Exploring an IPv6 protocol for mobile sensor network communication

by

Weiqi Zhang

TR13-226, May 31, 2013

This is an unaltered version of the author's MCS thesis

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
E-mail: fcs@unb.ca
http://www.cs.unb.ca

Abstract

This research explores IPv6 in mobile wireless sensor networks (WSNs). An indoor WSN mobile sensor network testbed of length 24 m was built and used for mobile WSN testing. The test network enabled the use of one or two moving nodes and six stationary nodes. TelosB sensor nodes were used for testing.

The thesis presents a detailed explaination of sending and receiving User Datagram Protocol (UDP) packets using the IPv6 Low Power Wireless Area Network (6LoWPAN) software stack in the Berkeley Low power Internet Protocol (BLIP) implementation of TinyOS 2.1.1 and 2.1.2. A Java based Web application called WSNWeb was built that displays real-time route topology changes and sensor data. The data is updated in a log file, and used to compute packet loss and determine the number of route topology changes.

We created 35 test cases, 15 with two moving nodes, and 20 with one moving node. On a test track, model train velocities between 0.076 m/s and 0.376 m/s were used, with three different routing table update periods (RTUPs) of 60 s, 6 s, and 0.6 s. The results show that the one moving node 0.6 seconds RTUP has significantly higher packet loss (up to 1.4% compared to 0.16%) over a five hour test compared to RTUPs of 60 s and 6 s. The two moving nodes test shows that RTUP of 0.6 s still has a higher packet loss compared to RTUPs of 6 s and 60 s.

Acknowledgements

Foremost, with all my sincerity I would like to thank my supervisor, Dr. Bradford G. Nickerson for his kind support of my master's degree study and research. Dr. Nickerson's guidance helped me in all the time of research and writting of my thesis. He is very patient and was always available for help and advice.

I would like to thank my dear parents, Mr. Zhongwei Zhang and Mrs. Lifeng Gao, for aspiring me to give my best. Although they are far away from me in China, they never stop cheering and encouraging me. Without their financial and emotional support, I could not have imagined completing my thesis.

I would also like to thank the UNB Faculty of Computer Science for offering so many helpful courses and providing lab equipment for my study and research.

Last but not least, I would like to thank my friends whoever near or far, for giving me so much fun and cheering me up.

Table of Contents

Abstract ii

Acknowledgments iii

Table of Contents vi

List of Tables vii

List of Figures viii

Abbreviations ix

1 Introduction 1

1.1 Wireless Sensor Networks 1

1.2 Thesis Objectives 2

2 IPv6 in Wireless Sensor Networks 3

2.1 6LoWPAN Architecture 3

3 Architecture and Design 7

3.1 Moving Node Software 7

3.2 Stationary Node Software 9

3.3 Web Application 10

3.3.1 Server 10

3.3.2 Client 16

4 Implementation **23**

4.1 Berkeley Low-Power IPv6 Stack (BLIP) 23

4.1.1 Sending 23

4.1.2 Receiving 30

4.2 ip-driver Software Architecture 32

4.3 IPBaseStation Software Architecture 38

4.4 Moving Node Software Implementation 40

4.4.1 Define Sensors 40

4.4.2 Sensor Readings Calculation 42

4.4.2.1 Light Sensors 42

4.4.2.2 Humidity and Temperature Sensor 43

4.4.2.3 Voltage Sensor 43

5 Experimental Design and Results **49**

5.1 Mobile Wireless Sensor Architecture 49

5.2 Experimental Testing Software Architecture 53

5.2.1 Packet Specification 53

5.2.2 Testing software architecture 56

5.2.2.1 Detecting Packet Route Topology Change 56

5.2.2.2 Measuring Packet Loss 61

5.3 What We Measured 64

5.3.1 Testing Results 66

5.3.1.1 One Moving Node Testing 66

5.3.1.2 Two Moving Nodes Testing 68

6 Summary and Conclusions **72**

6.1 Summary 72

6.2 Conclusions 73

6.3 Future Work 73

References **76**

A Moving Node Software **77**

A.1 Data Structures Used in UDP Packets in Testing 77

A.2 Main Program for Moving Node 78

B Stationary Node Software **82**

C Testing Software **85**

C.1 Java Monitor Program 85

C.2 Python Monitor Program 87

D WSNWeb Application **90**

D.1 Server Side Source Code 90

D.2 Client Side Source Code 96

Vita

List of Tables

3.1 Functions of each `Read` interface in the module `UDPMovingP`. 8
3.2 Methods of a `connection` object for the `WSNWeb` client. 18

4.1 TinyOS 2.1.1 interfaces and implementations for 6LoWPAN support in BLIP. 39
4.2 TelosB built-in sensors used in moving node application. 41

5.1 UDP payload structure. 55
5.2 Speed step velocities map for train engine 6404. 65
5.3 Speed step velocities map for train engine 1478. 65
5.4 Number of topology changes (NTC) and packet loss under different routing table update periods, and different train velocities in one moving node testing. 66
5.5 Number of topology changes (NTC) and packet loss under different routing table update periods, and different train velocities in two moving node testing. 69

List of Figures

2.1 IEEE 802.15.4 frame, from [20]. 4
2.2 Data encapsulation. 4
2.3 Initial fragment of Adapt header [16]. 4
2.4 Noninitial fragment of Adapt header, from [16]. 5
2.5 Dual stack, integrating 6LoWPAN with IPv6, from [20]. 5
2.6 6LoWPAN addressing, based on the description in RFC4944 [16]. . . 6

3.1 Wiring of interfaces for the `UDPMoving` application (adapted from Graphviz version). 9
3.2 Architecture diagram of the WSNWeb web application. 11
3.3 Boot sequence of the Apache Tomcat server with embedding Jetty server, the details of the Tomcat boot sequence is shown in [12]. . . . 12

3.4 Subset of Jetty library *.jar files used to implement a WebSocket communication. . . . 13
3.6 The file `web.xml` defines Listener and startup web page. . . . 13
3.7 Sequence diagram of starting the thread in the constructor of WsnWebSocketHandler. . . . 15
3.8 Defining a WebSocket object in JavaScript. . . . 17
3.9 Processing of route topology and sensor reading message received at the client. . . . 19
3.5 Class diagram of the WSNWeb web application. . . . 21
3.10 A screen shot of the WSNWeb application. . . . 22

4.1 UDP interface sending packets to an IP address. . . . 24
4.2 Sequence diagram of `sendTask()`. . . . 25
4.3 Sequence diagram of getNextFrag(). . . . 26
4.4 Picture of the edge router with connected USB Raven and gateway. . 27
4.5 A complete UDP packet captured in Wireshark. . . . 28
4.6 The first fragment of the UDP packet in Figure 4.5. . . . 29
4.7 The subsequent fragment of the UDP packet in Figure 4.5. . . . 29
4.8 The implementation of `receive()` event. . . . 31
4.9 How BLIP handles first fragment when a node receives it. . . . 32
4.10 Dual stack, integrating 6LoWPAN with IPv6, adapted from [20]. . . . 33
4.11 Wireless sensor network architecture in ITB214. . . . 34
4.19 `read_and_process` reads and processes up to one 6LoWPAN packet. . 36
4.21 Sequence diagram of reassembly when serial port receives 6LoWPAN packets. . . . 38
4.22 Transmission of packets from radio to uart. . . . 40
4.23 Transmission of packets from uart to radio. . . . 40
4.12 Sequence diagram of setting up TCP socket and tunnel interface in the `ip-driver` program. . . . 44
4.13 Sequence diagram of opening a serial port and initializing routing in ip-driver. . . . 45
4.14 Sequence diagram of tunneling process of receiving data. . . . 46
4.15 Sequence diagram of `tun_input()`. . . . 46
4.16 Sequence diagram of `serial_input()`. . . . 47
4.17 Definition of data type `serial_source_t`. . . . 48
4.18 Definition of data type `packed_lowmsg_t`. . . . 48
4.20 Definition of data type `reconstruct_t`. . . . 48

5.1 Model train running on the wireless sensor network testbed in ITB214. 49
5.2 A Telosb node is carried on the model train. . . . 50
5.3 General view of train track in ITB214. . . . 51
5.4 Model railroad track in ITB214. . . . 52
5.5 Model railroad track corner. . . . 53
5.6 Packet structure of UDP over IPv6. . . . 54
5.7 141 bytes Payload structure. . . . 54

5.8 Packet specification of field sensor_reading. 56
5.9 `Monitor.java` monitors the topology changes of the network, and `UDPMonitor.py` stores all the received UDP packets and calculates packet loss. 56
5.10 A sample response to the `routes` command. 57
5.11 Message passing between `Monitor` and `ip-driver`. 60
5.12 A portion of a sample `route_info.log` file. 60
5.13 Measuring packet loss. 64
5.14 Sample output from the `UDPMonitor` program. Line 1262 shows the 141 byte payload for a received UDP packet. The first two bytes (1, 158) are the high and low parts of the sequence number, which is $1 \times 256 + 158 = 414$. 64
5.15 Packet loss vs. train velocity for one moving node testing. 67
5.16 Number of topology changes vs. train velocity for one moving node testing. 68
5.17 Packet loss vs. train velocity for two moving nodes testing. 70
5.18 Total packet loss vs. train velocity for two moving nodes testing. . . . 70
5.19 Number of topology changes vs. train velocity for two moving node testing. 71

List of Abbreviations

AHM	Airplane Health Management
BLIP	Berkeley Low-Power IP stack
CRC	Cyclic Redundancy Check
EUI	Extended Unique Identifier
GHz	GigaHertz
HC	Header Compression
HTML5	HyperText Markup Language 5
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Messasge Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IR	Infrared
JMRI	Java Model Railroad Interface
js	JavaScript
kB	kilo Bytes
LQI	Link Quality Identifier
MAC	Media Access Control
MHz	MegaHertz
MTU	Maximum Transmission Unit
NTC	Number of Topology Changes
PAN	Personal Area Network
PL	Packet Loss
RAM	Random Access Memory
R/T	*Received/Transmitted*
RTUP	Routing Table Update Period
SI	International System of Units
TCP	Transmission Control Protocol
TOS	TinyOS
tun	tunnel
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
UNB	University of New Brunswick
USB	Universal Serial Bus
ws	WebSocket
WSN	Wireless Sensor Network

XML	Extensible Markup Language
6LoWPAN	IPv6 over Low power Wireless Personal Area Networks

Chapter 1

Introduction

1.1 Wireless Sensor Networks

Wireless sensor networks (WSNs) have numerous applications including environmental monitoring [13], industrial monitoring [21], and security [18]. WSNs make it possible to monitor dangerous places such as nuclear environments or extremely high temperature environments where humans cannot be present. In environmental monitoring, the WSN is deployed over an area where some environmental parameters are monitored. In industrial monitoring, the WSN nodes can be deployed for machinery condition-based maintenance. AHM (Airplane Health Management) [18] is an example of security. In general, WSNs consist of spatially distributed sensor nodes which can monitor environmental or physical conditions such as humidity, pressure, temperature, vibration, light intensity and so on. Sometimes the positions of nodes are fixed, but nodes can be mobile. An example of a mobile sensor node is Zebranet, a system for wildlife tracking that focuses on monitoring zebras [15]. In order to communicate with and control WSNs, WSN applications cannot be built in isolation; they have to be connected to an existing network such as the Internet.

1.2 Thesis Objectives

In some cases sensor nodes are static, while in some cases sensor nodes are moving. The objective of this thesis is to explore a protocol for mobile sensor network communication. This thesis will attempt to answer the following questions:

1. How well do 6LoWPAN libraries integrate with IP networks running IPv6?
2. Can an IPv6 protocol based on 6LoWPAN accommodate moving nodes?
3. If the IPv6 based protocol can accommodate moving nodes, how quickly can it adapt to a dynamic environment?

Chapter 2

IPv6 in Wireless Sensor Networks

Internet Protocol Version 6 (IPv6) is the designated successor of IPv4 as the network protocol for the Internet. There are 2^{32} IPv4 addresses, but all of those have now been allocated by the Internet Assignment Numbering Authority (IANA) as of Feb. 2011 [19]. To overcome this lack of addresses, IPv6 expands the IP address space from 32 to 128 bits. IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN [20]) makes it possible to connect WSN nodes to the Internet; every node can have an IP address. An implementation of 6LoWPAN on TinyOS is called BLIP [3] (the Berkeley Low-Power IP stack) has been already carried out by the University of California, Berkeley.

2.1 6LoWPAN Architecture

6LoWPAN allows IPv6 packets to be sent to or received from IEEE 802.15.4-based networks [7]. The IEEE 802.15.4 frame size is 127 bytes. Figure 2.1 shows an IEEE 802.15.4 frame. As we can see, there is space for only 31 bytes of data. The size of an IPv6 frame is at least 1280 bytes [17]. How can we fit an IPv6 frame into an IEEE 802.15.4 frame? The solution is fragmentation. The data is first encapsulated by an

IPv6 header, and then encapsulated by an Adapt header, which contains fragment information, and lastly by a MAC header. The encapsulation is shown in Figure 2.2 for UDP packets. TCP packets have a longer header (20, 24, or 28 bytes). Figure 2.3 shows the first fragment of the Adapt header, Figure 2.4 shows a noninitial fragment of the Adapt header.

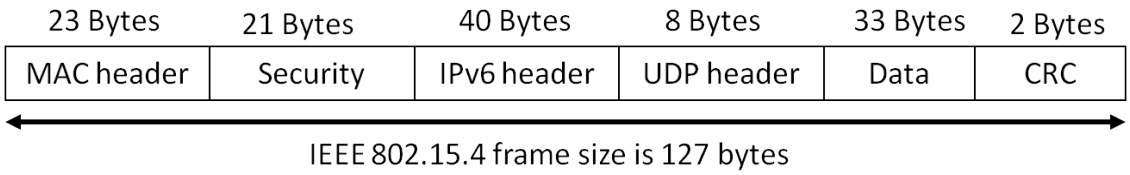


Figure 2.1: IEEE 802.15.4 frame, from [20].

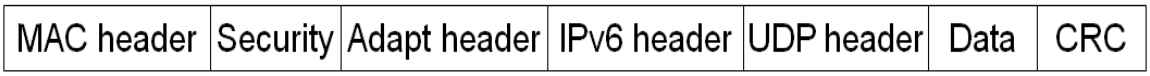


Figure 2.2: Data encapsulation.

Figure 2.3: Initial fragment of Adapt header [16].

The layers of 6LoWPAN networks are shown in Figure 2.5. The packet is adapted to 6LoWPAN packet in the 6LoWPAN adaptation layer of 6LowPAN networks. In the gateway (usually a router), the frames are adapted to IEEE 802.15.4 frame. After converting to IEEE 802.15.4 frame, the packet can be transmited into high speed networks (such as Internet).

The space left for data is already very small. If the Adapt header is introduced, the space will become smaller, i.e. 28 or 29 bytes. To guarantee efficiency, the IPv6 header must be compressed.

1	1	1	0	0	Datagram_size (11 bits)	Datagram_tag (16 bits)
Datagram_offset (8 bits)						

Figure 2.4: Noninitial fragment of Adapt header, from [16].

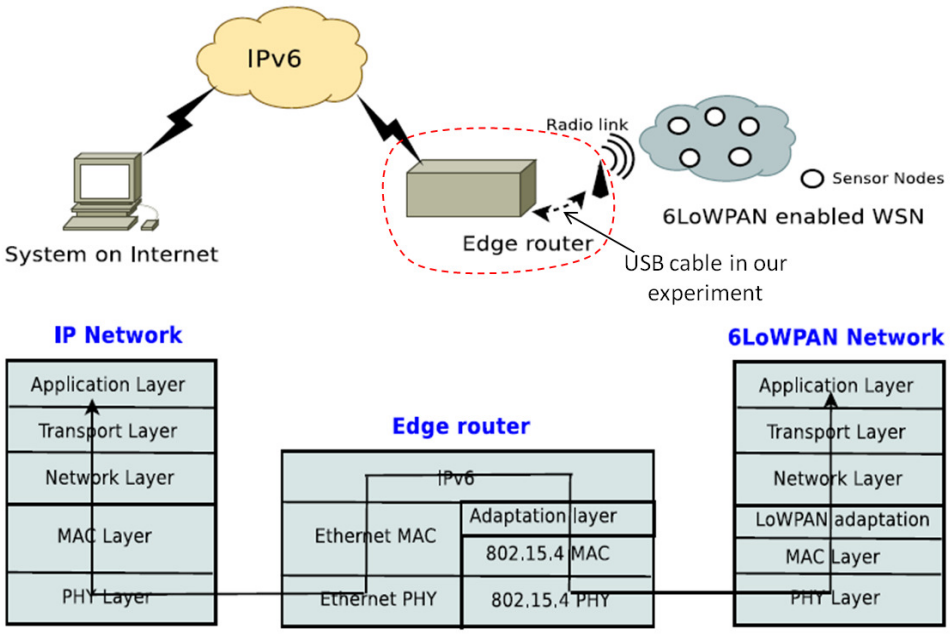


Figure 2.5: Dual stack, integrating 6LoWPAN with IPv6, from [20].

Hui et al. [14] define an encoding format for 6LoWPAN called LOWPAN_IPHC. LOWPAN_IPHC assumes the following will be the common case for 6LoWPAN:

1. Version is 6.
2. Traffic label class and flow label are both zero.
3. Payload length can be inferred from low layers.
4. Hop limit will be set to a well-known value by the source.
5. Addresses assigned to 6LoWPAN prefix will be formed using the link-local prefix or a small set of routable prefixes assigned to the entire 6LoWPAN.
6. Addresses assigned to 6LoWPAN interfaces are derived directly from either the 64-bit extended or 16-bit short IEEE 802.15.4 MAC addresses [14].

This results in the IPv6 40 bytes header being reduced to 2 bytes.

An IPv6 address consists of two parts: the first part is a 64 bit prefix, the second part is a 64 bit interface ID. The prefix is formed using the link-local address `fe80::/64` (this notation means the first 16 bits are `fe80`, followed by 48 zeroes) [16]. The interface ID is derived from the MAC address. 16 bits `fffe` follow the first 24 bits of the MAC address, which are then followed by the remaining 24 bits of the MAC address. IEEE 802.15.4 has two forms of MAC address, 64-bit EUI-64 (Extended Unique Identifier-64) and 16-bit short addresses. If the MAC address uses a 64-bit long address, the 6LoWPAN interface ID directly uses the MAC address as its interface ID. If the MAC address uses a 16-bit short address, the 6LoWPAN interface ID first uses the 16-bit short address and PAN ID (Personal Area Network Identifier) to generate a 48-bit pseudo MAC address. Then the interface ID is derived from this pseudo MAC address as discussed above. The 6LoWPAN address is shown in Figure 2.6.

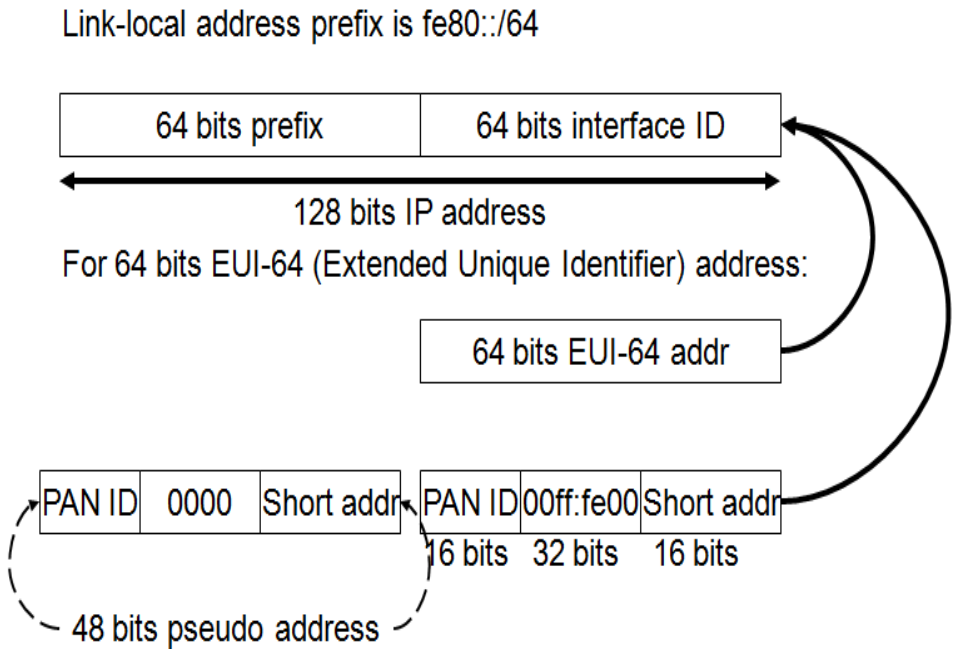


Figure 2.6: 6LoWPAN addressing, based on the description in RFC4944 [16].

Chapter 3

Architecture and Design

3.1 Moving Node Software

The moving node application software is comprised of two files: `UDPMovingP.nc` and `UDPMovingC.nc`. `UDPMovingC.nc` is the configuration file of the application which defines the components used in this application and interface wiring, and `UDPMovingP.nc` defines the module UDPMovingP of the application. Module `UDPMovingP` uses interface `UDP` to send UDP packets, and interface `Timer<TMilli>` to control the UDP sending period. Interface `SplitControl` is used to start radio transmission. The `UDPMovingP` application module uses five interfaces of the interface `Read` using the names `ReadTSR`, `ReadPAR`, `ReadExtTemp`, `ReadHum`, and `ReadVolt`. The functions of each `Read` interface are shown in Table 3.1. Four instances of the `Statistics` interface are used to constitute a UDP payload. Complete details of implementing the sensors are discussed in Section 4.4.1. Lastly, `UDPMovingP` also uses `Leds` and `Boot` interfaces.

Table 3.1: Functions of each `Read` interface in the module `UDPMovingP`.

Name	Function
ReadTSR	Read IR sensor readings
ReadPAR	Read visible sensor readings
ReadExtTemp	Read temperature sensor readings
ReadHum	Read humidity sensor readings
ReadVol	Read voltage sensor readings

TinyOS developers use the Nesdoc tool to displays the structure and composition of nesc applications. For example, to show the structure of an TinyOS application that uses BLIP, we issue the following command:

```
make telosb blip docs
```

With the `docs` argument, the `make` command automatically generates a `/doc/nesdoc` subdirectory. The `nesdoc` subdirectory contains `.dot` files used by Graphviz [5] to generate a component graph. The component graph generated for our `UDPMovingP` module is shown in Figure 3.1. Single line boxes represent modules, double line boxes represent configurations, and edges are labeled with interface names. Dashed double line boxes indicate generic components.

