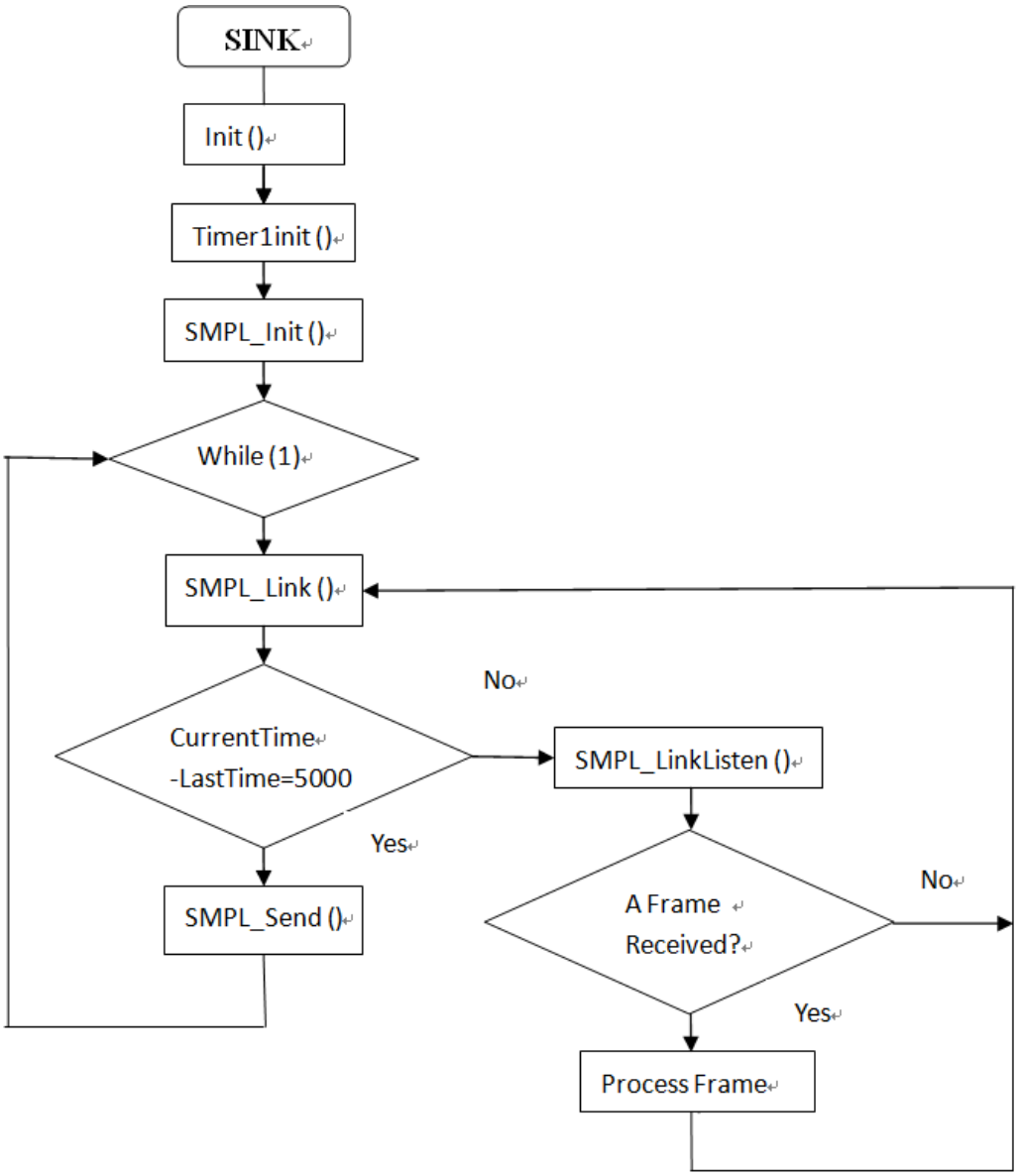


**Figure 8.** SINK flowchart.

- **SINK application**

1. Power up the evaluation board.
2. Initialize all the input and output of the CC2510.
3. Timer1 initialized
4. SimpliciTI initialized.
5. Broadcast Synchronization (SYNC) frame every five seconds.
6. If a node reply, switch to receiving mode.
7. Receive data frame from node.