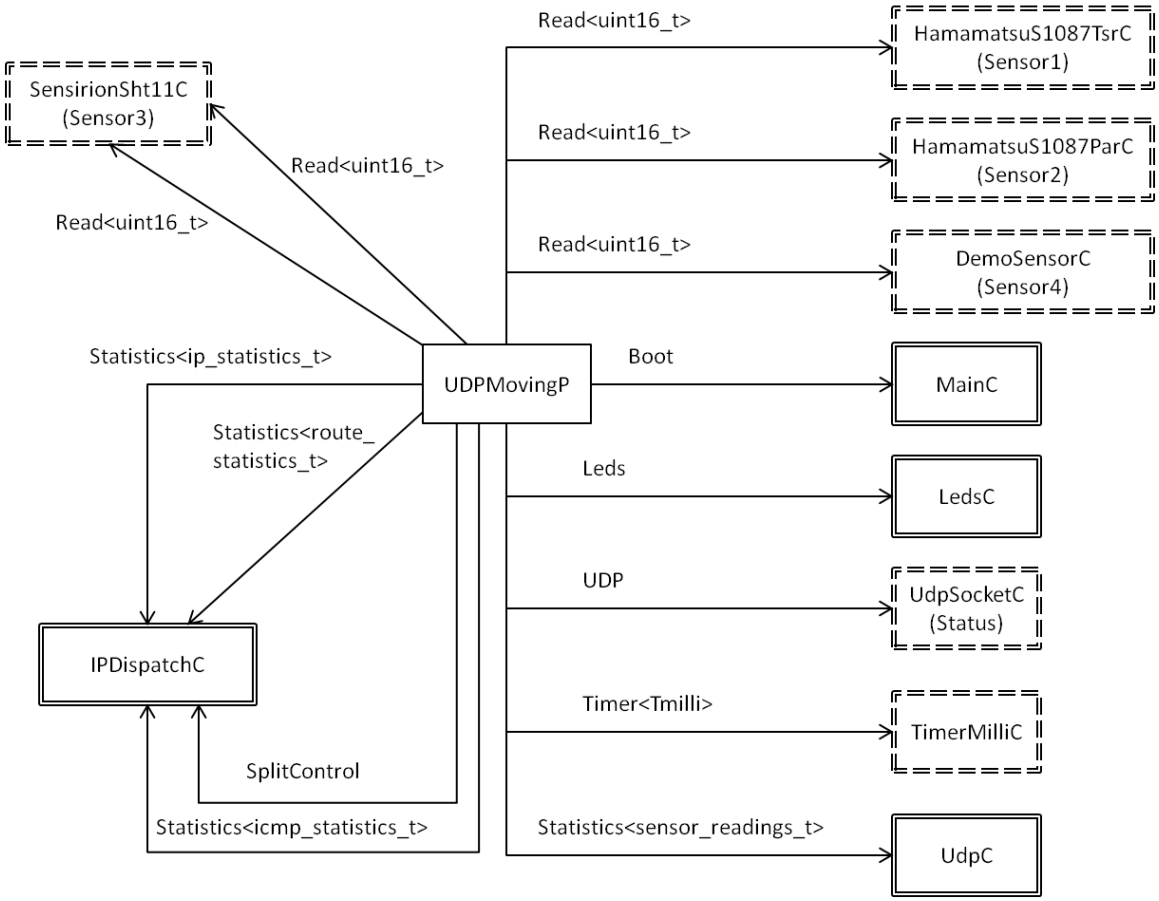


Figure 3.1: Wiring of interfaces for the `UDPMoving` application (adapted from Graphviz version).

The moving node application sends UDP packets from port 7001 to destination `fec0::64` port 7000. The timer is fired every ten seconds. Every time the timer is fired, the application reads sensors and fills the payload of the UDP packet, and sends out the UDP packet.

3.2 Stationary Node Software

The stationary node application software consists of two files: `UDPStationaryP.nc` and

`UDPStationary.nc`. `UDPStationary` is adpated from `UDPMoving`, the difference is

that

`UDPStationary` does not read sensors and send any UDP packets when the timer is fired, but it does transmit 6LoWPAN packets.

3.3 Web Application

We wrote a Java-based web application called WSNWeb that monitors the topology of the wireless sensor network. The WSNWeb application implements an embedded Jetty server [4] comprised of two Java programs called

`WsnServerServletContextListener.class` and

`WsnWebSocketHandler.class` as shown in Figure 3.2. The Listener and Handler manage HTTP requests and WebSocket requests, respectively. The server side code creates a TCP socket to `ip-driver` through port 6106, and gets new wireless sensor network topology when the topology changes. The client side code displays new routes information in text form and draws network topology using the HTML5 Canvas element [6]. The WSNWeb server and clients (web browsers) communicate via the Jetty WebSocket Handler component (see the next section) [9].

3.3.1 Server

Figure 3.3 shows the boot sequence of the Apache Tomcat [1] server with an embedded Jetty server. We use Jetty 8.1.8.v20121106, which is the latest stable version of the Jetty server. To use Jetty WebSocket, some Jetty libraries need to be imported into the Eclipse project, as illustrated in Figure 3.4. As shown in Figure 3.5, class `WsnWebSocketHandler` is a subclass of `WebSocketHandler`, which handles WebSocket requests. When this handler is registered in the Jetty Server instance, the client side JavaScript code creates a WebSocket connection to the server, in this ex-

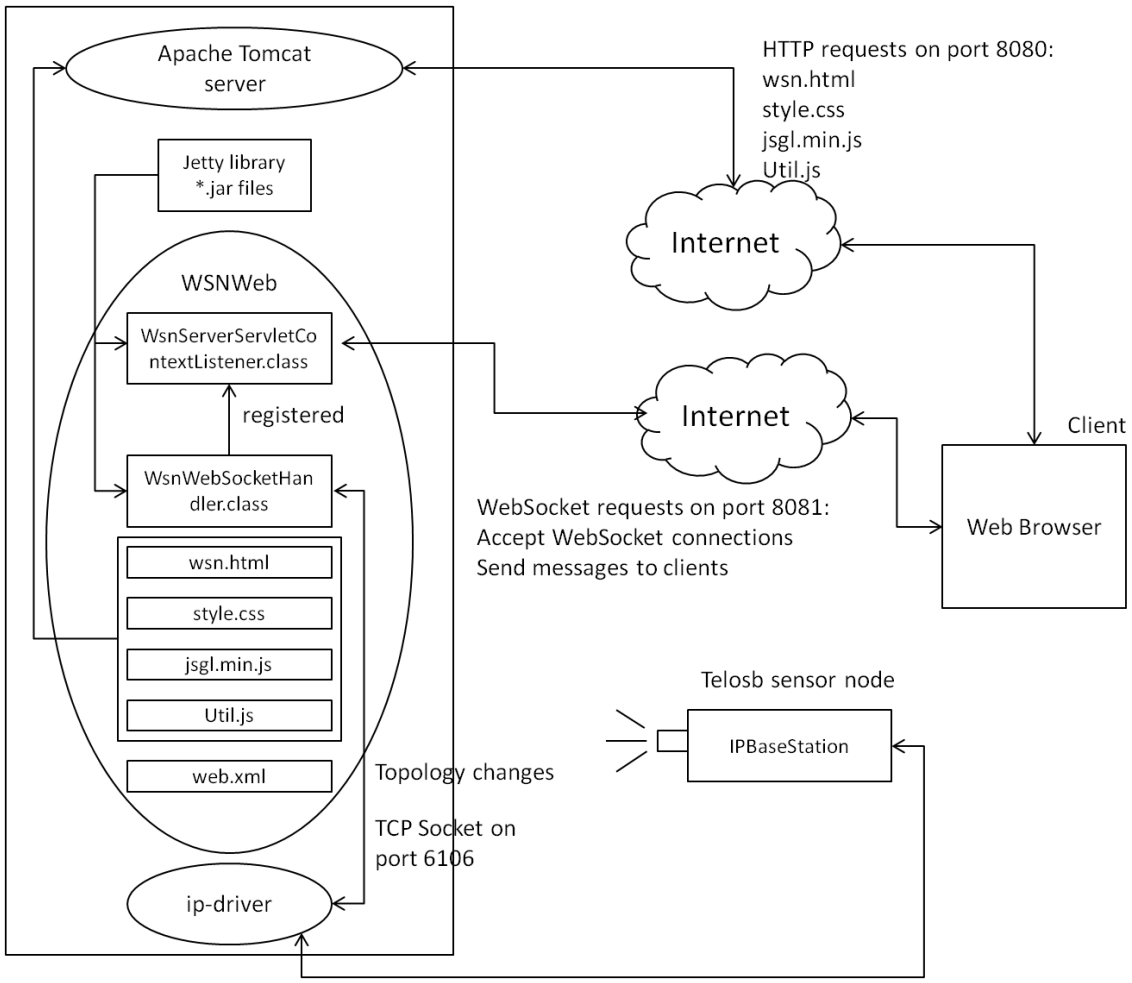


Figure 3.2: Architecture diagram of the WSNWeb web application.

ample, whose address is 131.202.243.6 on port 8081, with protocol `ws` (WebSocket). The public method `doWebSocketConnect()` returns a `WsnWebSocket` instance when a new WebSocket connection comes in.

As shown in Figure 3.5, `WebSocket` is a Java interface, and we need to create a class to implement it to instantiate a `WebSocket` object. `WebSocket` has some internal interfaces, and we chose one to implement. Text messages are used to transmit routes information in the WSNWeb application, so we implement a `WebSocket.OnTextMessage` interface inside class `WsnWebSocketHandler` using the following statement:

```
private class WsnWebSocket implements WebSocket.OnTextMessage
```

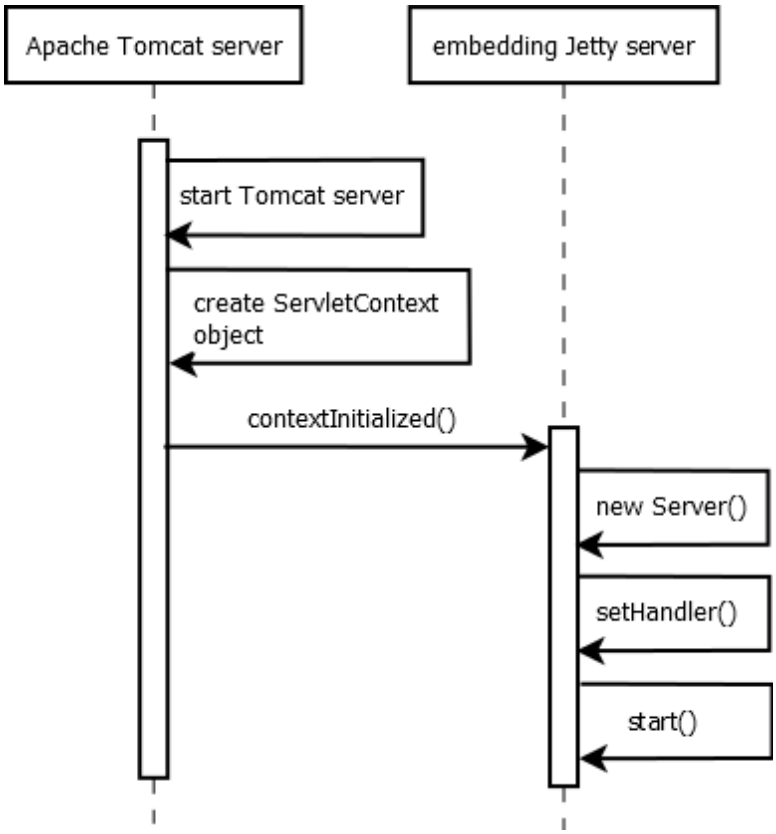


Figure 3.3: Boot sequence of the Apache Tomcat server with embedding Jetty server, the details of the Tomcat boot sequence is shown in [12].

There are three methods we need to implement in class `WsnWebSocket`, including `onOpen()`, `onMessage()`,and `onClose()`.

`onOpen()` is called when a client WebSocket opens a connection, and we store the new opened connection in the set of `WsnWebSocket` items. `onMessage()` is called when a message comes in from a client, and `onClose()` is called when a client WebSocket closes, and we remove the WsnWebSocket object from the set of `WsnWebSocket`.

The WSNWeb web application relies on a file called `web.xml` located at `WSNWeb/WebContent/WEB-INF/web.xml`. The file `web.xml` is shown in Figure 3.6, file `wsn.html` is defined to be invoked on startup, and is invoked when a client opens the WSNWeb application.

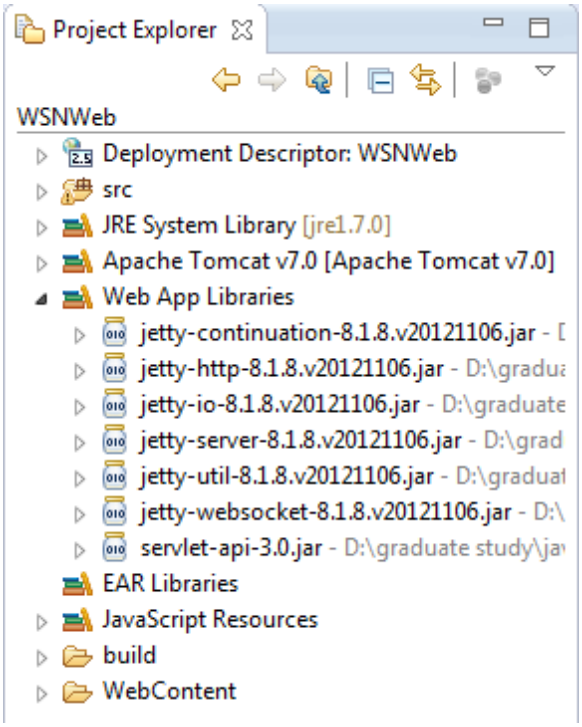


Figure 3.4: Subset of Jetty library *.jar files used to implement a WebSocket communication.

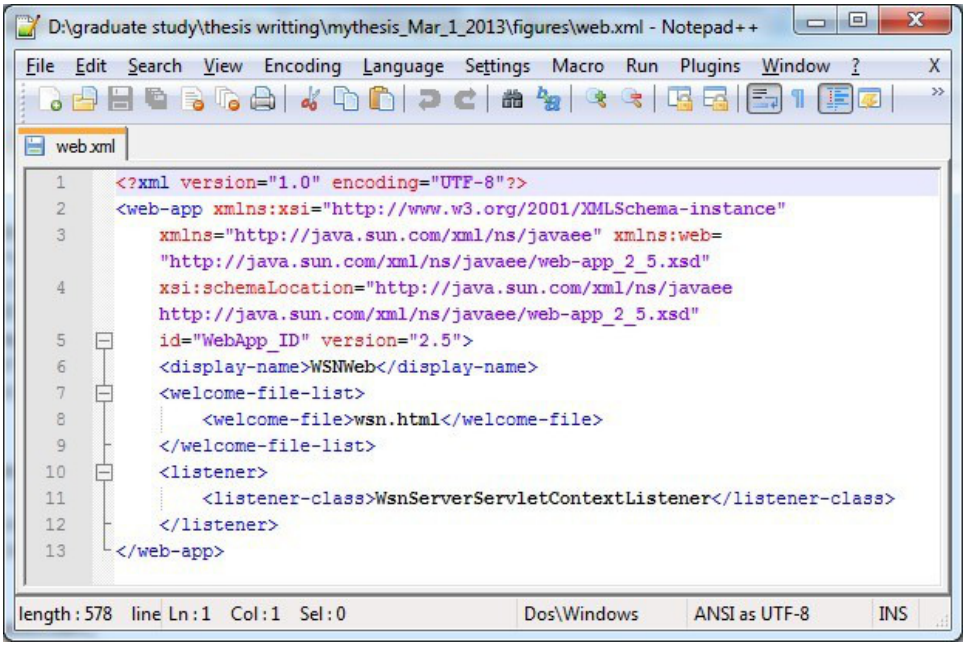


Figure 3.6: The file `web.xml` defines Listener and startup web page.

`WsnServerServletContextListener` is set to be the class that responds to

ServletContext event such as initialized and destroyed.

Class WsnServerServletContextListener implements interface javax.servlet.ServletContextListener in the file WsnServerServletContextListener.java, as follows:

```
public class WsnServerServletContextListener implements ServletContextListener
```

The ServletContextListener interface is responsible for receiving notification events about ServletContext lifecycle changes. When the web application initialization process is starting, contextInitialized() is called. When the ServletContext is about to be shut down, contextDestroyed() is called. The Jetty server instance is created in contextInitialized() (see Figure 3.3), and an instance of WsnWebSocketHandler is registered to the Jetty server instance. All the HTTP requests are processed on port 8080 using Apache Tomcat Server (e.g. displaying HTML files), and all the WebSocket requests are processed on port 8081 using the embedded Jetty server (e.g. sending topology change messages to client web browsers).

The thread that creates a TCP socket to ip-driver connection, and monitors the network topology is created in the constructor of class WsnWebSocketHandler. The thread starts when an instance of WsnWebSocketHandler is instantiated in WsnServerServletContextListener. Figure 3.7 shows the code structure of starting the thread. The thread keeps getting the network topology from the created TCP socket. If there is a route change, the thread will send the new route to all clients connected to the server through WebSocket.

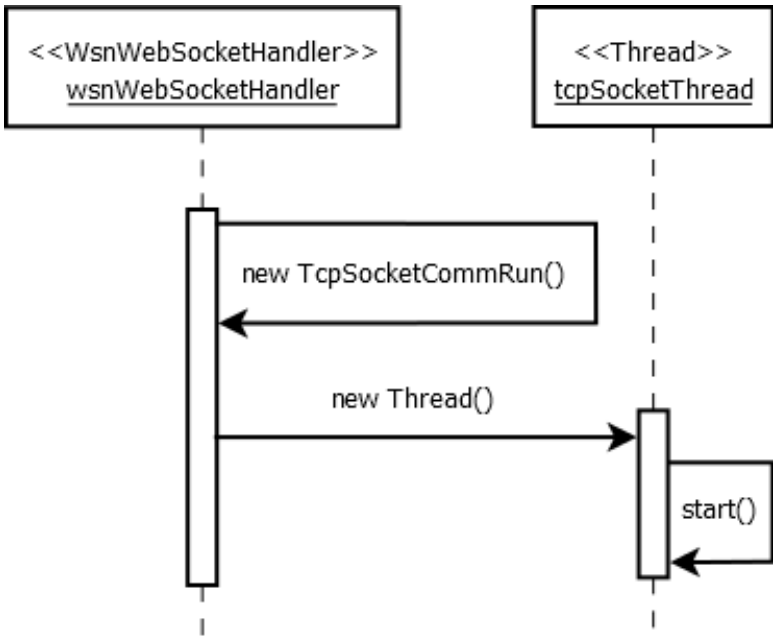


Figure 3.7: Sequence diagram of starting the thread in the constructor of WsnWebSocketHandler.

The destination of UDP packets is `fec0::64` on port 7000. Another thread called `udpSocketThread` that creates a UDP socket to `fec0::64` on port 7000 is also created in the constructor of class `WsnWebSocketHandler`. `udpSocketThread` reads sensor readings contained in the UDP packets, and sends sensor readings to clients. As described in Section 5.2.1, the payload of each UDP packet used in the testing is of size 141 bytes, so we use a 141-byte buffer to receive UDP packets in `udpSocketThread`. The sender address is the first 3rd and 4th bytes, light is the 14th and 15th bytes, temperature is the 18th and 19th bytes, humidity is the 20th and 21st bytes, and voltage is the 22nd and 23rd bytes. Each byte in a UDP packet is an 8-bit unsigned integer, but in Java the `byte` data type is an 8-bit signed two's complement integer. We need to convert each 8-bit unsigned integer to its two's complement representation to get the original sensor reading value. A positive integer's two's complement representation is the integer itself, and an integer in Java is four bytes long. We first cast `byte` to Java data type `int`, and then perform a bit-wise and (`&`) with `0xff` to reset the first 24 bits of the resultant integer. The complete Java code for converting

sensor readings is given in Appendix D.1.

As an example, the binary representation of unsigned integer 156 is 10011100. If we interpret these bits as a two's complement integer, the decimal value is -100. The Java statement `(int)recvByte[14] & 0xff` converts the unsigned integer at byte offset 14 from the `recvByte` array into a 4-byte integer. After conversion, thread `udpSocketThread` calculates the actual environmental values based on the sensor readings and broadcasts the actual environmental values to all clients. The details of the calculation of actual environmental values are discussed in Section 4.4.2.

3.3.2 Client

The client side code is written in JavaScript. File `Util.js` contains a `HashMap` class, and file `jsgl.min.js` contains the code of the JSGL library. JSGL is an open source, browser independent 2D vector graphics library for JavaScript [8]. Animation such as moving nodes and network topology updating is created using JSGL. The JavaScript inside of the `wsn.html` body manages the WebSocket connection and message processing. Our WSNWeb server creates an embedded Jetty server on port 8081, and an instance of `WsnWebSocketHandler` is registered to the Jetty server instance. A WebSocket connection is available at `ws://131.202.243.6:8081`, where `ws://` stands for the WebSocket protocol. Clients process all the HTTP requests through
`http://131.202.243.6:8080/WSNWeb`, and the JavaScript code inside of `wsn.html` connects to the Jetty server and receives messages through
`ws://131.202.243.6:8081`. A WebSocket object is created in the client by the definition shown in Figure 3.8.

```
var connection = {
    join : function() {
        var location = "ws://131.202.243.6:8081/";
        this._ws = new WebSocket(location);
        this._ws.onopen = this._onopen;
        this._ws.onmessage = this._onmessage;
        this._ws.onclose = this._onclose;
        this._ws.onerror = this._onerror;
    },
    // code for other functions
```

Figure 3.8: Defining a WebSocket object in JavaScript.

The WebSocket connection request is sent using TCP. Once the server recieves a WebSocket connection request from a client, the server side class `WsnServerServletContextListener` calls `doWebSocketConnect()`, which returns a new `WsnWebSocket` object to handle WebSocket events. At this point, the client WebSocket has opened a connection, and `onOpen()` of `WsnWebSocket` is called. A `WebSocket.Connection` interface is passed to the newly created `WsnWebSocket` object via `onOpen()`, and
`WebSocket.Connection` has methods for sending messages (e.g. `sendMessage()`). The receiving of messages is handled by method `onMessage()`.

In the JavaScript code inside of `wsn.html`, an object called `connection` handles WebSocket connections, message processing, and data display on the web page. Table 3.2 shows the methods of a `connection` object. Figure 3.9 illustrates how both message types (i.e. topology changes and sensor readings) are received at the client.

Table 3.2: Methods of a `connection` object for the `WSNWeb` client.

Name	Function
join	Create WebSocket connection to the server
onopen	implementation of WebSocket `onopen` method, sends "connected" to the server when successfully connected
onmessage	implementation of WebSocket `onmessage` method, interacts to incoming messages
onclose	implementation of WebSocket `onclose` method, close the established WebSocket connection
onerror	implementation of WebSocket `onerror` method, display error messages
send	sends messages to the server

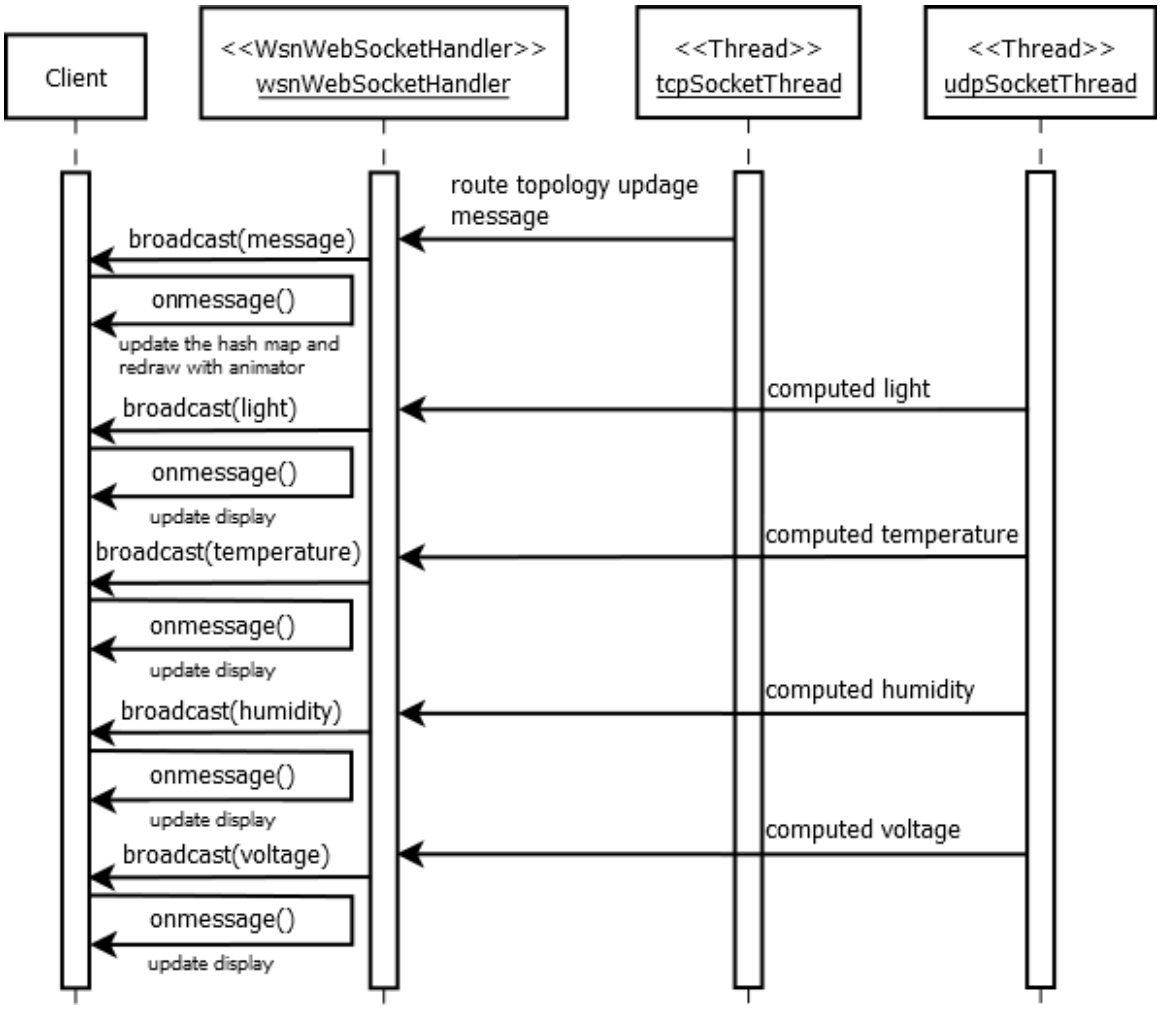


Figure 3.9: Processing of route topology and sensor reading message received at the client.

As we can see from Table 3.2, method `onmessage` handles incoming route topology change and sensor reading messages. When the client receives a message from the server, `onmessage` checks whether it is a route topology update message or a sensor reading message. If the message is a route topology update message (begins with `"New"`, e.g. `"New route for 0x1:  [1.0]0x6 [1.0]0x4 [1.0]0x64"`), `onmessage` stores the new route in the hash map on the client side, and updates the dynamic route topology on the web page. The route of each node (both moving node and stationary node) is represented by a polyline, whose start point is the moving node and end point is the edge router (0x64). After storing the new route in the hash

map, `onmessage` updates the intermediate points of each node according to the route stored in the hash map. For example, if the route of node `0x1` stored in the hash map is `0x6, 0x4, 0x64` after updating, the points of node `0x1`'s polyline become points that represent the positions of nodes `0x1`, `0x6`, `0x4`, and `0x64`. An instance of `jsgl.util.Animator` called `lineAnimator` keeps updating the positions of the polyline's points since the nodes are moving, and line segments are drawn between consecutive points. The moving node positions are updated using a simulated linear velocity corresponding to approximately 0.45 m/s for outside moving node and 0.68 m/s for inside moving node.

If the message is a sensor reading message (identified by keywords starting with "temp", "humi", "ligh", "volt"), `onmessage` parses the string to extract the message type (temperature, humidity, voltage, or light), and displays the sensor reading at the proper position on the web page. `onmessage` displays the sensor reading by dynamically creating a JavaScript `span` element and updating its `innerHTML` attribute. The complete client side code is given in Appendix D.2.

Figure 3.10 shows an example screen shot of the WSNWeb application. Field 1 shows the new routes information, Field 2 shows the topology of the wireless sensor network, and Field 3 shows the updating sensor readings. New routes and new sensor information is continuously updated by the server through the WebSocket.

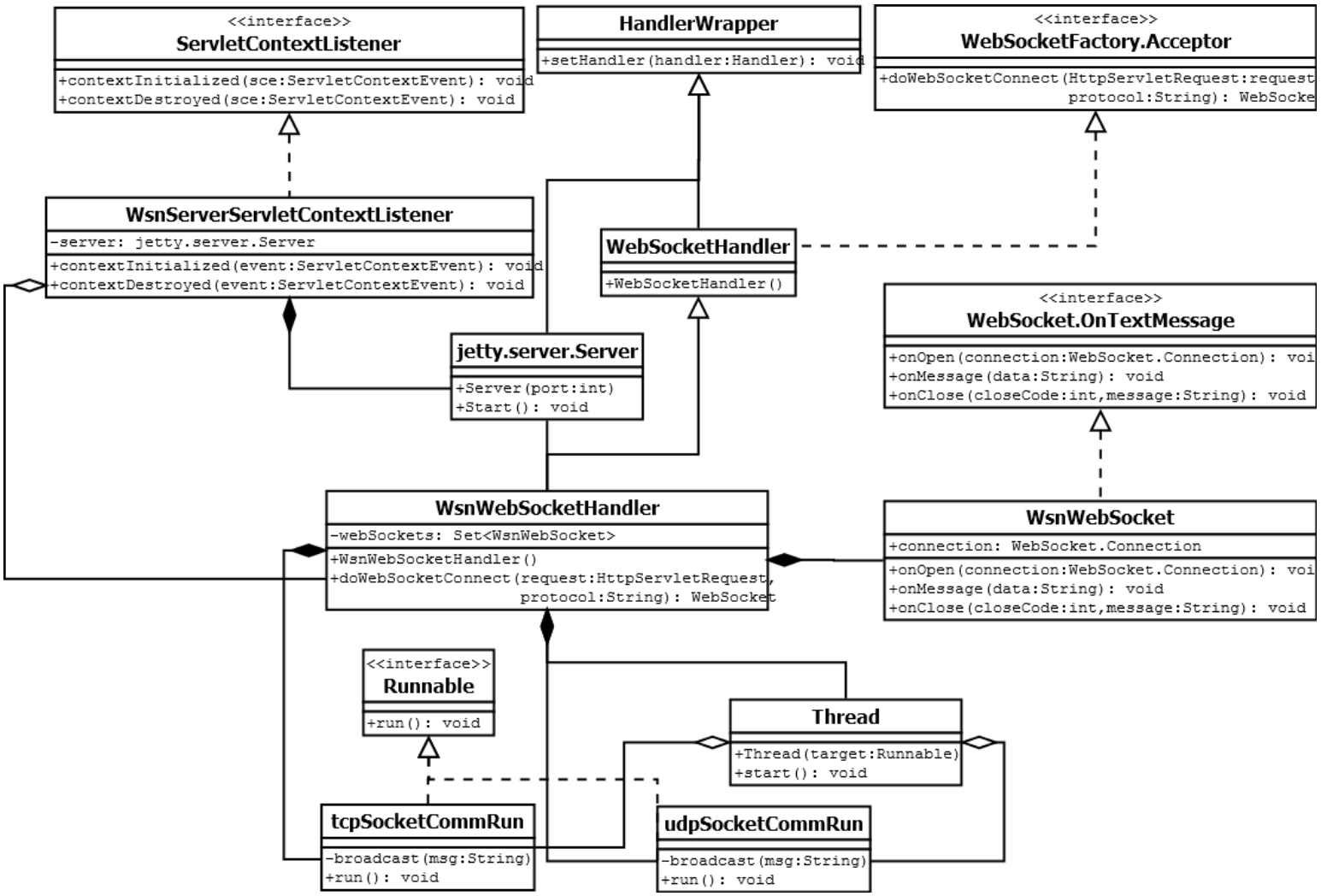


Figure 3.5: Class diagram of the WSNWeb web application.

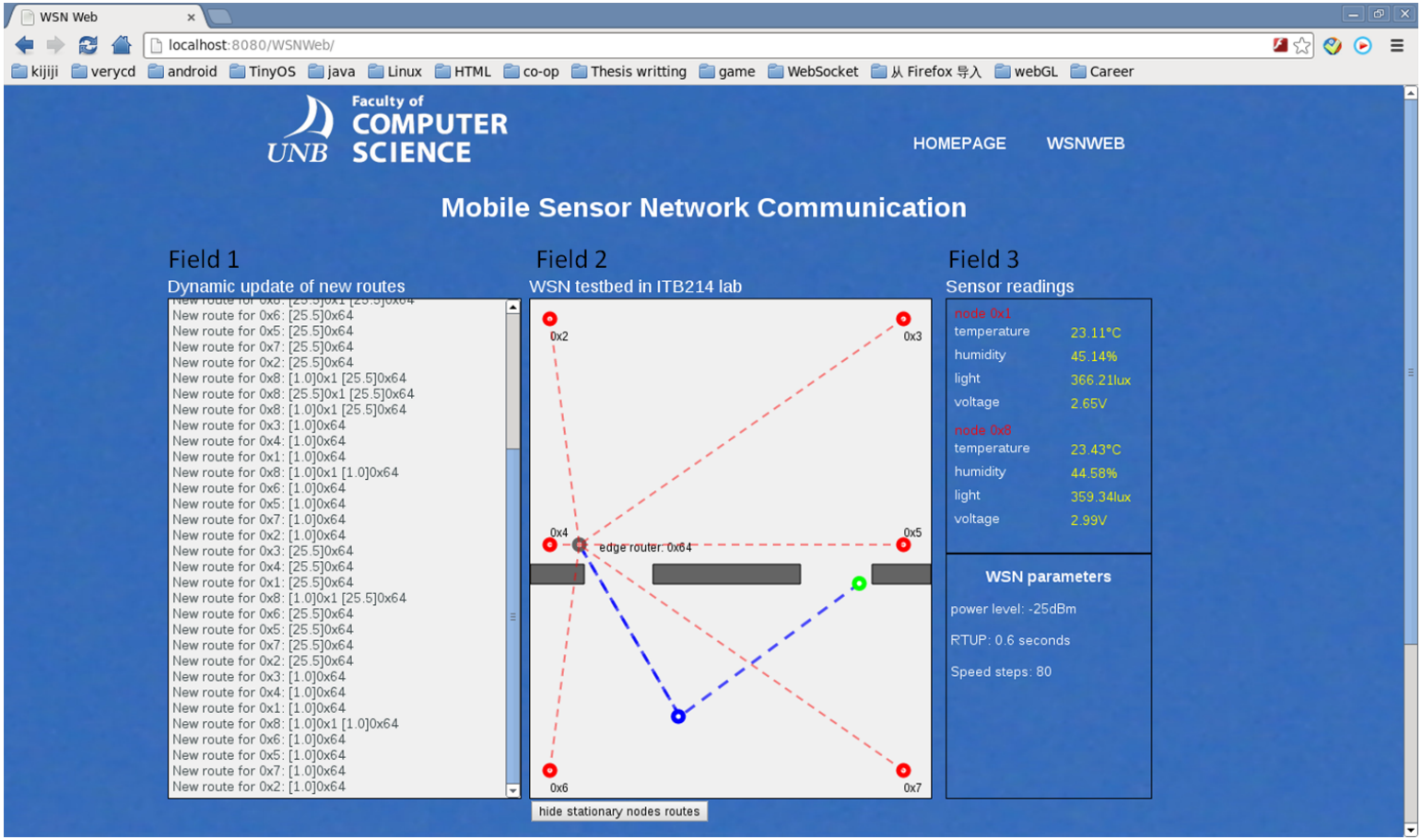


Figure 3.10: A screen shot of the WSNWeb application.

Chapter 4

Implementation

4.1 Berkeley Low-Power IPv6 Stack (BLIP)

BLIP is an IP stack that `UDPStationary` and `UDPMobile` use, so the sending and receiving of packets are based on the BLIP stack. TinyOS 2.1.1 provides some interfaces for using BLIP such as `UDP`. BLIP is being imported when those interfaces are added to applications.

4.1.1 Sending

BLIP provides a transport layer interface called `UDP` that uses IPv6. The TinyOS application `UDPMobile` uses the `UDP` interface, which has a command `sendto(struct sockaddr_in6 *dest, void *payload, uint16_t len)`. The `sendto()` command is used to send a payload to the IPv6 socket address indicated. Figure 4.1 shows the sequence diagram of sending packets using this `UDP` interface. The packets go into the network layer when `IP.Send()` is invoked, and `IP.Send()` calls `IP.bareSend()`. The actual packets that go into the air are 6LoWPAN packets (with an IEEE 802.15.4 physical layout), so the IPv6 packets have to be

transformed to 6LoWPAN packets. The Fragment layer is responsible for splitting IPv6 packets into fragments which are 6LoWPAN packets. BLIP keeps enqueuing fragments as long as the next fragment's length is greater than 0, which means there is a next fragment. Finally, the fragments are sent using `sendTask()`, Figure 4.2 shows the sequence diagram of `sendTask()`.

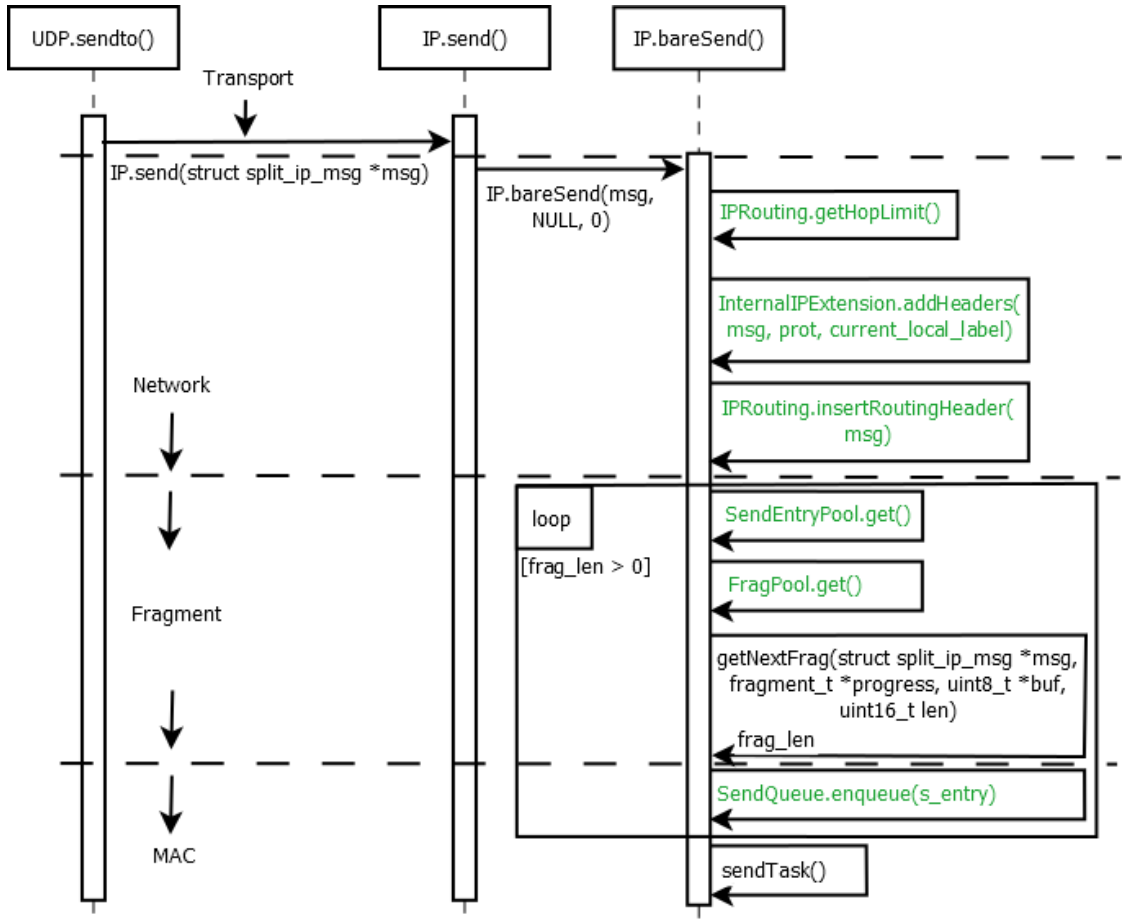


Figure 4.1: UDP interface sending packets to an IP address.

Figure 4.2: Sequence diagram of `sendTask()`.

Figure 4.3 is the sequence diagram for `getNextFrag()`. The `fragment_t` tracks progress of fragmentation. The fragment is a first fragment (FRAG1) when the offset equals 0; otherwise the fragment is a subsequent fragment (FRAGN). At the end of `getNextFrag()`, the fragment length in bytes is returned to `IP.bareSend`. `s_entry` is a variable of struct `send_entry_t`, which is the actual queue item, pointing to a fragment and the packet metadata.

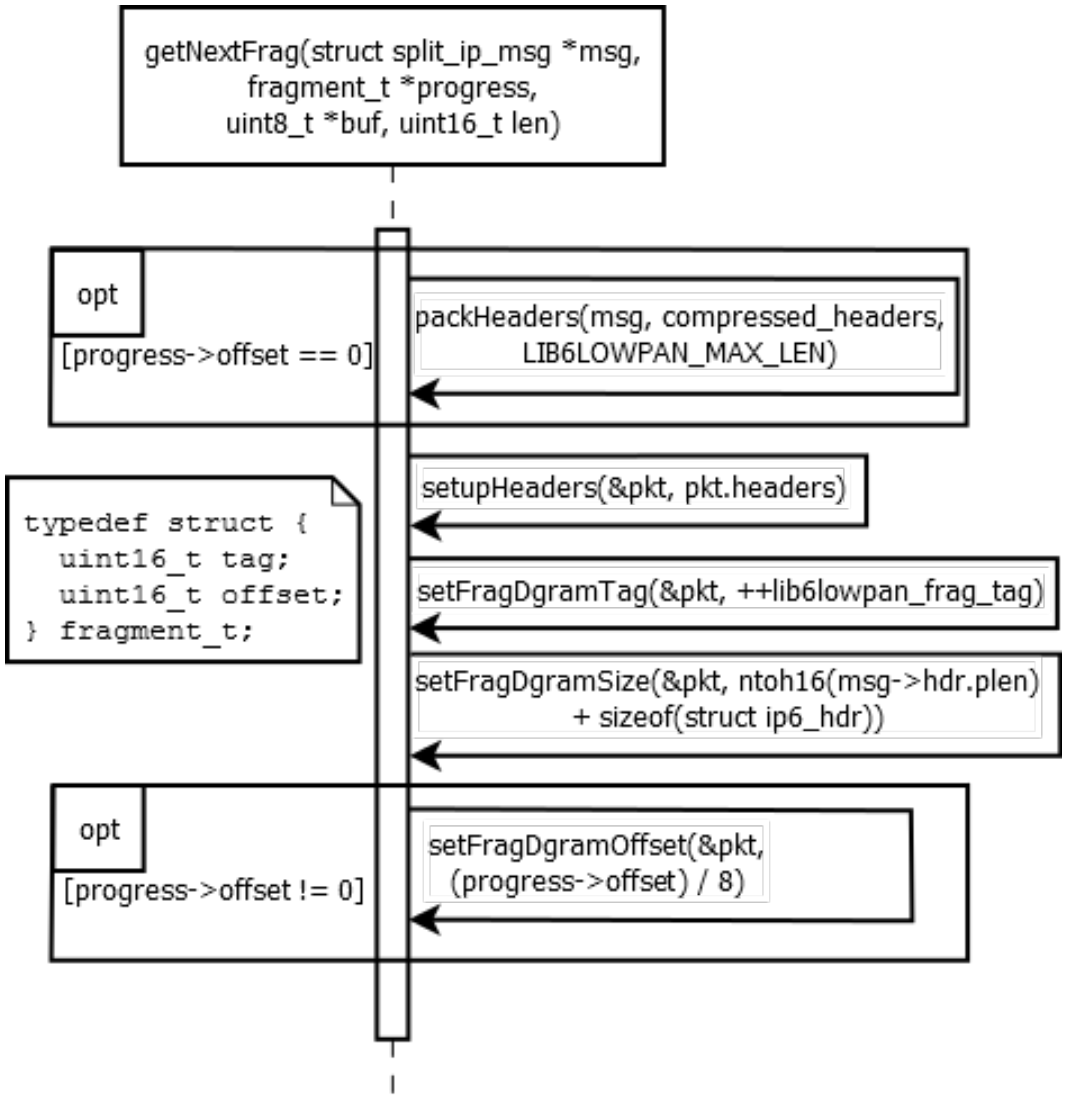


Figure 4.3: Sequence diagram of getNextFrag().

We use an example to illustrate how the moving node sends UDP packets to the edge router. A Raven USB sniffer (Atmel part number AVRRZUSBSTICK [2]) is used to sniff the 6LoWPAN packets in the air. A program called `15dot4` is running on the Raven USB sniffer, and the Raven USB sniffer is connected to the PC through a USB cable (see Figures 4.11 and 4.4). When plugging in the Raven USB stick, it comes up as an Ethernet interface (`eth1`). A sniffer control command line program is used to configure the Raven USB sniffer. All the Telosb nodes in the network are using channel 15, so we should set the channel of the Raven USB sniffer to 15.

Running the following command will set the channel the of Raven USB sniffer to 15 where `/dev/hidraw2` is the device name of the sniffer:

```
./Ravenusb -d /dev/hidraw2 -c 15
```

The device name is discovered by running the `dmesg` command, which shows messages describing devices as they are added and removed. Once configured on a Linux workstation, the Raven USB reads wirelessly transmitted IEEE 802.15.4 packets which can, in turn, be read using Wireshark. Wireshark is next to configured to capture the `tun0` and `eth1` interfaces. UDP packets are captured on interface `tun0` and 6LoWPAN packets are captured on interface `eth1`.