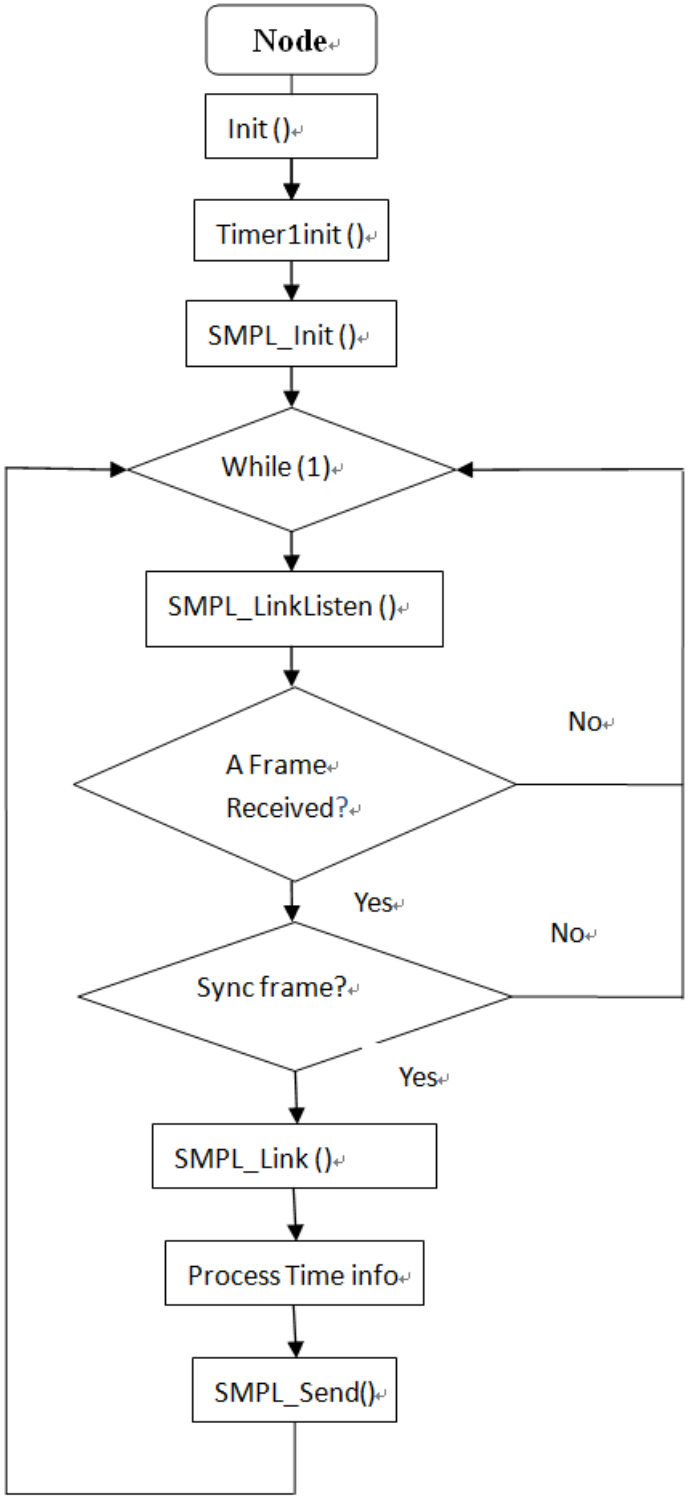


**Figure 9.** Node flowchart.

- **Node application**

1. Power up the evaluation board.
2. Initialize all the input and output of the CC2510.
3. Timer1 initialized.
4. SimpliciTI initialized.
5. Hunt sync frame.
6. If a sync frame is correctly received, switch to sending mode.
7. Send data frame to SINK.