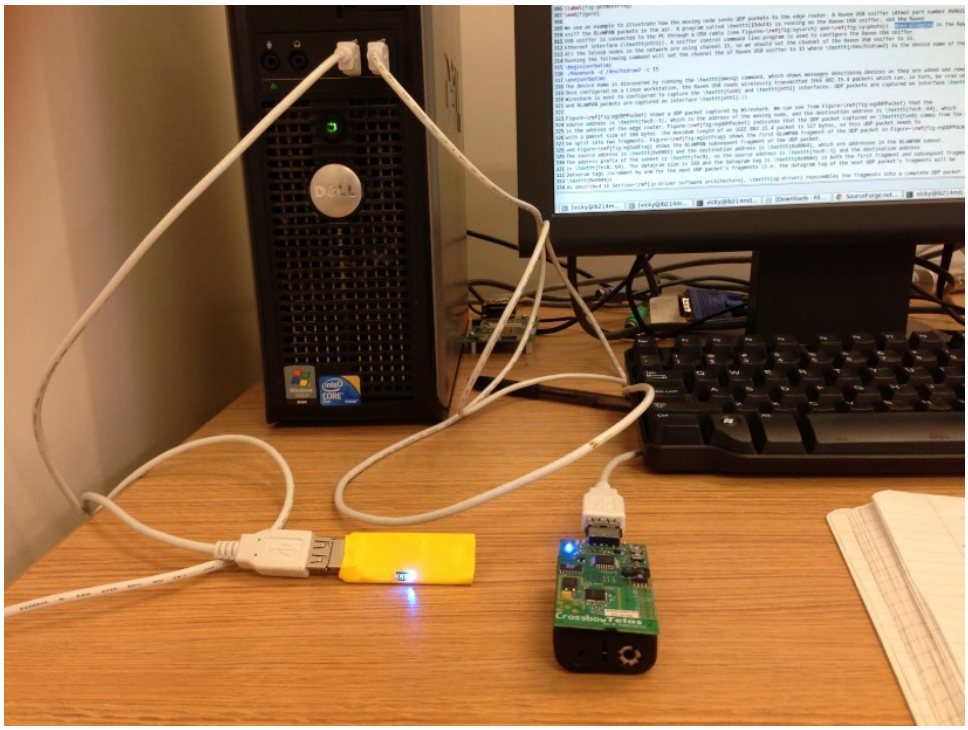


Figure 4.4: Picture of the edge router with connected USB Raven and gateway.

Figure 4.5 shows a UDP packet captured by Wireshark. We can see from Figure 4.5 that the source address is `fec0::1`, which is the address of the moving node, and the destination address is `fec0::64`, which is the address of the edge router. Figure 4.5

indicates that the UDP packet captured on `tun0` comes from the moving node, with a packet size of 189 bytes. The maximum length of an IEEE 802.15.4 packet is 127 bytes, so this UDP packet needs to be split into two fragments. Figure 4.6 shows the first 6LoWPAN fragment of the UDP packet in Figure 4.5, and Figure 4.7 shows the 6LoWPAN subsequent fragment of the UDP packet. The source address is `0x0001` and the destination address is `0x0064`, which are addresses in the 6LoWPAN subnet. The address prefix of the subnet is `fec0`, so the source address is `fec0::1` and the destination address is `fec0::64`. The datagram size is 189 and the datagram tag is `0x004d` in both the first fragment and subsequent fragment. Datagram tags increment by one for the next UDP packet's fragments (i.e. the datagram tag of the next UDP packet's fragments will be `0x004e`). As described in Section 4.2, `ip-driver` reassembles the fragments into a complete UDP packet when `ip-driver` receives them.

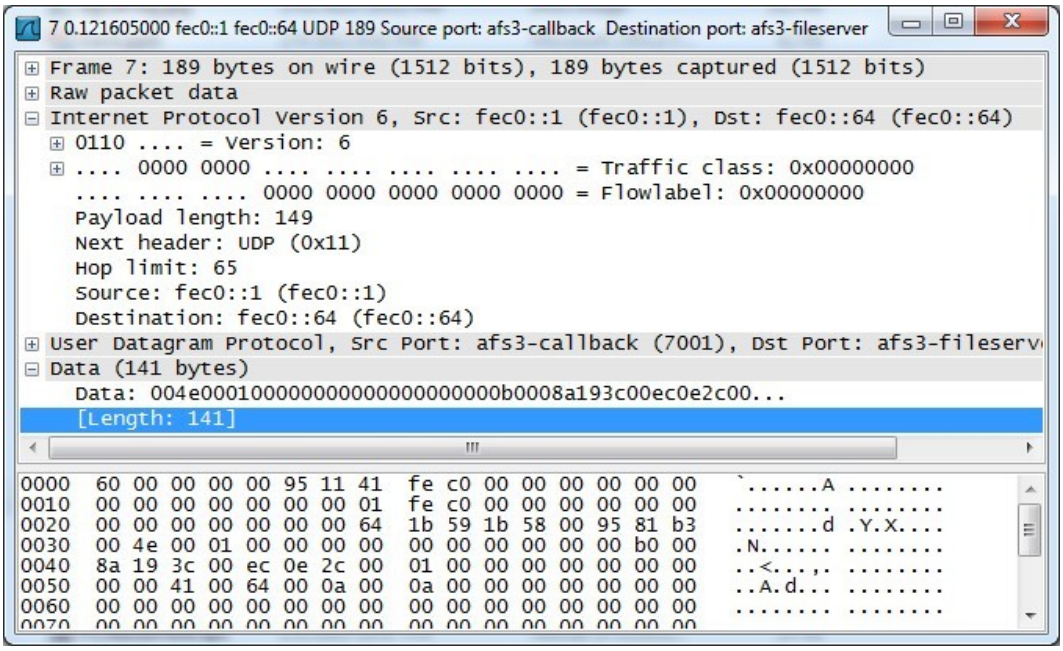


Figure 4.5: A complete UDP packet captured in Wireshark.

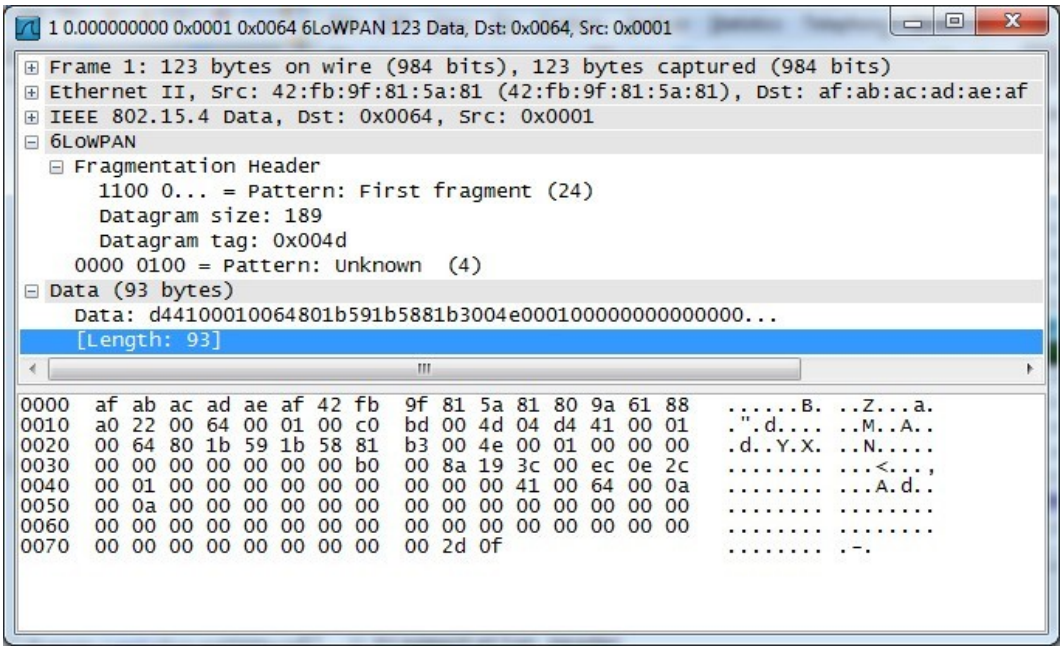


Figure 4.6: The first fragment of the UDP packet in Figure 4.5.

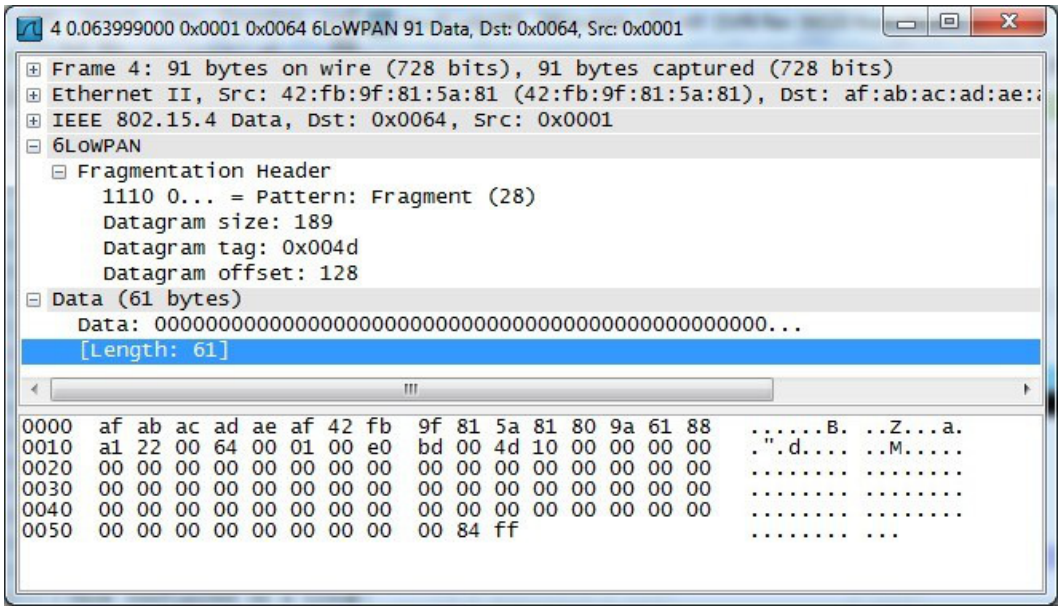


Figure 4.7: The subsequent fragment of the UDP packet in Figure 4.5.

4.1.2 Receiving

By implementing the event

`message_t* receive(message_t* msg, void* payload, uint8_t len)`

of `Receive` interface as shown in Figure 4.8, mobile nodes and stationary nodes are able to transmit and receive packets. In `IPDispatchP.nc`, interface `Receive` is defined as follows:

```
interface Receive as Ieee154Receive;
```

The processes of handling first fragment and handling subsequent fragment are different. Figure 4.9 shows how BLIP handles first fragment when it receives one. When a node (either mobile or stationary) receives a fragment, the node will check whether the fragment is for the node itself. If the fragment is for the node itself, BLIP will create a buffer and store the fragment in the buffer, otherwise the node will forward the fragment to the next hop.

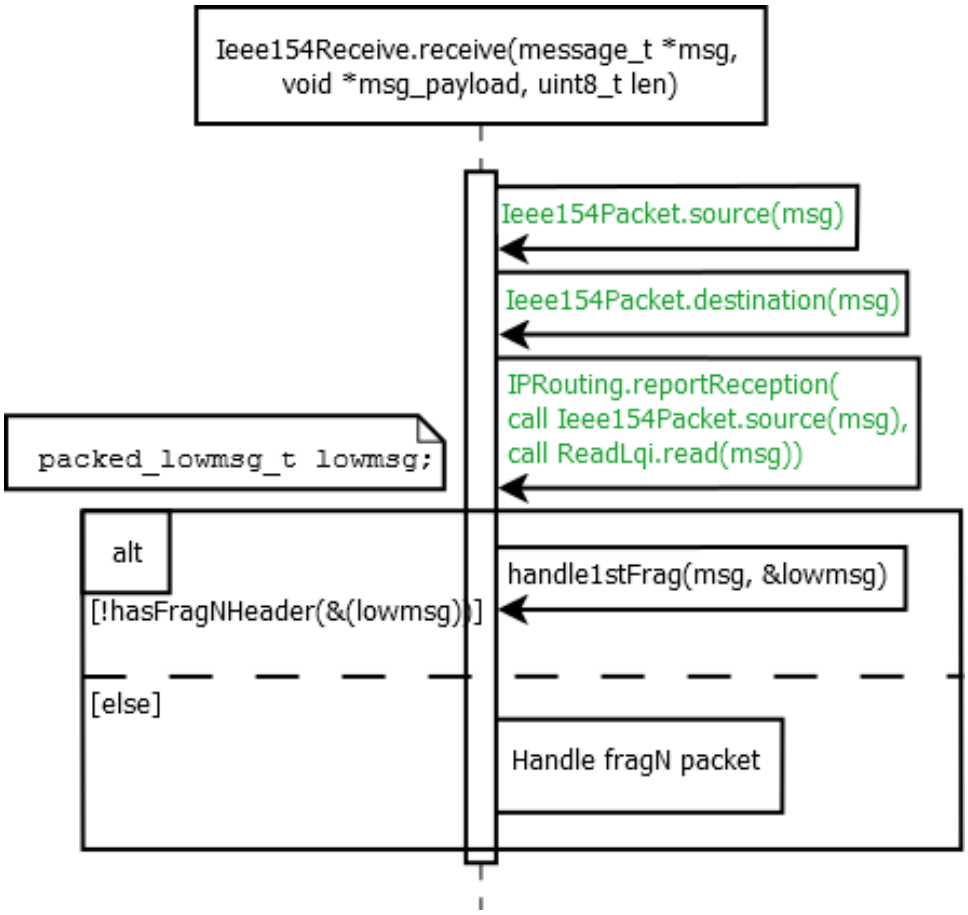


Figure 4.8: The implementation of `receive()` event.

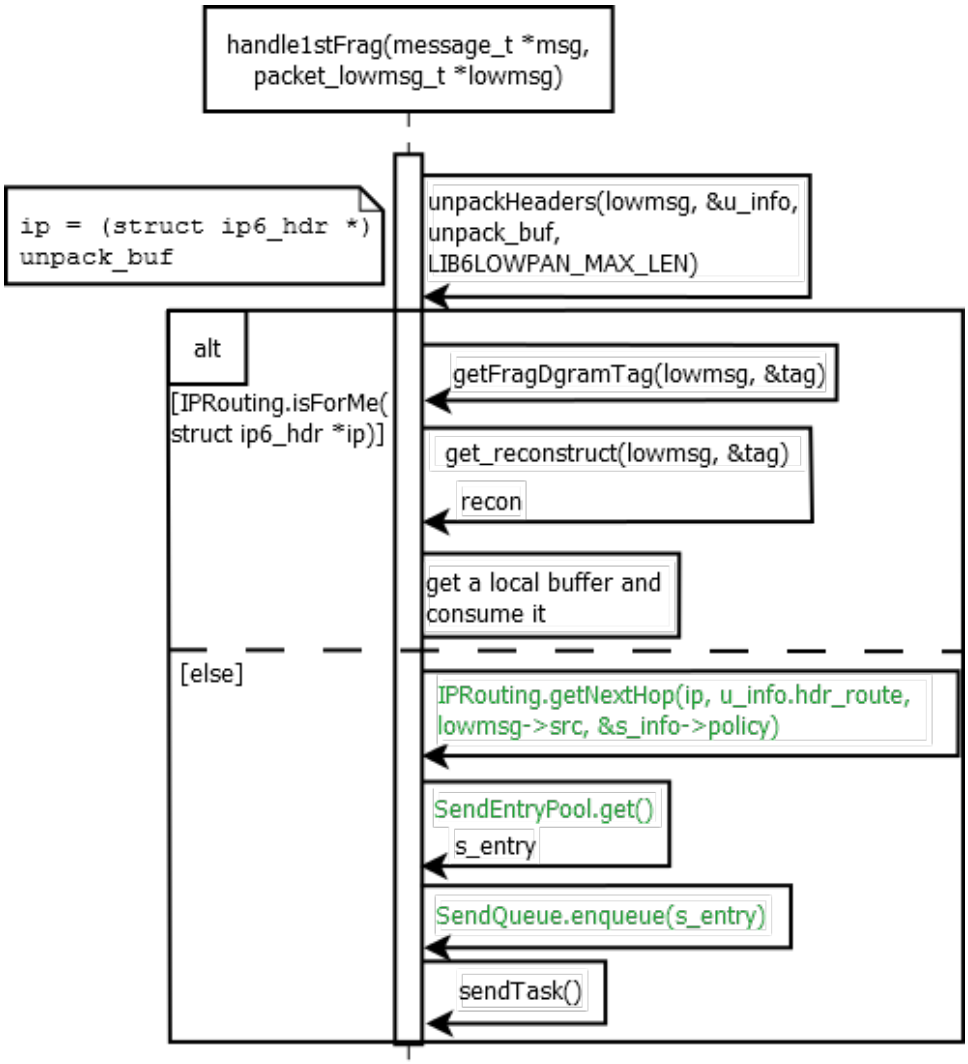


Figure 4.9: How BLIP handles first fragment when a node receives it.

4.2 ip-driver Software Architecture

Figure 4.10 shows how 6LoWAPN is integrated with IPv6 using a dual stack. The fragmentation (also called encoding) happens in the 6LoWPAN network, and the reassembly of packets (also called decoding) happens in the edge router. In our experiment, a TelosB node running a program called `IPBaseStation` combined with a Linux workstation running a program called `ip-driver` comprises the edge router, and the moving nodes running `UDPMobile` and the stationary nodes running

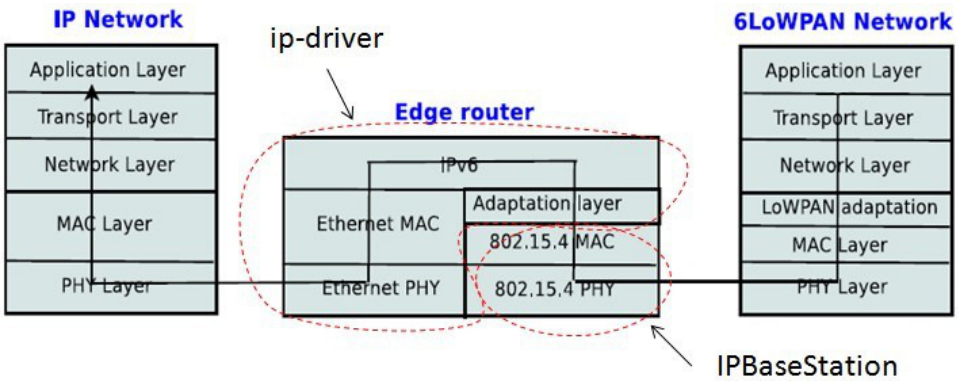


Figure 4.10: Dual stack, integrating 6LoWPAN with IPv6, adapted from [20].

`UDPStationary` constitute the 6LoWPAN network. Figure 4.11 shows the architecture of the wireless sensor network in the ITB214 wireless communication lab.

The `ip-driver` program first opens a TCP socket on port 6106 by calling `vty_init()`. A sequence diagram showing how the TCP socket and tunnel interface are organized is shown in Figure 4.12. Inside the function `vty_init()` as shown in Figure 4.12, some system calls are made. Functions labeled in red in Figure 4.12 and other Figures are Linux system calls. After creating the TCP socket, `ip-driver` creates a `tun0` network interface.

Since the TelosB node running `IPBaseStation` is connected to a PC through a USB cable, the 6LoWPAN packets are received and transmitted to the PC through a physical serial port. In `ip-driver`, a serial port is opened by calling function `open_serial_source()`. The `argv[optind]` in main is `/dev/ttyUSB0`, and `argv[optind + 1]` in main is `telosb` which is the string representation of baud rate 115200. `routing_init()` is called after serial port setup is done. In `routing_init()`, `nw_init()` sets up the network state data structures, and `nl_init()` starts a netlink session to the kernel. The serial port is initialized by typing the following command:

```
> sudo $TOSROOT/support/sdk/c/blip/driver/ip-driver /dev/ttyUSB0 telosb
```

A sequence diagram illustrating the opening of a serial port and initializing routing

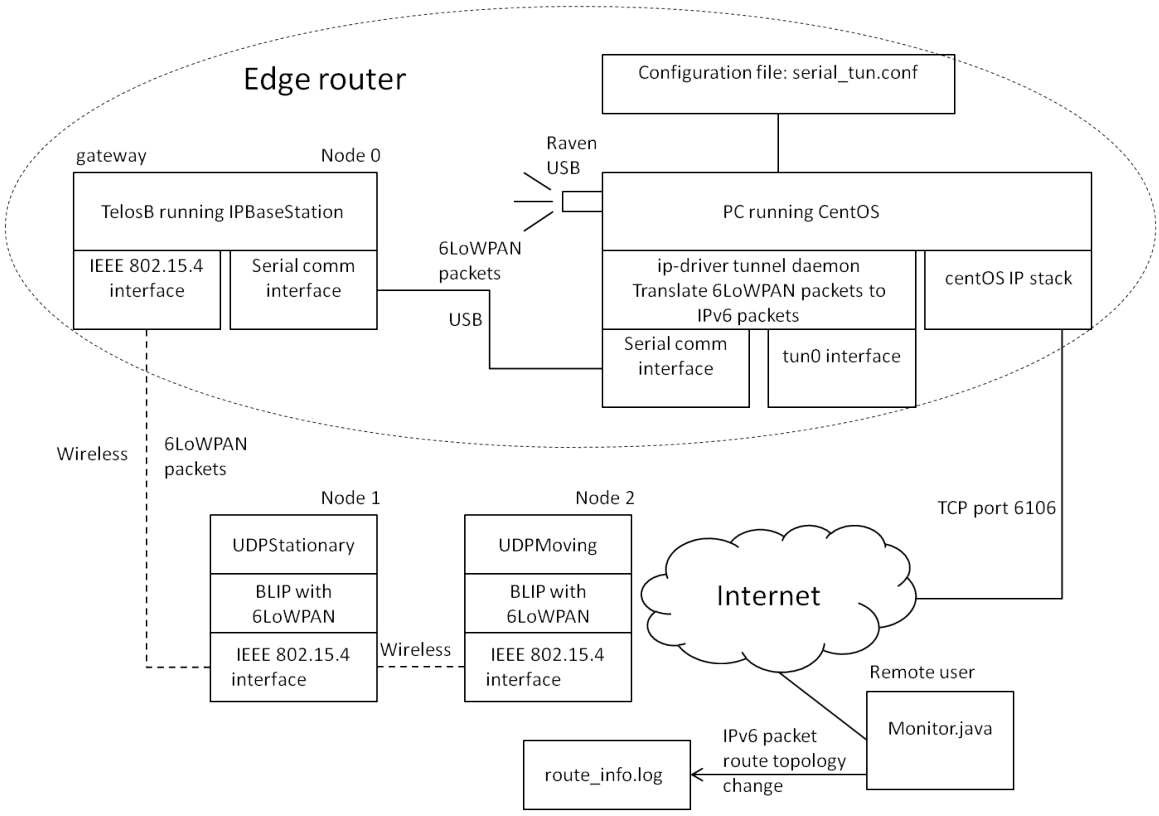


Figure 4.11: Wireless sensor network architecture in ITB214.

is shown in Figure 4.13.

At this point `ip-driver` can start tunneling. The tunneling process is handled in function `serial_tunnel`. The sequence diagram shown in Figure 4.14 illustrates the tunneling process of receiving data. In `serial_tunnel`, two functions which handle receiving are being called. The first one is `tun_input()` (see Figure 4.15), and the second one is `serial_input()` (see Figure 4.16). The loop encompassing the function `tun_input` and `serial_input` (in Figure 4.14) continues to read data from the tunnel and serial interface until no more data is available.