# 5 OUTCOME OF PROJECT

This application permits construction of the following simple network.

1. The network is a three-device peer-to-peer application. Both devices are End Devices.

2. There is one SINK and two nodes.

3. The SINK sends a 13 bytes message with a 2 byte payload at varying intervals ranging from 5 seconds.

a. The two byte payload sent by the sink consists of the following:

| LinkID | Ref number |
|---|---|

b. The Light Emitted Diode (LED) to toggle is LED 2.

c. Also transaction ID is incremented by 1 in each new message. It is treated as an

unsigned value.

d. The sink implements an Rx callback handler to handle replies sent back by Nodes.

4. The nodes wait to receive a Sync message. The main application waits on a semaphore that is released when a message is received. It uses the Rx callback feature.

a. Upon message receipt the registered Rx Callback is invoked.

b. The handler toggles the LED specified in the message. The handler then posts to a semaphore that releases the main application and returns. The LED to toggle is LED 1.

c. The main application waits on the semaphore. When released it sends a message back to the Sink. This message is also consist of a 13 bytes frame with 2 byte payload.

## 5.1 SimpliciTI protocol

The functionality provided by the protocol is simply to provide support for connection-based peer-to-peer communications. The intent is to wrap the fundamental radio portion and remove that domain from the Customer's concern during development.

The functionality is realized in a simple set of API calls available to the Customer's application. The simplicity comes with the price of flexibility. The vision is that the use of this simple, small footprint protocol will be in scenarios that require only limited flexibility.

The table below shows the frame structrue we are used in our project, LinkID and Sequence number are the payloads which I have mentioned in last section. If we need to use the simpliciTI protocol, we also need some more components of the frame to support our transmission. Table 4 shows the frame structure of SimpliciTI application.

| Length | DSTADD | SRCADD | Transaction ID | Link ID | Sequence Number | User Port |
|---|---|---|---|---|---|---|
| 1 Byte | 4 Bytes | 4 Bytes | 1 Byte | 1 Byte | 1 Byte | 1 Byte |

**Table 4.** SimpliciTI frame structure.

- LENGTH: Length of remaining packet in Bytes. Inserted by FW on Tx. Partially filterable Rx.
- DSTADDR Destination address, Inserted by FW. Filterable depending on radio.
- SRCADDR Source address, Inserted by FW.
- TRACTID Transaction id Inserted by FW. Discipline depends on context. Need not be sequential.
- PORT Encryption context (7-6), Application port number (bits5-0). Inserted by FW. Port namespace reserves 0x20-0x3F for customer applications and 0-1F for NWK management.

## 5.2 System test

The system test is to running this wireless sensor network for 11 hours, during this testing period, I used TI packet sniffer to sniffer the packets which communicate between SINK and Nodes, after analyze the packet sniffer result I am able to calculate the packet lost rate and time interval between two consecutive arrivers. Figure 10 is the Packet sniffer result which I captured after the transmission finished.