The data received from the serial port are 6LoWPAN packets. Linux system call `read()` attempts to read up to `n` bytes from file descriptor `src->fd` into the buffer starting at `buffer`. `src` is of type `serial_source_t` which is defined in Figure 4.17:

cnt in Figure 4.16 is the number of bytes read, and function serial_read returns the number of bytes read in variable n to read_byte. Function read_and_process reads and processes up to one 6LoWPAN packet as shown in Figure 4.19, it uses an infinite loop to keeps reading serial port, the loop terminates when no data is available on serial port, which means at the end of one 6LoWPAN packet. Function read_serial_packet() reads the serial source src, if a packet is available, read_serial_packet() will return it. Because the value of parameter non_blocking is TRUE, read_serial_packet() does not wait for one when no packet is available. read_serial_packet() returns the received 6LoWPAN packet to its caller function serial_input() ,variable ser_src now points to the read packet from serial port. As shown in Figure 4.21, pkt is a variable of struct packed_lowmsg_t, which represents the received 6LoWPAN packet. The structure of a packed_lowmsg_t is shown in Figure 4.18: In function serial_input, the header and the payload of the received 6LoWPAN packet are assigned to the corresponding attribute of pkt by the following code block:

```
1 IEEE154_header_t *mac_hdr = (IEEE154_header_t *)(ser_data
      + 1);
2 pkt.data = ser_data + 1 + sizeof(IEEE154_header_t);
3 pkt.src = mac_hdr->src;
4 pkt.dst = mac_hdr->dest;
```

In 6LoWPAN packet, a dispatch byte is before the header [16], so one byte is added for the computation of the header and the payload position.

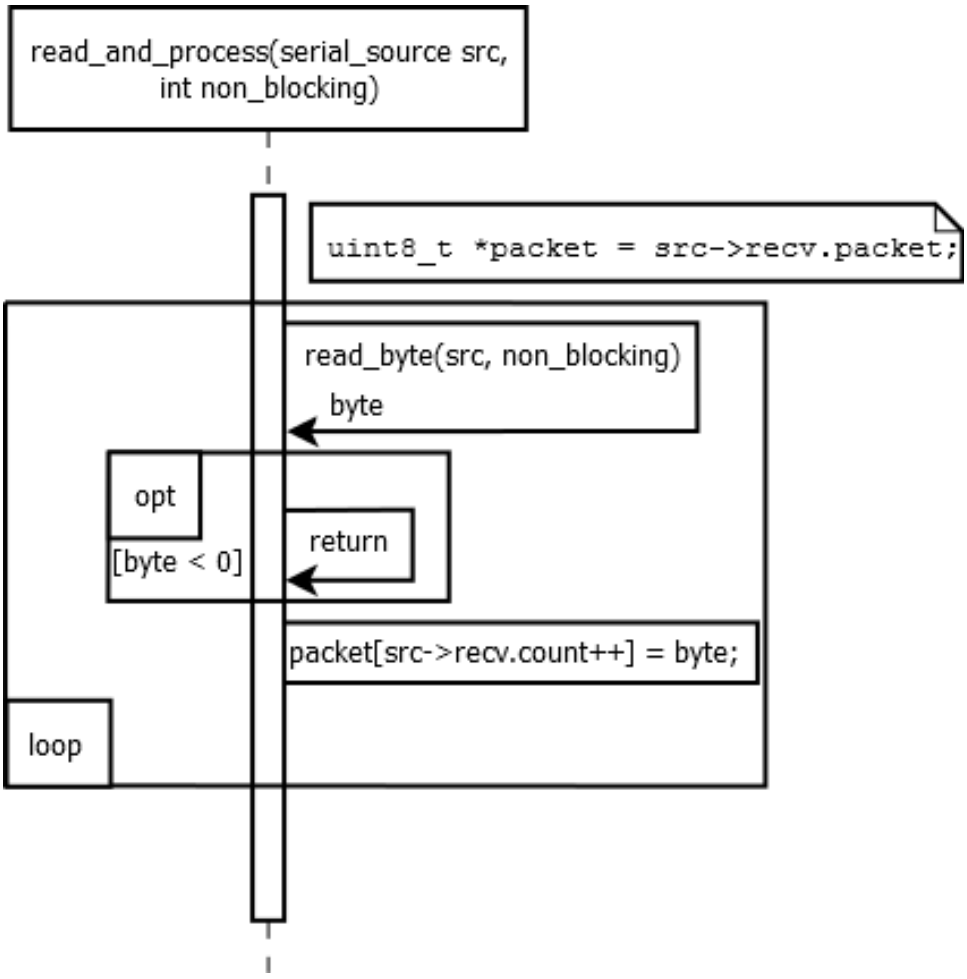


Figure 4.19: `read_and_process` reads and processes up to one 6LoWPAN packet.

Reassambly of 6LoWPAN packets happens inside `serial_input()`. After a 6LoWPAN packet is read from the serial port (see the `read(src->fd, buffer, n)` call in Figure 4.16), function `serial_input()` reassembles the 6LoWPAN packets it has received into an IPv6 packet (see Figure 4.21).

Function `getHeaderBitmap` returns a bitmap indicating which LoWPAN header is present in the message pointed to by `pkt`. Function `getReassembly()` is called if the 6LoWPAN header is either FRAG1 header or FRAGN header, `unpackHeaders()` will unpack all headers, including IP headers and any compressed transport headers. `msg` is of type `split_ip_msg`, and `msg->hdr` is the buffer to unpack the headers into. `reconstruct_t` is a struct that includes the reassembly information, as shown in

Figure 4.20: `buf` is the memory location of reconstructed IPv6 packet, whose data type is `split_ip_msg`. `ip-driver` creates an array to store reconstructed packets:

```
1 reconstruct_t reconstructions [N_RECONSTRUCTIONS];
```

The value of `N_RECONSTRUCTIONS` is 10. Function `getReassembly` takes a 6LoWPAN packet as its input argument, and returns a pointer which points to one of the elements in `reconstructions`. For the first fragment of an IPv6 packet, function `getReassembly` first gets its 6LoWPAN packet's tag and size, and then returns an available entry from `reconstructions` for the IPv6 packet reassembly. The value returned from `getReassembly` points to the next available `reconstruct_t` array entry in the `reconstructions[]`. Function `getReassembly` will return the same pointer of `reconstruct_t` when the other fragments of this IPv6 packet arrived, `serial_input` keeps appending payload to the reconstructed IPv6 packet until the received bytes equal to total packet bytes. Finally, `handle_serial_packet` handles the decompressed IPv6 packet `msg`, which is writen to the `tun0` interface by calling `tun_write()`. Figure 4.21 shows the sequence diagram of reassembly when the serial port receives fragments. The "No fragmentation" block in Figure 4.21 handles the case that the header of `pkt` is neither a FRAG1 header nor a FRAGN header, in this case, there is no fragmentation on sending side, so there is no need for reassembly, `serial_input` just unpacks headers.

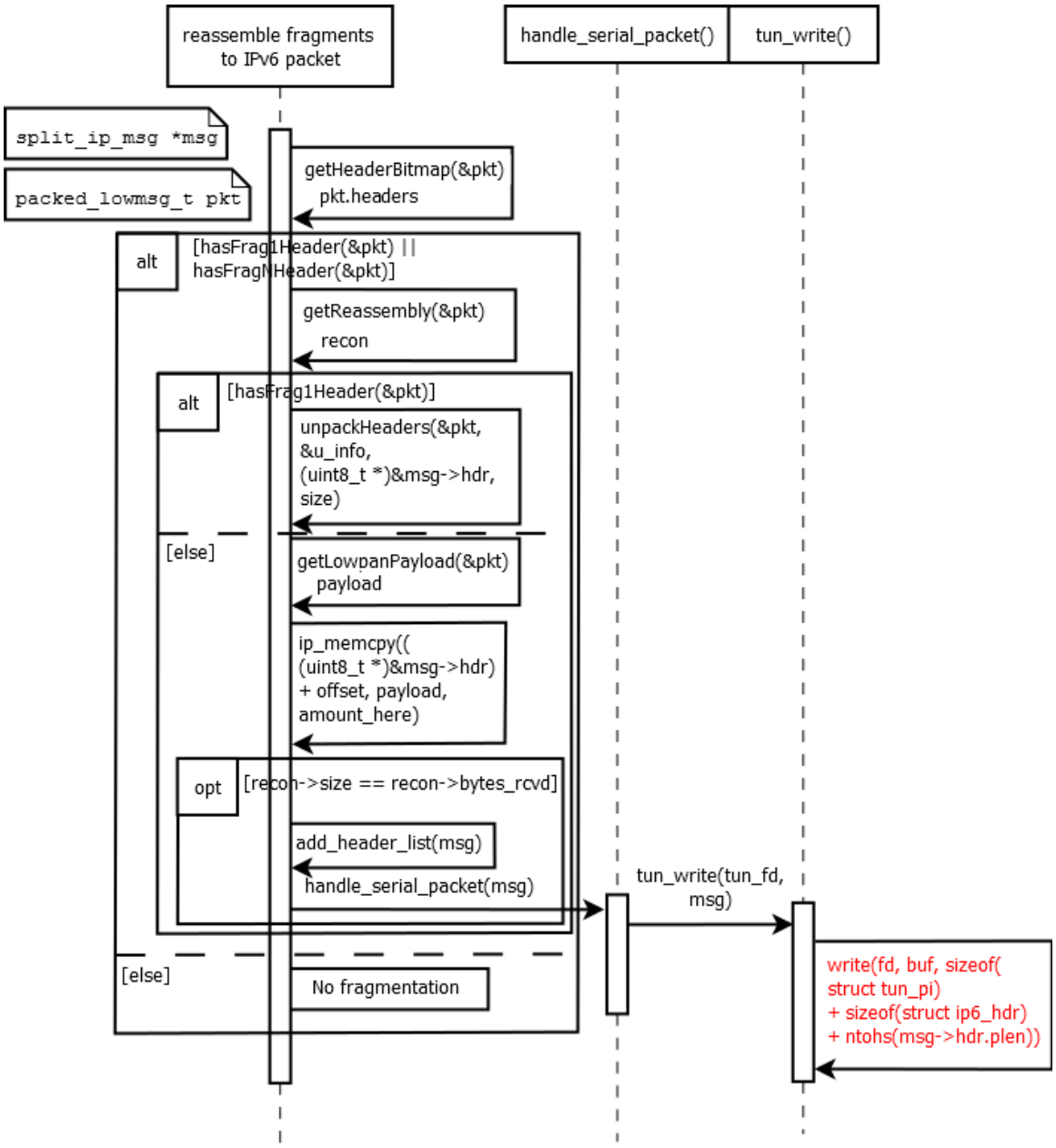


Figure 4.21: Sequence diagram of reassembly when serial port receives 6LoWPAN packets.

4.3 IPBaseStation Software Architecture

IPBaseStation sends and receives packets between a serial channel and the radio channel. IPBaseStation is running on a TelosB node which connects to a PC using USB cable. In `BaseStationP.nc` file, some send and receive interfaces are defined.

`UartSend` is defined as a `Send` interface, `RadioSend` is defined as an `Ieee154Send` interface, `UartReceive` and `RadioReceive` are defined as `Receive` interfaces. `Send`, `Ieee154Send` and `Receive` are TinyOS built-in interfaces, the interfaces and their implementations are shown in Table 4.1. The `Receive` interface has an event `message_t* receive(message_t* msg, void* payload, uint8_t len)` which must be implemented. When the `receive` event is triggered, the function that handles the event is called. When the radio interface receives IEEE 802.15.4 packets, it will send the packets to the PC through the serial port (UART interface). The edge router also sends packets to nodes in the 6LoWPAN network to check whether nodes are reachable, so the packets are sent to a radio channel to broadcast to nodes in the network when the UART interface receives packets. Figure 4.22 and Figure 4.23 show the sequence of packet transmission between radio interface and UART interface, respectively. Note that green function calls are that made to the TinyOS 2.1.1 libraries.

Table 4.1: TinyOS 2.1.1 interfaces and implementations for 6LoWPAN support in BLIP.

interface	implementation
`Send`	`UartSend`
`Ieee154Send`	`RadioSend`
`Receive`	`UartReceive`
`Receive`	`RadioReceive`

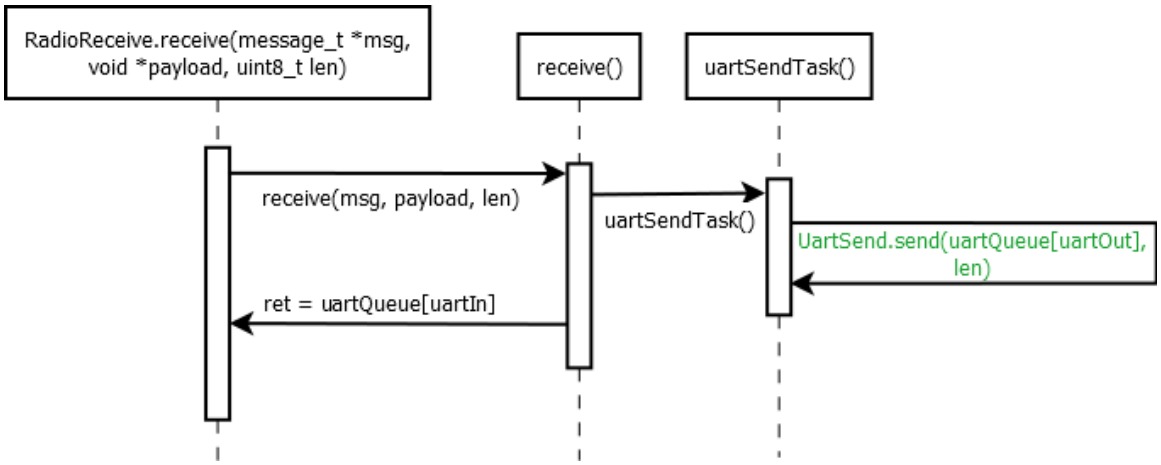


Figure 4.22: Transmission of packets from radio to uart.

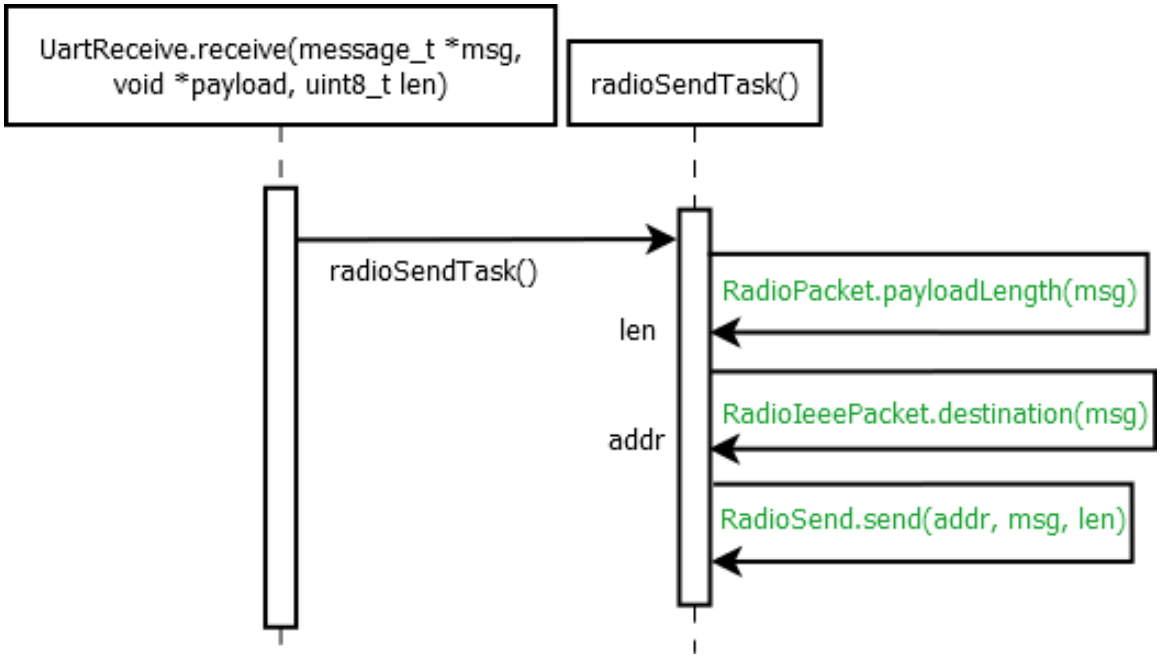


Figure 4.23: Transmission of packets from uart to radio.

4.4 Moving Node Software Implementation

4.4.1 Define Sensors

First, we need to define a data structure to store sensor readings. We add a new data structure in `TOSROOT/tos/lib/net/blip/Statistics.h`, as follows:

```
typedef nx_struct {
  nx_uint16_t tsr;  // visible to IR light sensor
     reading
  nx_uint16_t par;  // visible light sensor reading
  nx_uint16_t exttemp;  // temperature sensor reading
  nx_uint16_t hum;      // humidity sensor reading
  nx_uint16_t volt;     // voltage sensor reading
  nx_uint16_t sender;   // ID of the sender
} sensor_readings_t;
```

We also need to add reading interfaces in the `UDPEchoP.nc` file, ie.

```
interface Read<uint16_t> as ReadTSR;
interface Read<uint16_t> as ReadPAR;
interface Read<uint16_t> as ReadExtTemp;
interface Read<uint16_t> as ReadHum;
interface Read<uint16_t> as ReadVolt;
```

Five types of TelosB built-in sensors are used in this application. Table 4.2 shows the detail information of the sensors.

Table 4.2: TelosB built-in sensors used in moving node application.

Sensor type	Component in TinyOS	Manufacturer	Units
Visible to IR Light Sensor	HamamatsuS10871TsrC	Hamamatsu	Lux
Visible Light Sensor	HamamatsuS1087ParC	Hamamatsu	Lux
Temperature Sensor	SensirionSht11C	Sensirion	%
Humidity Sensor	SensirionSht11C	Sensirion	°C
Voltage Sensor	Msp430InternalVoltageC	Texas Instruments	Volts

In order to use the interfaces, we need to add sensor components in the `UDPEchoC.nc` file, ie.

```
components new HamamatsuS10871TsrC() as Sensor1;
components new HamamatsuS1087ParC() as Sensor2;
components new SensirionSht11C() as Sensor3;
components new DemoSensor() as Sensor4;
```

These components are part of the TinyOS source code.

Then we need to wire the interfaces to components, as follows:

```
UDPEchoP.ReadTSR -> Sensor1.Read;
UDPEchoP.ReadPAR -> Sensor2.Read;
UDPEchoP.ReadExtTemp -> Sensor3.Temperature;
UDPEchoP.ReadHum -> Sensor3.Humidity;
UDPEchoP.ReadVolt -> Sensor4.Read;
```

For each `Read` interface, it is required to implement a function to handle the `readDone()` event. When a reading operation of one sensor is done, it should signal the next reading. For example, the `readDone()` event for the visible to IR light sensor is:

```
1  event void ReadTSR.readDone(error_t result, uint16_t
       data) {
2    if (result == SUCCESS) {
3      stats.udp.tsr = data;
4      call ReadPAR.read();
5    }
6  }
```

`stats` is a `udp_report` data structure as follows:

```
1  nx_struct udp_report {
2    nx_uint16_6 seqno;
3    nx_uint16_t sender;
4    ip_statistics_t ip;
5    sensor_readings_t udp;
6    icmp_statistics_t icmp;
7    route_statistics_t route;
8  };
9
10 nx_struct udp_report stats;
```

The `udp_report` data structure was originally defined in BLIP, and we added the `sensor_readings_t` struct.

4.4.2 Sensor Readings Calculation

4.4.2.1 Light Sensors

TelosB has two built-in light sensors: Hamamatsu S1087 is a visible light sensor, Hamamatsu S1087-01 is a visible to IR light sensor. The component of Hamamatsu S1087 is `HamamatsuS1087ParC.nc`, the component of Hamamatsu S1087-01 is `HamamatsuS10871TsrC.nc`. The current of the sensor (I) can be converted to lux using the following formulas:

$$S1087 : lux = 0.769 \times 10^5 \times I \times 1000 \tag{4.1}$$

$$S1087 - 01 : lux = 0.625 \times 10^6 \times I \times 1000 \tag{4.2}$$

Lux is the SI (International System of Units) unit of illuminance and luminous emittance, measuring luminous flux per unit area [11]. The resistance at the output is 100 $k\Omega$, then the output current can be calculated by the following equation where AD_{output} is the sensor reading, $V_{ref} = 1.5$ V:

$$I = \frac{V_{sensor}}{100000\ \Omega}; V_{sensor} = AD_{output} \times \frac{V_{ref}}{2^{12}} \tag{4.3}$$

4.4.2.2 Humidity and Temperature Sensor

Humidity and Temperature are measured using the same sensor Sensirion SHT11 on TelosB, the component is `SensirionSht11C.nc`. The following formulas can convert the sensor readings to humidity and temperature:

$$humidity = -0.0000028 \times data \times data + 0.0405 \times data - 4 \tag{4.4}$$

$$temperature = -40 + 0.01 \times data \tag{4.5}$$

4.4.2.3 Voltage Sensor

The voltage sensor is the Msp430 internal voltage sensor, the component is `Msp430InternalVoltageC.nc`. The value of the sensor readings can be converted to voltage using the following formula:

$$voltage = \frac{data}{4096} \times 3 \tag{4.6}$$

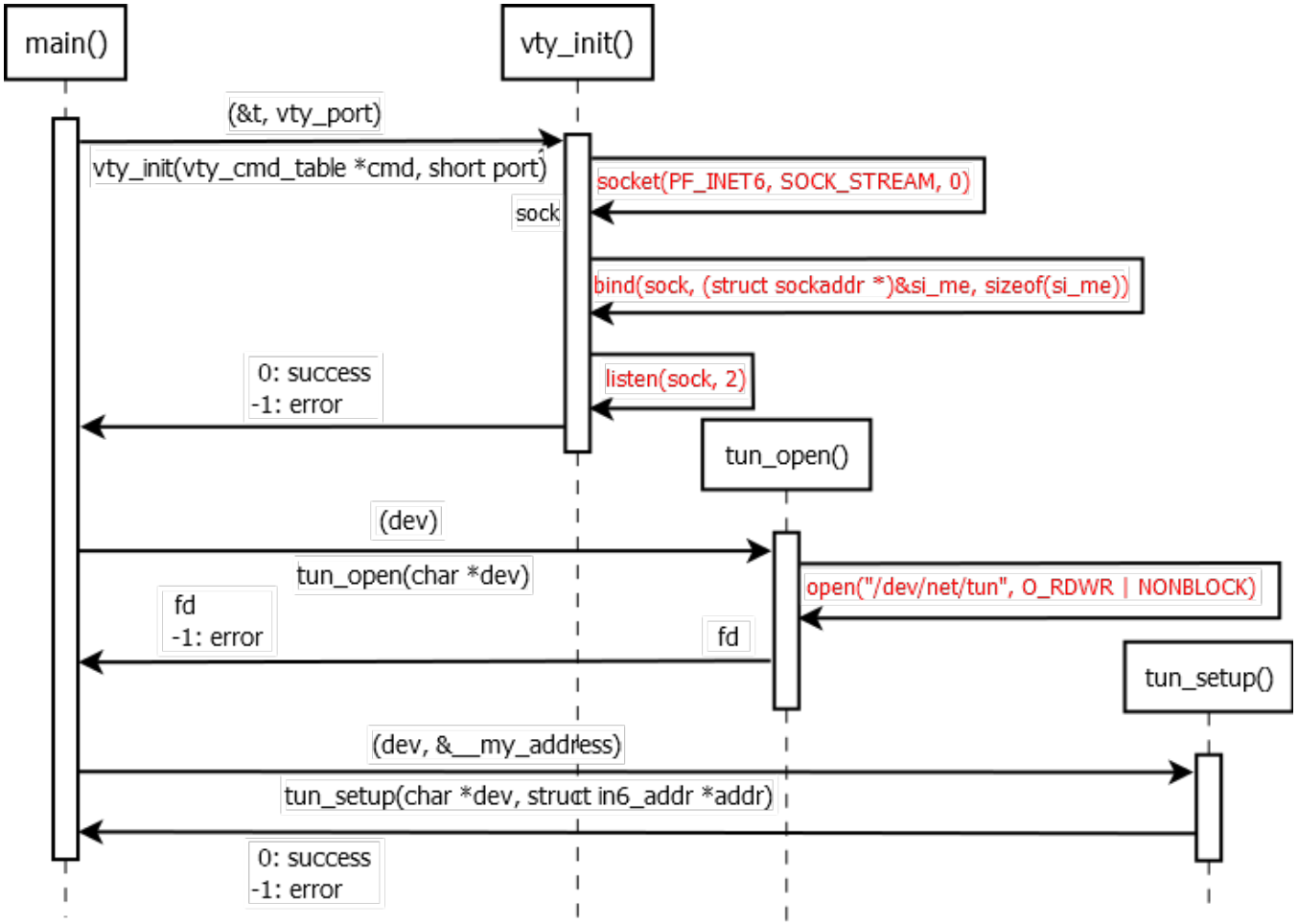


Figure 4.12: Sequence diagram of setting up TCP socket and tunnel interface in the `ip-driver` program.

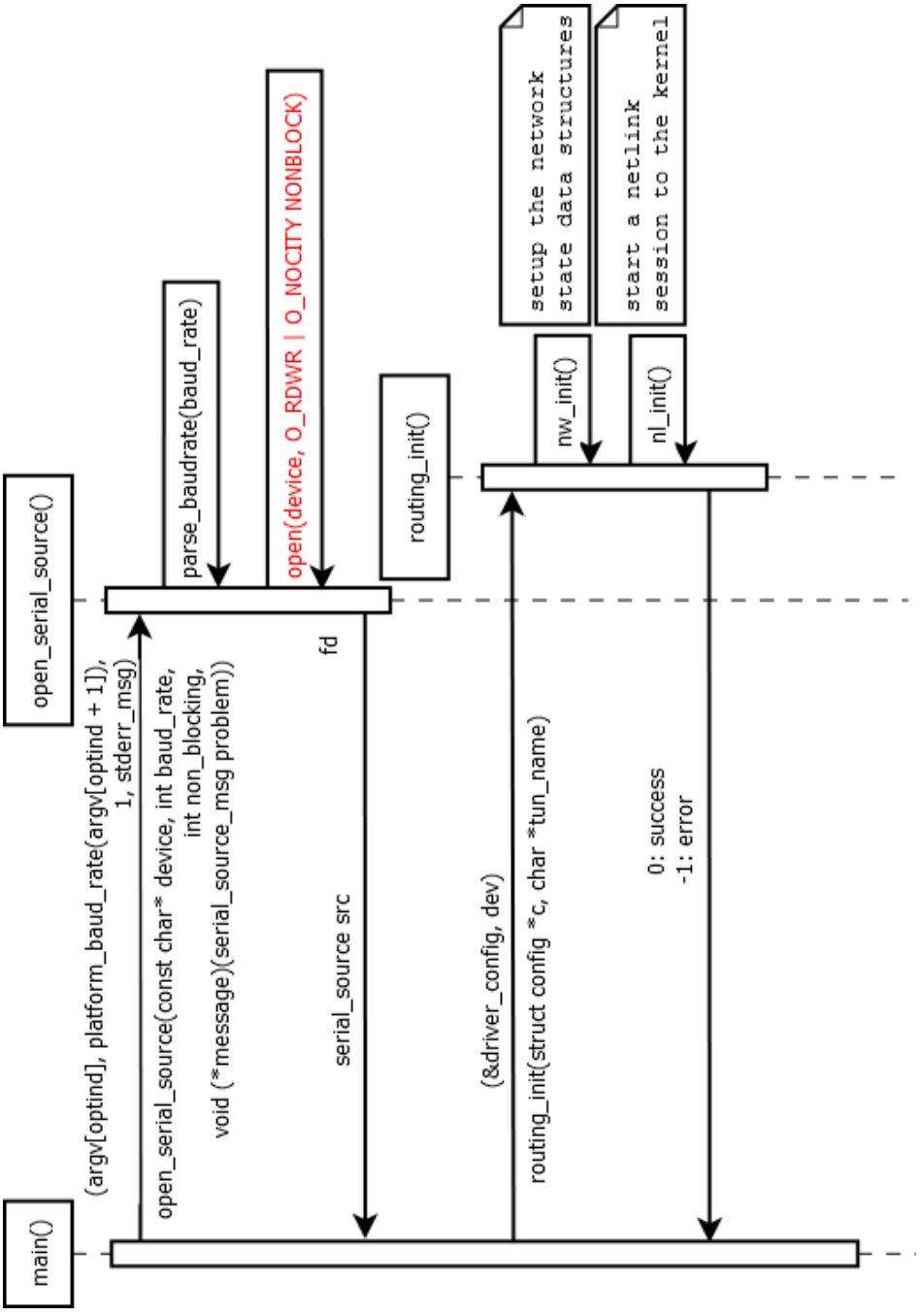


Figure 4.13: Sequence diagram of opening a serial port and initializing routing in ip-driver.

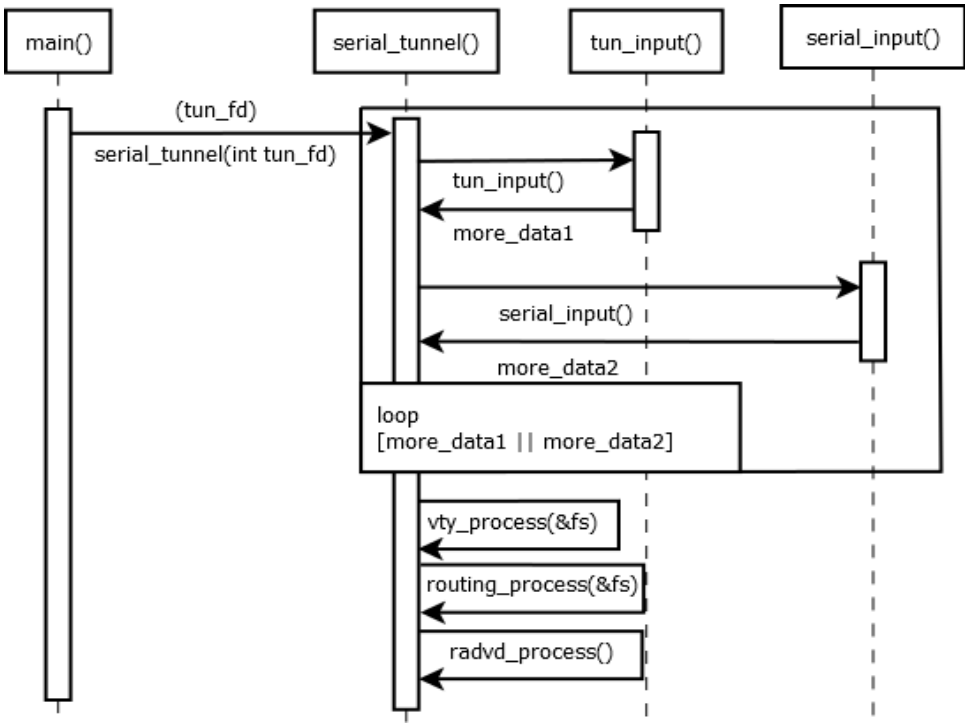


Figure 4.14: Sequence diagram of tunneling process of receiving data.

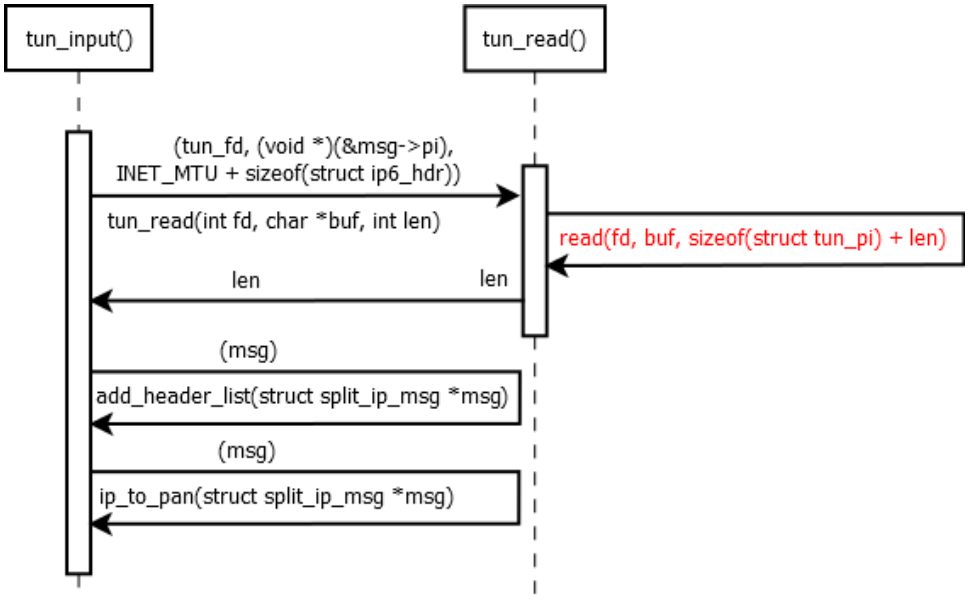


Figure 4.15: Sequence diagram of `tun_input()`.

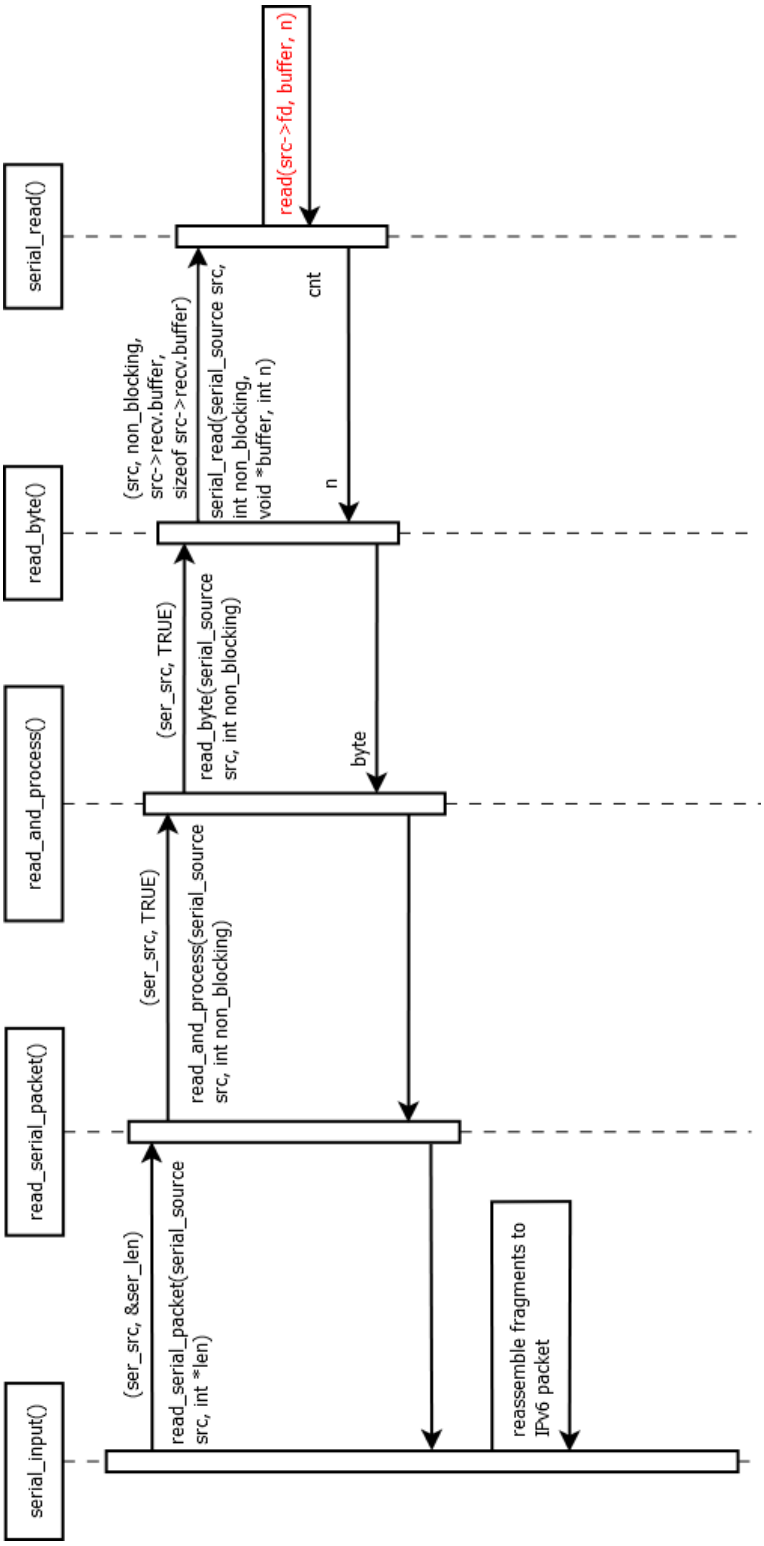


Figure 4.16: Sequence diagram of `serial_input()`.

```
struct serial_source_t {
#ifndef LOSE32
  int fd;
#else
  HANDLE hComm;
#endif
  bool non_blocking;
  void (*message)(serial_source_msg problem);

  /* Receive state */
  struct {
    uint8_t buffer[BUFSIZE];
    int bufpos, bufused;
    uint8_t packet[MTU];
    bool in_sync, escaped;
    int count;
    struct packet_list *queue[256]; // indexed by protocol
  } recv;
  struct {
    uint8_t seqno;
    uint8_t *escaped;
    int escapeptr;
    uint16_t crc;
  } send;
};
```

Figure 4.17: Definition of data type `serial_source_t`.

```
typedef struct packed_lowmsg {
  uint8_t headers;
  uint8_t len;
  // we preprocess the headers bitmap for easy processing.
  ieee154_saddr_t src;
  ieee154_saddr_t dst;
  uint8_t *data;
} packed_lowmsg_t;
```

Figure 4.18: Definition of data type `packed_lowmsg_t`.

```
typedef struct {
  uint16_t tag;           /* datagram label */
  uint16_t size;          /* the size of the packet we are reconstructing */
  void    *buf;           /* the reconstruction location */
  uint16_t bytes_rcvd;    /* how many bytes from the packet we have
                             received so far */
  uint8_t timeout;
  uint8_t nxt_hdr;
  uint8_t *transport_hdr;
  struct ip_metadata metadata;
} reconstruct_t;
```

Figure 4.20: Definition of data type `reconstruct_t`.

Chapter 5

Experimental Design and Results

5.1 Mobile Wireless Sensor Architecture

We evaluated the moving node dynamically using a mobile sensor network testbed. The testbed uses a model train (see Figure 5.1) to carry our moving nodes (see Figure 5.2).

Figure 5.1: Model train running on the wireless sensor network testbed in ITB214.

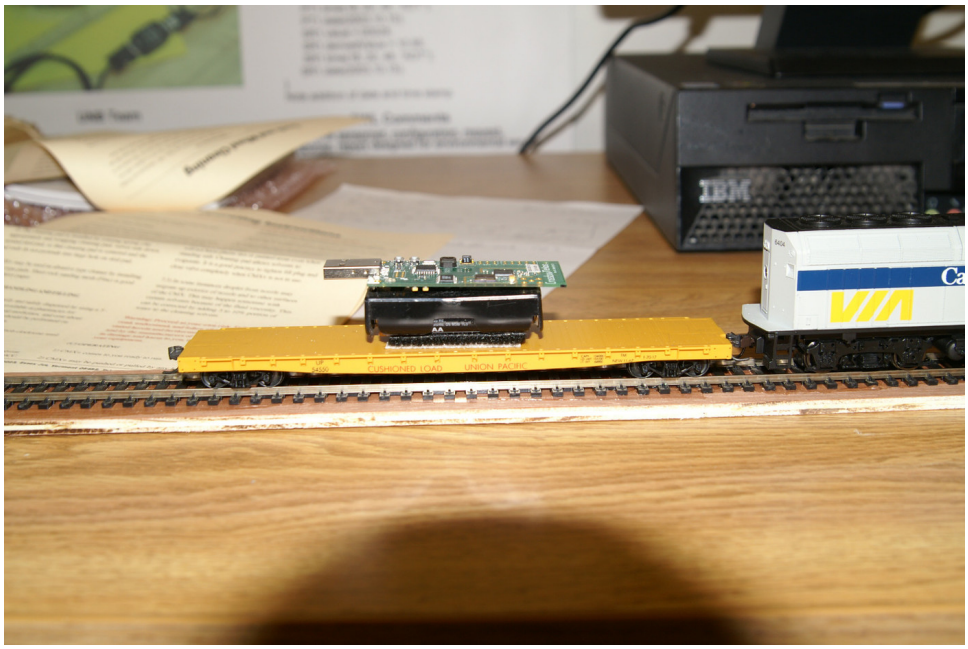


Figure 5.2: A Telosb node is carried on the model train.

The testbed model railroad track is hanging from light fixtures in the wireless communications lab ITB214 (see Figure 5.3).

Figure 5.3: General view of train track in ITB214.

The parameters of testbed are shown in Figure 5.4.

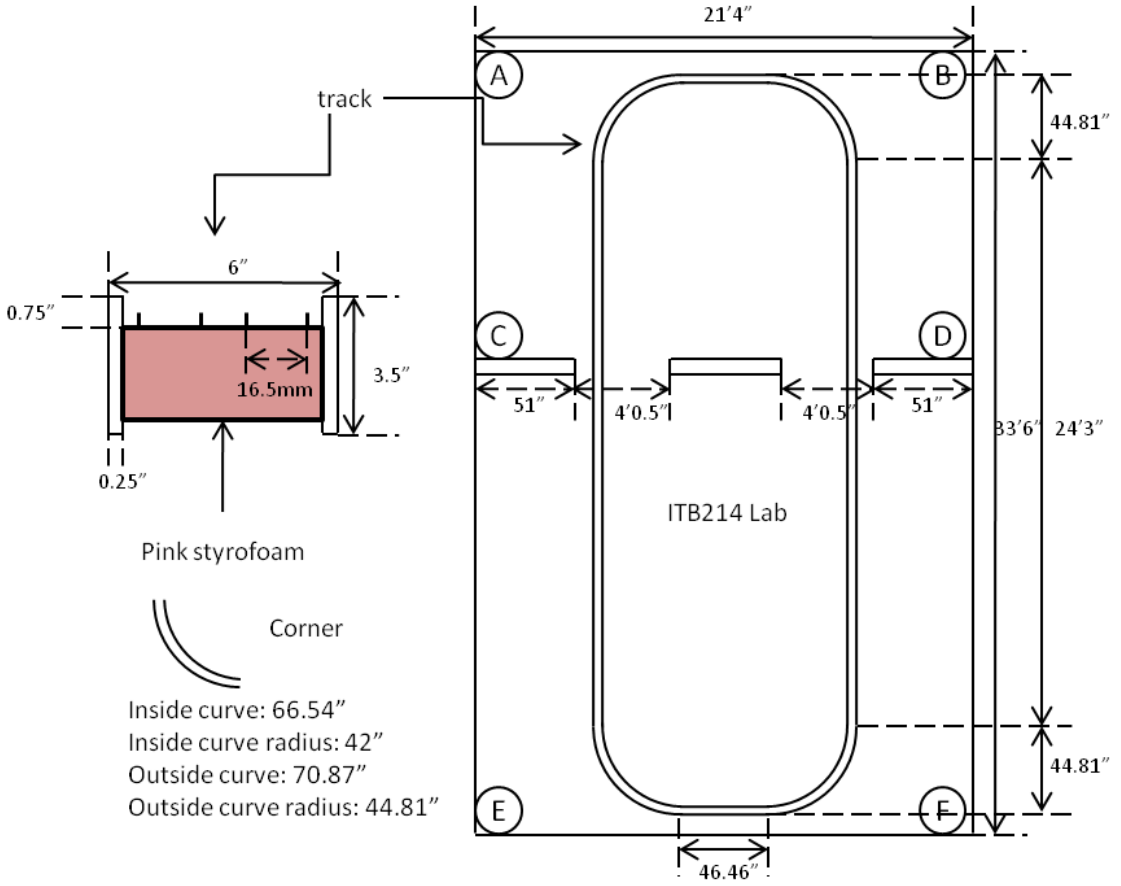


Figure 5.4: Model railroad track in ITB214.

A, B, C, D, E, F in Figure 5.4 show the approximate positions of the stationary nodes; moving nodes are on one or two model trains. The railroad train track has two short sides, two long sides and four corners. The corner is shown in Figure 5.5.

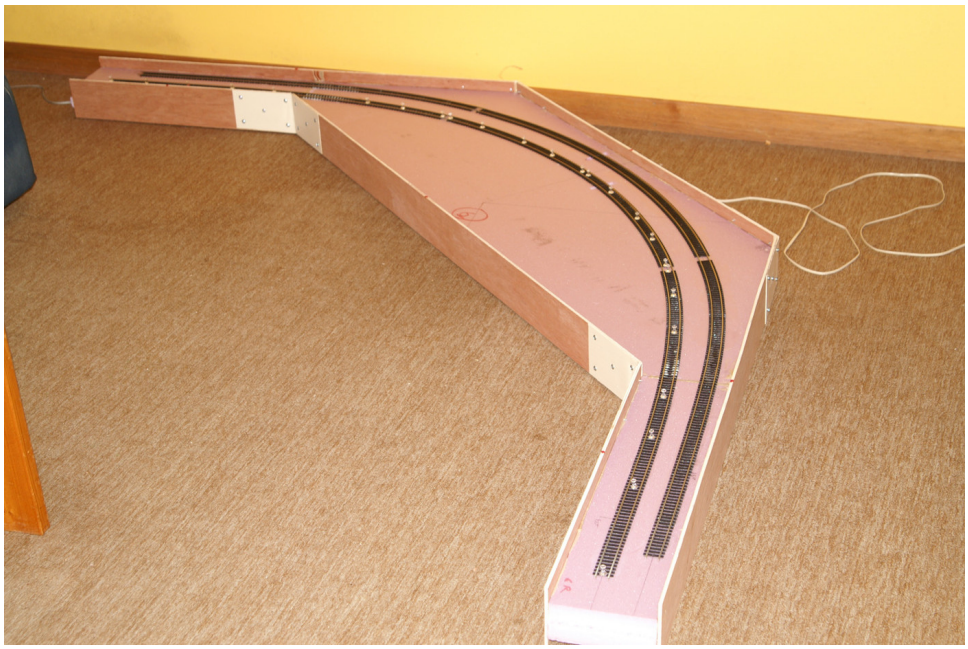


Figure 5.5: Model railroad track corner.