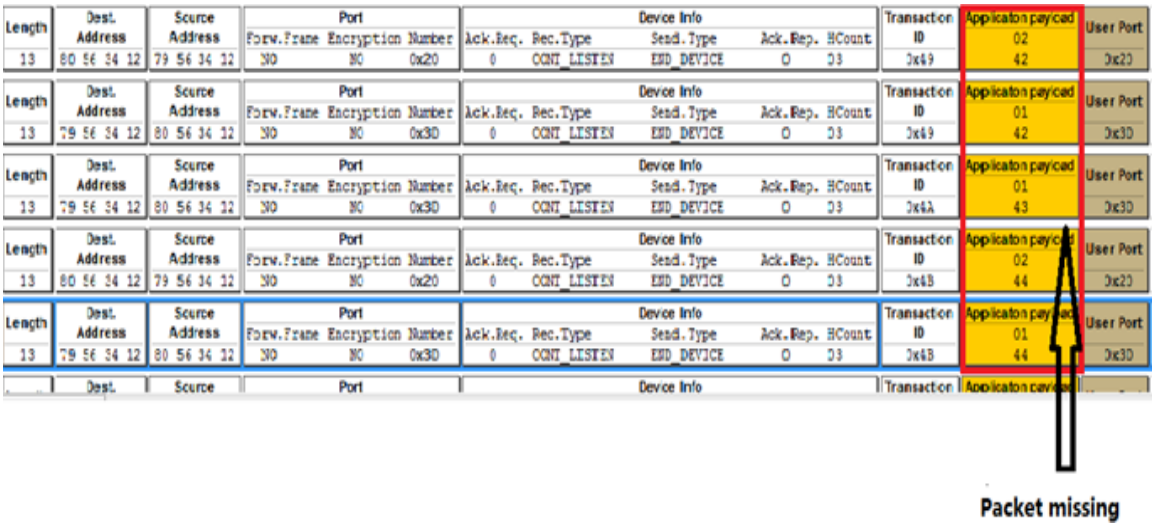


**Figure 10.** Packet sniffer result.

This is the way I calculate packet lost rate of this project. After check the application payload which captured by the packet sniffer I understand the packet with Link ID 02 and reference number 43 is missing during the transmission. Link ID 02 means this missing message is sending from Node side. Figure 11 gives us a clearly overview about frames travel in time domain.

$$R_p = \frac{N_l}{N_l + N_R} \qquad (1)$$

$R_P$: Packet loss rate

$N_l$: Number of lost packet

$N_R$: Number of packets received successfully

According to the formula above I can calculate the packet loss rate from both sides:

The lost package from Sync side is 3 packets over 7920 total packets. The packet loss rate is $3{,}78*10^{-4}$.

The lost package from Node side is 7 packets over 7920 total packets. The packet lost rate is $8.83*10^{-4}$.

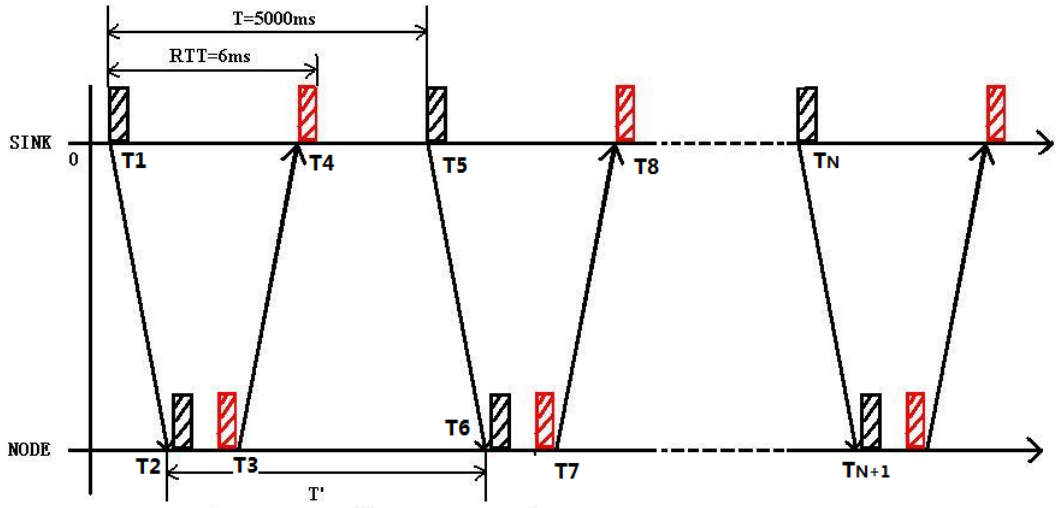


**Figure 11.** Frames travelling in time domain.

$T_1$: The accurate time SINK sends out the first SYNC frame.

$T_2$: The accurate time Node receives the first SYNC frame

$T_3$: The accurate time Node sends out the first DATA frame.

$T_4$: The accurate time SINK receives the first DATA frame.

$T_5$: The accurate time SINK sends out the second SYNC frame.

$T_6$: The accurate time Node receives the second SYNC frame.