5.2 Experimental Testing Software Architecture

5.2.1 Packet Specification

In our testing, the moving node sends UDP packets to the edge router over IPv6. The whole UDP packet structure is shown in Figure 5.6. In our testing, the whole UDP packet size is 189 bytes, and the payload of UDP packet is 141 bytes as shown in Figure 5.6. Figure 5.7 shows the payload of one UDP packet used in our testing. The payload begins from the 49th byte of the UDP packet shown in Figure 5.6, each field of the payload is illustrated in Table 5.1.

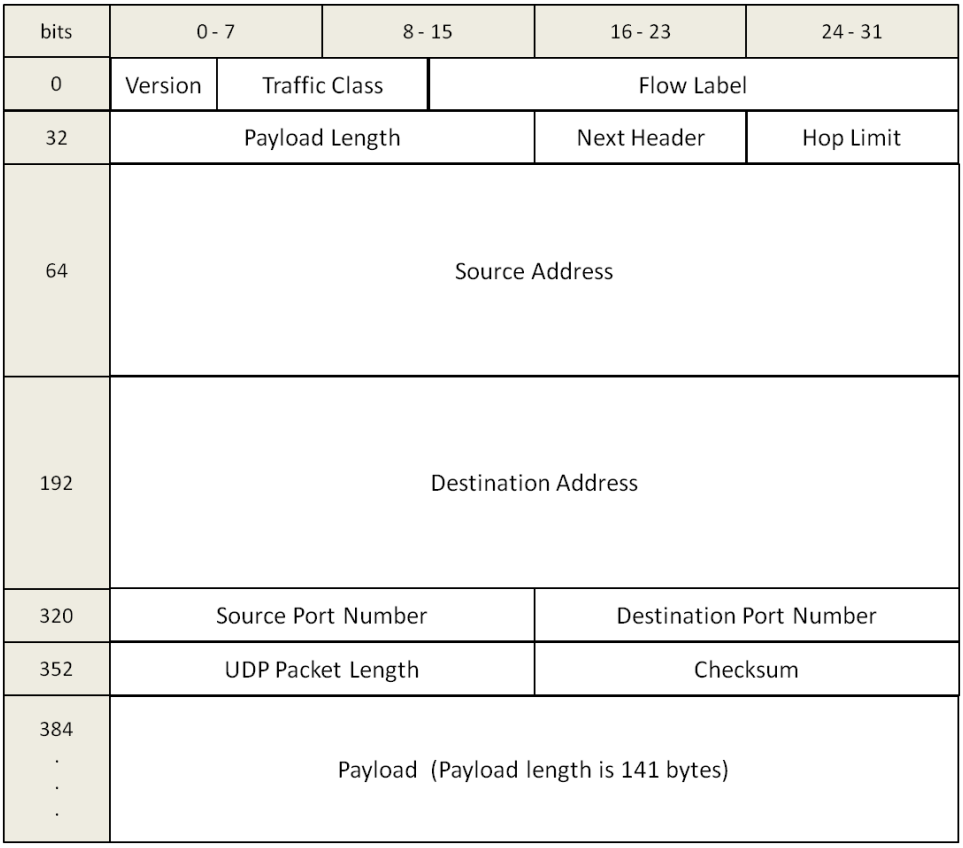


Figure 5.6: Packet structure of UDP over IPv6.

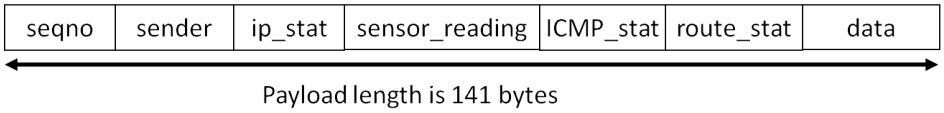


Figure 5.7: 141 bytes Payload structure.

Table 5.1: UDP payload structure.

Field	size (in bytes)	Meaning
seqno	2	Packet sequence number
sender	2	The TOS_NODE_ID of the sender node
ip_stat	9	IP statistics
sensor_reading	12	Sensor reading values
ICMP_stat	9	ICMP statistics
route_stat	7	Route statistics
data	100	An array of zeros

The following is the details of the payload fields.

1. seqno: Packet sequence number is a 16-bit integer, seqno increments by one for each out going packet from the sender node.

2. sender: The TOS_NODE_ID of the sender node, TOS_NODE_ID is specified when downloading the program to sensor node. In our testing, the TOS_NODE_ID of the moving nodes are 1 and 8.

3. ip_stat: ip_stat is of type `ip_statistics_t`, it is TinyOS built-in data structure.

4. sensor_reading: sensor_reading is of type `sensor_readings_t`, Figure 5.8 shows the specification of sensor_reading.

5. ICMP_stat: ICMP_stat is of type `icmp_statistics_t`, it is TinyOS built-in data structure.

6. route_stat: route_stat is of type `route_statistics_t`, it is TinyOS built-in data structure.

7. data: data is an array of zeros, the size of the array is 100 bytes. We need this array to force each UDP packet to exceed 127 bytes, so that one UDP packet can be fragmented into several 6LoWPAN fragments.

The details of the data structures used in UDP packets can be found in Appendix A.1.

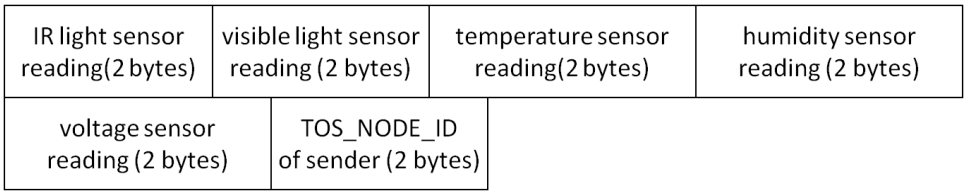


Figure 5.8: Packet specification of field sensor_reading.

5.2.2 Testing software architecture

Figure 5.9 shows the architecture of the software testing system.

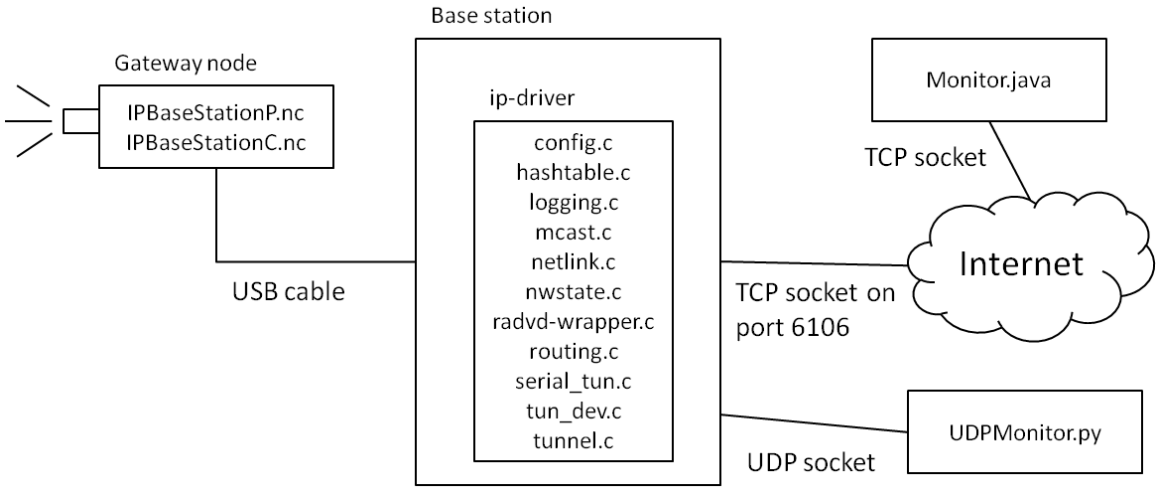


Figure 5.9: `Monitor.java` monitors the topology changes of the network, and `UDPMonitor.py` stores all the received UDP packets and calculates packet loss.

5.2.2.1 Detecting Packet Route Topology Change

Our Java testing program is called `Monitor.java`. As shown in Figure 5.11, `Monitor.java` builds a TCP socket connection to the server which is running `ip-driver`

on port 6106. As discussed in Section 4.2, `ip-driver` is a C program that opens a TCP socket and keeps listening on port 6106. `Monitor.java` has one input stream called `in` which is an object of type `BufferedReader` and one output stream called `out` which is an object of type `PrintWriter`. Object `in` is used to receive messages from `ip-driver` and `out` is used to send messages to `ip-driver`. When a client is connected to `ip-driver`, the client receives a welcome message. `Monitor.java` stores the welcome message in a buffer. In our experiment, we monitor the topology changes when nodes are moving, so `Monitor.java` keeps sending the `routes` command to `ip-driver` using the `PrintWriter`. An example output from the `routes` command is shown in Figure 5.10. Algorithm 5.1 shows the main topology change detection logic used in the `Monitor.java` program.

```
0x2: [1.1]0x1 [1.1]0x64
0x3: [1.1]0x64
0x6: [1.1]0x64
0x5: [1.1]0x1 [1.1]0x64
0x4: [1.1]0x1 [1.1]0x64
0x8: [1.0]0x6 [1.1]0x64
0x7: [1.1]0x1 [1.1]0x64
0x1: [1.1]0x64
0x64:
blip:ib214m06.cs.unb.ca>
```

Figure 5.10: A sample response to the `routes` command.

Route information of each node returned by `ip-driver` is stored in a Java `HashMap` object. The sending of a `routes` command to `ip-driver` and the receiving of route information from `ip-driver` are synchronized, which means that `Monitor.java` will not send the next `routes` command to `ip-driver` until it receives all the data from the socket input stream. `Monitor.java` makes sure it receives all the data from the socket input stream by checking whether the return value of `in`'s `readLine` method is `null` (see line 5 of Algorithm 5.1).

When `Monitor.java` receives a node's route information, the program checks whether the node exists in the `HashMap` (see line 21 of Algorithm 5.1).

Algorithm 5.1: Algorithm for detecting wireless sensor network topology changes.

Input: continuous lines of route strings output by `ip-driver` "routes" command

Output: Detecting topology changes and storing changed routes in `route_info.log`

```
1  Map path ← HashMap whose keys and values are of type String;
2  PrintWriter out ;                                   /* to ip-driver */
3  BufferedReader in ;                                 /* from ip-driver */
4  String msg;
5  out.println("routes");
6  while (msg ← in.readLine()) ≠ NULL do
7      if msg starts with "blip" then
8          arrowIndex ← msg.indexOf(">");
9          msg ← msg.substring(arrowIndex +1);
10     end
11     if msg equals "0x64" then
12         out.println("routes");
13     else
14         String[] splitMsg ← msg.split(":");
15         String node ;                                /* left part */
16         String route ;                               /* right part */
17         node ← splitMsg [0];
18         if splitMsg.length > 1 then
19             route ← splitMsg [1];
20         end
21         if path.containsKey(node) and route ≠ NULL then
22             String oldPath ← path.get(node).split("&")[0];
23             if oldPath ≠ route then
24                 String changeTime ← current time ;   /* use Java Date
                   object */
25                 String routeAndTime ← route + "&" + changeTime;
26                 path.put(node, routeAndTime);
27             end
28         else
29             if route ≠ NULL then
30                 String changeTime ← current time;
31                 String routeAndTime ← route + "&" + changeTime;
32                 path.put(node, routeAndTime);
33             end
34         end
35     end
36 end
```

If the node does not exist in the `HashMap`, `Monitor.java` adds the new node with its route information and time stamp concatenated using the "&" symbol to separate the route and time stamp (see lines 30 to 34 of Algorithm 5.1). If the node exists in the `HashMap`, `Monitor.java` checks whether the stored route of the node is the same as the received route. If the routes are different, `Monitor.java` updates the node's route to be the new received route (see lines 23 to 27 of Algorithm 5.1), and saves the node's new route to a log file `route_info.log`. If the routes are the same, `Monitor.java` ignores the update. Figure 5.11 shows the message passing process between `Monitor` and `ip-driver`. Figure 5.12 shows a portion of a sample `route_info.log` file.

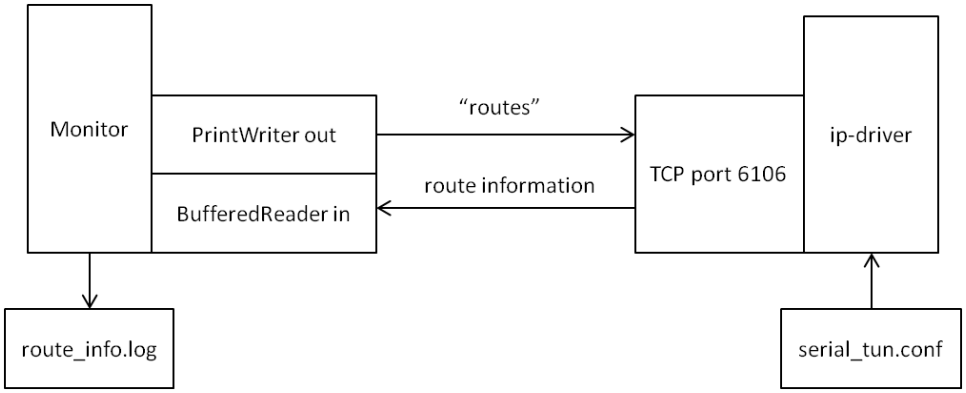


Figure 5.11: Message passing between `Monitor` and `ip-driver`.

```
New route for node 0x1: [25.5]0x64  2013/02/22 13:26:13
New route for node 0x5: [1.0]0x1 [25.5]0x64     2013/02/22 13:26:13
New route for node 0x2: [25.5]0x7 [1.0]0x1 [25.5]0x64   2013/02/22 13:27:07
New route for node 0x2: [1.0]0x7 [1.0]0x1 [25.5]0x64    2013/02/22 13:32:07
New route for node 0x2: [25.5]0x7 [1.0]0x1 [25.5]0x64   2013/02/22 13:37:07
New route for node 0x4: [25.5]0x64  2013/02/22 14:04:55
```

Figure 5.12: A portion of a sample `route_info.log` file.

The new routes information is read line by line, so `Monitor.java` receives one line at each interation. Lines 6 to 9 in Algorithm 5.1 get rid of the BLIP console message (e.g. `blip:ib214m06.cs.unb.ca>`). The address of the edge router is set to `0x64` in a config file `serial_tun.conf`, and it does not have a route, so lines 10 and 11

ignore a `0x64` only route, and send the next `routes` command. If the source of the new route is not `0x64`, Algorithm 5.1 splits the new message received into two parts: source node and route (lines 13 to 19). For example, message

```
0x7: [1.0]0x6 [1.0]0x4 [1.0]0x64
```

can be split into `0x7` and `[1.0]0x6 [1.0]0x4 [1.0]0x64`, where `0x7` is the source node and `[1.0]0x6 [1.0]0x4 [1.0]0x64` is the route. A Java hash map is used to store the latest nodes' routes. As shown in lines from 29 to 35, if node `0x7` does not exist in the hash map, Algorithm 5.1 adds node `0x7` and its route `[1.0]0x6 [1.0]0x4 [1.0]0x64` with the current time into the hash map, where `node` is the key, and the route with current time `routeAndTime` is the value (route and change time are separated by the "&" character). If node `0x7` already exists in the hash map, Algorithm 5.1 compares the stored route of node `0x7` and the most recent route. If they are not equal, Algorithm 5.1 stores the most recent route of node `0x7` to the hash map as shown in lines 21 to 28. All changed routes are recorded in the `route_info.log` file.

5.2.2.2 Measuring Packet Loss

A Python program called `UDPMonitor.py` calculates packet loss during testing. The moving node sends UDP packets through port 7001 to the edge router port 7000, so `UDPMonitor.py` opens a socket on port 7000 to receive UDP packets from the edge router using the following Python statements:

```
1 s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
2 s.bind(('fec0::64', 7000))
```

The packet loss is calculated using equation (5.1),

$$PL = 100(T - R)/T \tag{5.1}$$

where PL is packet loss in percent, T is the number of UDP packets transmitted

during a specific time period (e.g. five hours), and R is the number of UDP packets received during the same time period. The UDP packets are read by line 21 in the `UDPMonitor.py` program given in Appendix C.2 as follows:

```
21 data, addr = s.recvfrom(65535)
```

where `data` is the byte stream of received UDP packets, `addr` is the source address of UDP packets, and 65535 is the maximum length of one received packet. If the length of `data` is greater than 0 (indicating a successfully received UDP packet), R increments by one, as shown in Algorithm 5.2. The first two bytes of `data` indicate the packet sequence number, which is a 16-bit integer. We assign the sequence number to T first. The first UDP packet's sequence number is one, and the sequence number increments by one for each UDP packet sent from the moving node. When `UDPMonitor.py` starts running, we assign the sequence number of the first received UDP packet minus one to offset O. T is translated by offset O to $T - O$ to account for the fact that the first received packet is likely not sequence number 1.

Algorithm 5.2: Algorithm for calculating packet loss in program `UDPMonitor.py`.

Input: continuous UDP packets on port 7000
Output: UDP packet loss in percent, and the received packets stored in the file `UDP.log`

```
1  T ← 0;
2  R ← 0;
3  while time elapsed ≤ 5 hours do
4  |  data ← recvfrom "fec0::64" port 7000;
5  |  print data to UDP.log;
6  |  if length of data > 0 then
7  |  |  R ← R + 1;
8  |  |  high ← first byte;
9  |  |  low ← second byte;
10 |  |  T ← high × 256 + low ;            /* sequence number (seqno) of
   |  |  transmitted packet */
11 |  |  if R == 1 then   /* account for the first received packet */
12 |  |  |  O ← T − 1;
13 |  |  end
14 |  |  T ← T − O;
15 |  |  PL ← 100 × (T − R)/T;
16 |  |  print T, R, PL to UDP.log;
17 |  end
18 end
```

The payload of one data packet is 141 bytes long, as specified in Section 5.2.1 and Table 5.1. Figure 5.13 illustrates the packet loss measurement process. The `ip-driver` program sends out the complete UDP packet to the destination in the wireless area network (e.g. `fec0::64` on port 7000 in our wireless sensor network) after reassembly. As UDP packets are sent at a rate of one packet per 10 seconds, this means we obtain approximately 360 records per hour in the `UDP.log` file.

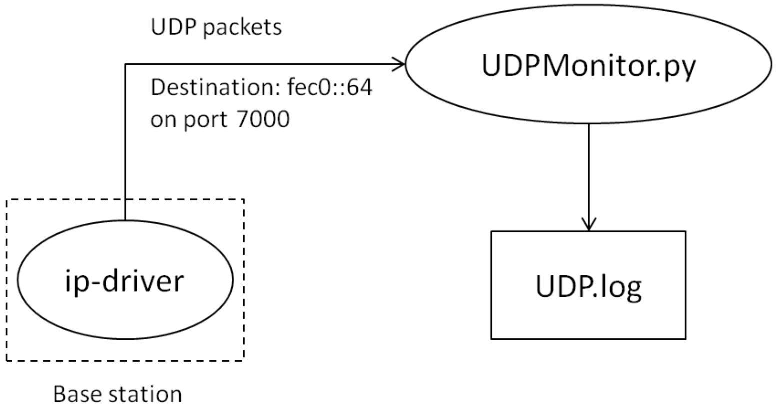
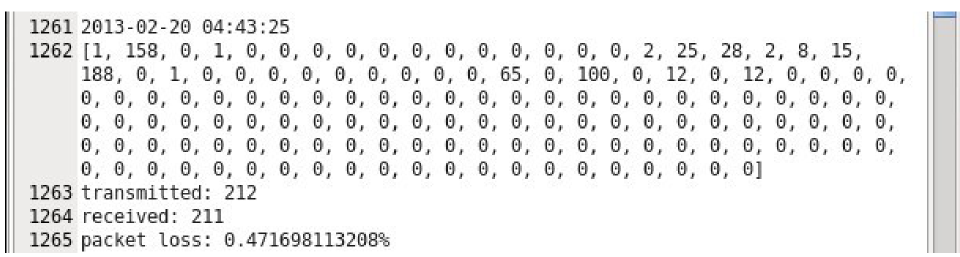


Figure 5.13: Measuring packet loss.

Figure 5.14 shows a sample output of the `UDPMonitor` program. The complete `UDPMonitor.py` program is given in Appendix C.2.



```
1261 2013-02-20 04:43:25
1262 [1, 158, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 25, 28, 2, 8, 15,
     188, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 65, 0, 100, 0, 12, 0, 12, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
1263 transmitted: 212
1264 received: 211
1265 packet loss: 0.471698113208%
```

Figure 5.14: Sample output from the `UDPMonitor` program. Line 1262 shows the 141 byte payload for a received UDP packet. The first two bytes (1, 158) are the high and low parts of the sequence number, which is $1 \times 256 + 158 = 414$.

5.3 What We Measured

The software that controls the train is called `JMRI`, the Java Model Railroad Interface [10]. `JMRI` is an open source project which is building tools for model railroad computer control. `JMRI` controls the direction of train travel and the speed of train in speed steps. In our experiment, we have six stationary nodes, whose positions are shown in Figure 5.4, and one or two moving nodes carried on the train. We

test topology changes and packet loss of the wireless sensor network under different train velocities and different routing table update periods. The default routing table update period is 60 seconds. In our experiments, we modified the routing table update period to 6 seconds, and 0.6 seconds. This update was made by changing the period of the `SortTimer` in the `IPRouting.nc` file. The speeds we chose in the experiment are 40, 60, 80, 100, and 120 (in speed steps). Speed step velocities map nonlinearly to actual speed, so we measured the actual velocities by recording the time required for the train to travel three times around the track. We used both the inner track and outer track in our testing, which have length of 23.92m and 24.36m, respectively. Engine 6404 is on the inner track and engine 1478 is on the outer track. The actual velocities corresponding to each speed step are shown in Table 5.2 and Table 5.3.

Table 5.2: Speed step velocities map for train engine 6404.

Speed Steps	40	60	80	100	120
Time Consumed	15:45	8:09	5:22	3:58	3:11
Velocity (m/s)	0.0759	0.1466	0.2229	0.3015	0.3757

Table 5.3: Speed step velocities map for train engine 1478.

Speed Steps	40	60	80	100	120
Time Consumed	20:51	10:23	6:48	4:56	4:04
Velocity (m/s)	0.0584	0.1173	0.1791	0.2469	0.2995

The sensor nodes used in the experiment are TelosB nodes that consist of an 8 MHz TI MSP430 microcontroller with 10 kB RAM, and an IEEE 802.15.4 2.4 GHz radio cc2420, our experiment operating system for sensor nodes is Tinyos 2.1.1. Each test uses six stationary nodes and one moving node, and we recorded all the received UDP packets and the topology changes of the wireless sensor network over a five

hour period. Packet loss was measured as described in Section 5.2.2.2.