$T_7$: The accurate time Node sends out the second DATA frame.

$T_8$: The accurate time SINK receives the second DATA frame.

$T_N$: The accurate time SINK sends out the SYNC frame with reference number N.

$T_{N+1}$: The accurate time Node receives the SYNC frame with reference number N.

RTT: Round Trip Time.

$$RTT = T_4 - T_1 \quad (2)$$

We can easily calculate the value of round trip time which is 6ms second.

$$T = T_5 - T_1 \quad (3)$$

$$T' = T_6 - T_2 \quad (4)$$

$$\Delta T = T - T' \quad (5)$$

$$\Delta T_N = T_N - T_N' \quad (6)$$

$$Sum = \sum \Delta T + \ldots \Delta T_N \quad (7)$$

$$\Delta T_A = Sum/N \quad (8)$$

Since we have a timer interrupt function in our code, I can print out all the values from $T_1$ to $T_N$.

The average value for the time interval between two consecutive arrivers can be calculated with the equation above.

$$\Delta T_A = 3.25ms$$

# 6 CONCLUSION AND DISCUSSION

By doing this project I built a wireless sensor network which includes two nodes, one SINK, and a packet sniffer. The maximum transmission range of this system is 20 meters.

After analyzed packet sniffer result, I am able to say this wireless communication system has very low packet loss rate and high efficiency during the transmission.

I learnt a lot from this project. I get familiar with 8051C language, and I understand the principle to construct an embedded system with SimpliciTI application, this can apply to many the electronic devices enhanced with 8051core.

At the beginning of this project, I spent a lot of time on learning embedded system programming, but I did not gain much from self-study. With the guidance and teaching from my supervisor, I found the programming part become much easier. At the end of this project, I found coding with 8051 C language was already proficient job for myself.

## 6.1 Further development:

Even though I get a good result after completed this project, this wireless sensor network is still on the develop steps. A lot of works can be done in the future:

- Increase the number of nodes in this WSN.

- Update the transmission method of his project into a more intelligent way, for example Time Division Multiple Access (TDMA).

- The message which we designed for our transmission did not contain much information. We could add more useful information in this message, for example detect the temperature from node and transmit the temperature value through message.

- The transmission range is very limited, by adding a range extender to the network, we can improve this weakness.

# REFERENCES

**Sources when numbered references are used**:

/1/ JN Al-Karaki, AE Kamal. Routing techniques in wireless sensor networks a survey. IEEE Wireless Communications. 2004

/2/ Texas Instrument, CC1110DKCC2510DK -- Development Kit User Manual (Rev. A, 2007). Available on the Internet: <URL: http://focus.ti.com.cn/cn/lit/ug/swru134a/swru134a.pdf>.

/3/ Texas Instrument, CC2510-CC2511DK Quick Start, Reversion 2.1, 14/9/2007. Available on the Internet: <URL: http://focus.ti.com.cn/cn/lit/ug/swru079b/swru079b.pdf>.

/4/ IAR, IAR Embedded Workbench® for 8051datasheet, 2010. Available on the Internet: <URL: http://www.iar.com/website1/1.0.1.0/244/1/ >.

/5/ Texas Instrument, SimpliciTI Developers Notes, 2008. Available on the Internet: <URL:http://tnt.etf.rs/~ms1sms/lab/SimpliciTI%20Developers%20Notes.pdf>.

/6/ Texas Instrument, SimpliciTI Specification 2008. Available on the Internet: <URL: http://tnt.etf.rs/~ms1sms/lab/SimpliciTI%20Specification.pdf>.

/7/ Texas Instrument SmartRF Packet Sniffer User Manual Rev 1.10, 2008.Available on the Internet: <URL: http://edge.rit.edu/content/P11207/public/SmartRFPacketSnifferUserManual.pdf>.

/8/ Texas Instrument, SimpliciTI API, 2008. Available on the Internet: <URL: http://yawing.googlecode.com/svn/trunk/yawing/Documents/SimpliciTI%20API.pdf>.