5.3.1 Testing Results

5.3.1.1 One Moving Node Testing

The moving node address is `fec0::1` for one moving node testing. Table 5.4 shows the number of topology changes (NTC) and packet loss (PL) with different train speeds using engine 6404, and different routing table update periods (RTUP).

Table 5.4: Number of topology changes (NTC) and packet loss under different routing table update periods, and different train velocities in one moving node testing.

Test	RTUP	Speed Steps	NTC	R/T	PL
1	60	40	99	1845 / 1845	0.000%
2	60	60	148	1845 / 1845	0.000%
3	60	80	97	1844 / 1845	0.054%
4	60	100	310	1845 / 1845	0.000%
5	60	120	334	1844 / 1845	0.054%
6	6	40	315	1843 / 1845	0.108%
7	6	60	403	1845 / 1845	0.000%
8	6	80	379	1843 / 1846	0.163%
9	6	100	583	1843 / 1845	0.108%
10	6	120	682	1843 / 1845	0.108%
11	0.6	40	398	1840 / 1845	0.271%
12	0.6	60	335	1836 / 1845	0.489%
13	0.6	80	428	1839 / 1845	0.325%
14	0.6	100	472	1832 / 1946	0.758%
15	0.6	120	842	1819 / 1845	1.409%

Figure 5.15 shows packet loss vs. train velocity with different routing table update periods. We can see from Figure 5.15 that packet loss increases when the routing table update period becomes shorter. Routing table update periods of 60 seconds and 6 seconds have the same packet loss for high and low train velocity, while a routing table update period of 0.6 seconds has increasing packet loss with increasing train velocity. The network with the BLIP default routing table update period of 60

seconds is the most reliable one.

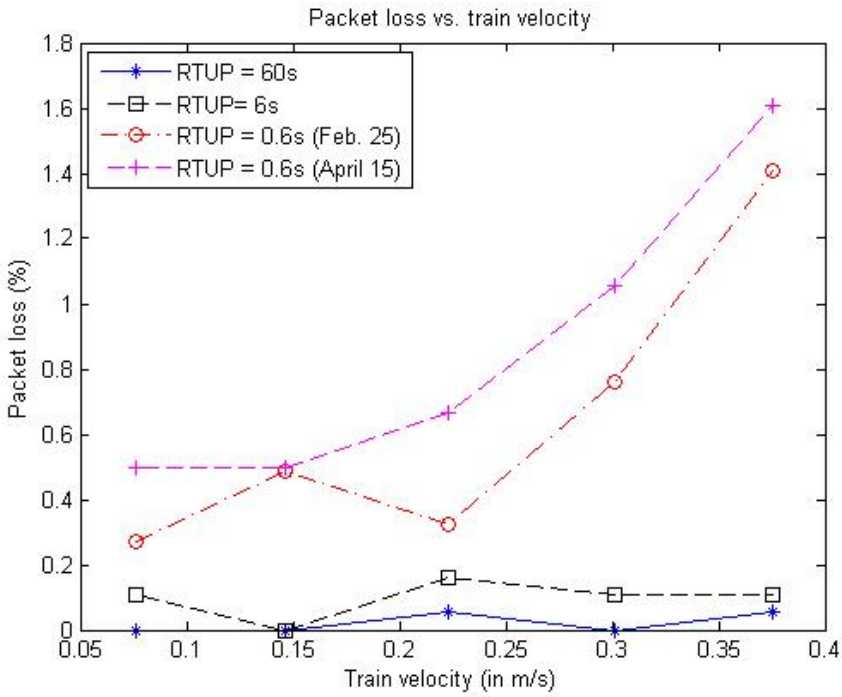


Figure 5.15: Packet loss vs. train velocity for one moving node testing.

To make sure the experimental readings we obtained are consistent, we repeated the one moving node testing experiment for a RTUP of 0.6 seconds. The result is shown as the April 15 0.6 seconds RTUP data line in Figure 5.15. As we can see, the packet loss still increases when train velocity goes faster, so our observation that packet loss increases significantly with a 0.6 seconds RTUP is repeatable.

Figure 5.16 plots the number of topology changes vs. train velocity under different routing table update periods. We can see from Figure 5.16 that the number of topology changes increases when the routing table update period decreases, and the number of topology changes is more likely to increase when velocity becomes faster. The fewest topology changes occur when the routing table update period is the default value of 60 seconds.

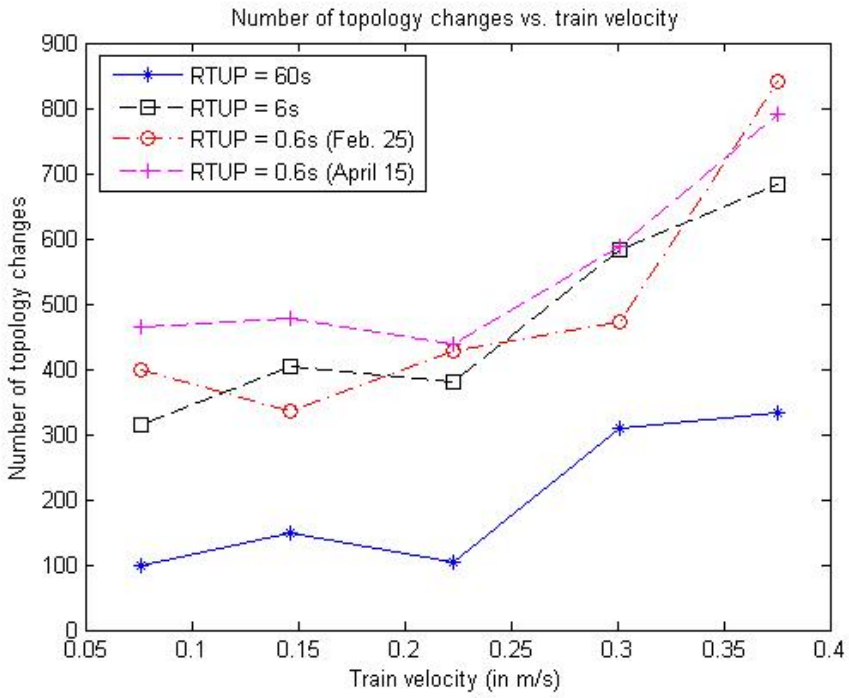


Figure 5.16: Number of topology changes vs. train velocity for one moving node testing.

5.3.1.2 Two Moving Nodes Testing

The moving nodes addresses are `fec0::1` (behind engine 6404) and `fec0::8` (behind engine 1478) in the two moving nodes testing. Table 5.5 shows the number of topology changes (NTC) and packet loss (PL) with different train speed steps (SS) and different routing table update periods (RTUP) in two moving nodes testing.

Table 5.5: Number of topology changes (NTC) and packet loss under different routing table update periods, and different train velocities in two moving node testing.

Test	RTUP	SS	NTC	R/T(fec0::1)	PL(fec0::1)	R/T(fec0::8)	PL(fec0::8)
1	60	40	175	1796/1801	0.278%	1791/1801	0.555%
2	60	60	356	1792/1800	0.444%	1798/1801	0.167%
3	60	80	252	1801/1801	0.000%	1798/1801	0.167%
4	60	100	385	1797/1800	0.167%	1795/1801	0.333%
5	60	120	340	1801/1801	0.000%	1801/1801	0.000%
6	6	40	1124	1794/1801	0.389%	1795/1801	0.333%
7	6	60	1033	1796/1801	0.278%	1800/1801	0.056%
8	6	80	810	1793/1801	0.444%	1793/1801	0.444%
9	6	100	570	1801/1801	0.000%	1799/1801	0.111%
10	6	120	771	1795/1801	0.333%	1798/1801	0.167%
11	0.6	40	781	1798/1801	0.167%	1798/1801	0.167%
12	0.6	60	672	1795/1801	0.333%	1798/1801	0.167%
13	0.6	80	909	1794/1801	0.389%	1780/1801	1.166%
14	0.6	100	691	1784/1801	0.944%	1788/1801	0.722%
15	0.6	120	262	1796/1801	0.278%	1792/1801	0.500%

Figure 5.17 shows packet loss vs. train velocity with different routing table update periods. Figure 5.18 shows the total packet loss vs. train velocity with different routing table update periods. Figure 5.19 shows the number of topology changes vs. train velocity under different routing table update periods.

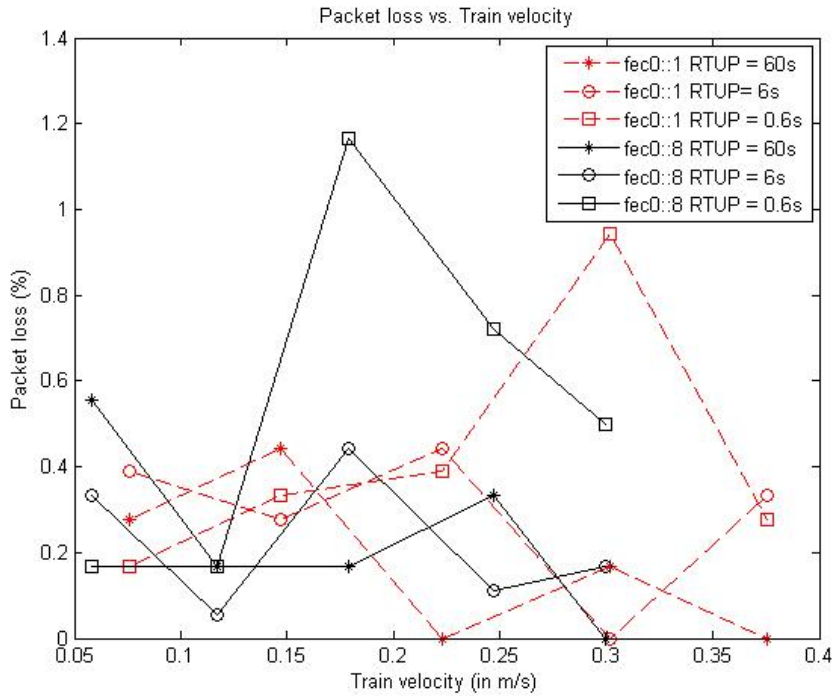


Figure 5.17: Packet loss vs. train velocity for two moving nodes testing.

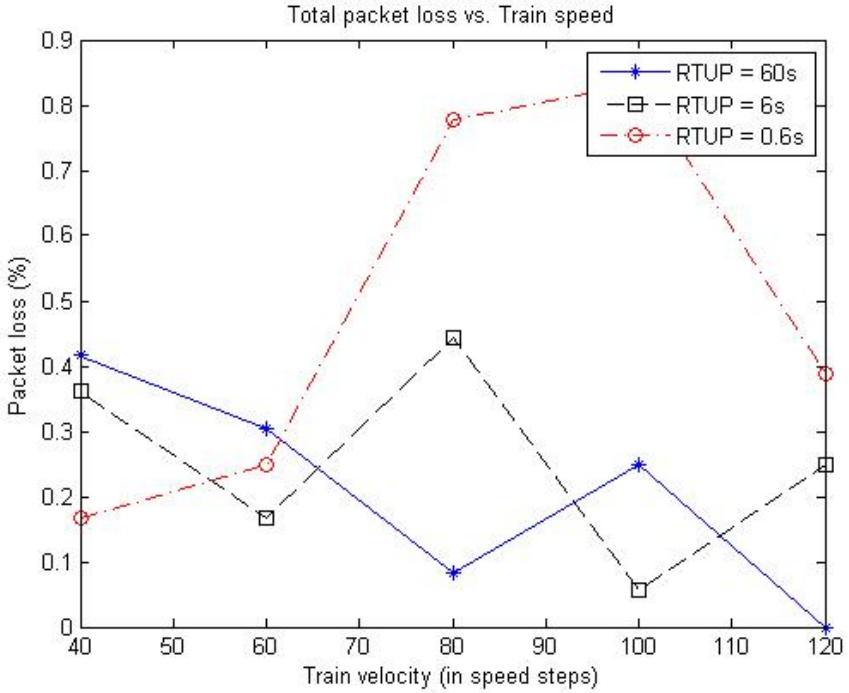


Figure 5.18: Total packet loss vs. train velocity for two moving nodes testing.

From Figure 5.17 and Figure 5.18, we can see that packet loss is higher at 0.6 seconds

routing table update period, but unlike single node testing, the packet loss decreases as velocity goes higher at 0.6 seconds RTUP and velocity of 100 (in speed steps, the two engines used in the two nodes testing have different speed steps to actual velocity mapping) for node `fec0::8`, and velocity of 100 (in speed steps) for node `fec0::1`. Both one node testing and two nodes testing show that the BLIP default RTUP of 60 seconds has the smallest packet loss.

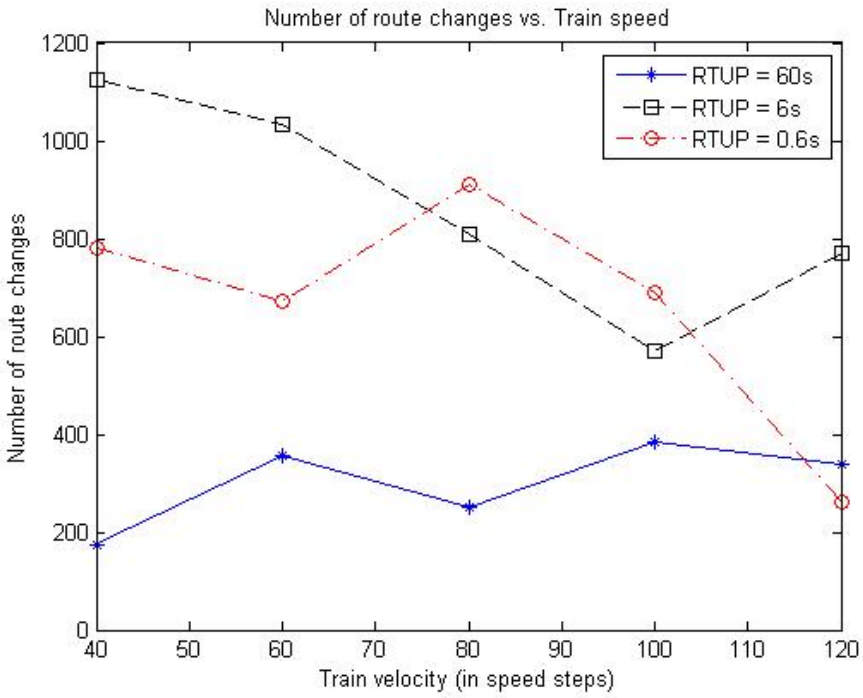


Figure 5.19: Number of topology changes vs. train velocity for two moving node testing.

We can see from Figure 5.18 that when the train velocity is greater than 60 (in speed steps), routing table update period of 0.6 seconds has higher packet loss than routing table update period of 60 seconds and 6 seconds.

Chapter 6

Summary and Conclusions

6.1 Summary

A detailed explanation of sending and receiving UDP messages using the Berkeley Low Power IP (BLIP) stack is provided in this thesis. A 6LoWPAN wireless sensor network with both one and two moving nodes and six stationary nodes has been built and tested under different BLIP routing table update periods and different train velocities. TelosB sensor nodes were used in all the experimental tests. The experimental results show that the 6LoWPAN protocol can accommodate objects moving at a rate up to 0.38 m/s. The maximum packet loss in our testing was 1.41% over a period of five hours. Our tests with routing table update periods of 60 s, 6 s, and 0.6 s showed that BLIP operates successfully in all three update periods. We observed that a RTUP of 0.6 s had an approximately 13 times higher packet loss compared to a 6 s RTUP, and 26 times higher packet loss compared to a 60 s RTUP where the moving node has a 0.376 m/s velocity.

A Java-based web application has been implemented to monitor mobile nodes moving in a wireless sensor network and communicating with a 6LoWPAN protocol.

WebSocket and TCP sockets are used to transmit route topology messages to client browsers, and UDP packets are used to transmit sensor readings from sensor nodes.

6.2 Conclusions

Having 0.6 second routing table update period (RTUP) resulted in a higher packet loss than with a 6 second or 60 second RTUP. In one moving node testing, routing table update rate of 0.6 s has significantly higher packet loss (up to 1.4% compared to 0.16%). In two moving nodes testing, the maximum packet loss of 0.6 second RTUP is approximately 1.2%, which is higher than packet loss of 6 second RTUP and 60 second RTUP (most of packet losses of 6 second RTUP and 60 second RTUP are between 0.2% and 0.4%).

6.3 Future Work

Future work on evaluating mobile nodes moving inside a wireless sensor network might include the following topics:

1. Testing of different sensor platforms instead of just TelosB (e.g. MicaZ, IRIS, TinyNode) could illustrate the robustness of 6LoWPAN with different hardware.

2. Our tests were all done at the lowest power level 3, would the packet loss decrease if higher power levels were used?

3. Our 24 m long mobile wireless network testbed track is too small for measuring. Extending the track to permit e.g. 250 m or more of travel distance would provide a more complete evaluation of the 6LoWPAN protocol under longer wireless distances.

4. The user interface of the WSNWeb application could be improved if that position information was available to display the accurate position of the train running in the wireless communication lab ITB214.

References

[1] *Apache Tomcat*, http://tomcat.apache.org/, Accessed March, 2013.

[2] *Atmel RZUSBstick*, http://www.atmel.com/tools/rzusbstick.aspx, Accessed January, 2013.

[3] *BerkeleyWEBS Wireless embedded systems*, http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip, Accessed July, 2011.

[4] *Embedding Jetty on eclipse*, http://wiki.eclipse.org/Jetty/Tutorial/Embedding_Jetty, Accessed March, 2013.

[5] *Graphviz - Graph Visualization Software*, http://www.graphviz.org/, Accessed April 5, 2013.

[6] *HTML5 Canvas at w3schools.com*, http://www.w3schools.com/html/html5_canvas.asp, Accessed March, 2013.

[7] *IEEE 802.15 WPAN Task Group 4 (TG4)*, http://www.ieee802.org/15/pub/TG4.html, Accessed August, 2011.

[8] *JavaScript Graphics Lirary (JSGL.org)*, http://www.jsgl.org/doku.php?id=home, Accessed March 25, 2013.

[9] *Jetty WebSocket*, http://wiki.eclipse.org/Jetty/Feature/WebSockets, Accessed February, 2013.

[10] *Jmri*, http://jmri.sourceforge.net/, Accessed February, 2013.

[11] *LUX on Wikipedia*, http://en.wikipedia.org/wiki/Lux, Accessed December, 2011.

[12] *Tomcat Architecture - Startup*, http://tomcat.apache.org/tomcat-5.5-doc/architecture/startup.html, Accessed March, 2013.

[13] G. Barrenetxea, F. Ingelrest, G. Schaefer, M. Vetterli, O. Couach, and M. Parlange, *Sensorscope: Out-of-the-box environmental monitoring*, Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on, April 2008, pp. 332 –343.

[14] Ed. J. Hui and P. Thubert, *Compression format for ipv6 datagrams in low power and lossy networks*, Tech. report, February 2011, IETF draft-ietf-6lowpan-hc-15, update 4944 (if aapproved).

[15] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li-Shiuan Peh, and Daniel Rubenstein, *Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet*, Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, 2002, pp. 96–107.

[16] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*, Ietf request for comments rfc4944, September 2007.

[17] C. Schumacher N. Kushalnagar, G. Montenegro, *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*, Tech. report, 2007.

[18] Krishna Sampigethaya, Radha Poovendran, Mingyan Li, Linda Bushnell, and Richard Robinson, *Security of wireless sensor network enabled health monitoring for future airplanes*, 26th International Congress of The Aeronautical Sciences at Alaska, USA, September 2008, ICAS Paper Number: ICAS 2008-9.1.3.

[19] Lucie Smith and Ian Lipner, *Free Pool of IPv4 Address Space Depleted*, `http://www.nro.net/news/ipv4-free-pool-depleted`, Accessed July, 2011.

[20] Lohith Y.S., *6LowPAN: Wireless Internet Of Things*, Presentation at Department of ECE, Indian Institue of Science, Bangalore, November 2010.

[21] Gang Zhao, *Wireless sensor networks for industrial process monitoring and control: A survey*, Network Protocols and Algorithms **3** (2011), no. 1, 46–63.

Appendix A

Moving Node Software

A.1 Data Structures Used in UDP Packets in Testing

`UDPReport.h` composes the definition for the UDP payload structure.

Listing A.1: `UDPReport.h`

```
#ifndef _UDPREPORT_H
#define _UDPREPORT_H

#include <Statistics.h>
#include <AM.h>

nx_struct udp_report {
  nx_uint16_t seqno;
  nx_uint16_t sender;
  ip_statistics_t    ip;
  sensor_readings_t   udp;
  icmp_statistics_t  icmp;
  route_statistics_t route;
  nx_uint16_t data[50];
} ;

#endif
```

`Statistics.h` contains the definitions for each field of UDP payload.

Listing A.2: `Statistics.h`

```
/* Statistics from the core 6lowpan/IPv6 fragmentation and forwarding engine */
typedef nx_struct {
  nx_uint16_t sent;       // total IP datagrams sent
  nx_uint16_t forwarded;  // total IP datagrams forwarded
  nx_uint8_t rx_drop;     // L2 frags dropped due to 6lowpan failure
  nx_uint8_t tx_drop;     // L2 frags dropped due to link failures
  nx_uint8_t fw_drop;     // L2 frags dropped when forwarding due to queue overflow
  nx_uint8_t rx_total;    // L2 frags received
  nx_uint8_t encfail;     // frags dropped due to send queue
```

```
10 } ip_statistics_t;
11
12 typedef nx_struct {
13   nx_uint8_t hop_limit;
14   nx_uint16_t parent;
15   nx_uint16_t parent_metric;
16   nx_uint16_t parent_etx;
17 } route_statistics_t;
18
19 typedef nx_struct {
20   nx_uint8_t sol_rx;
21   nx_uint8_t sol_tx;
22   nx_uint8_t adv_rx;
23   nx_uint8_t adv_tx;
24   nx_uint8_t echo_rx;
25   nx_uint8_t echo_tx;
26   nx_uint8_t unk_rx;
27   nx_uint16_t rx;
28 } icmp_statistics_t;
29
30 /* sensor readings we defined */
31 typedef nx_struct {
32   nx_uint16_t tsr;
33   nx_uint16_t par;
34   nx_uint16_t exttemp;
35   nx_uint16_t hum;
36   nx_uint16_t volt;
37   nx_uint16_t sender;
38 } sensor_readings_t;
```

A.2 Main Program for Moving Node

`UDPMovingC.nc` contains the configuration for the moving node application.

Listing A.3: `UDPMovingC.nc`

```
1 #include <6lowpan.h>
2 #ifdef DBG_TRACK_FLOWS
3 #include "TestDriver.h"
4 #endif
5
6 configuration UDPMovingC {
7
8 } implementation {
9   components new HamamatsuS10871TsrC() as Sensor1;
10  components new HamamatsuS1087ParC() as Sensor2;
11  components new SensirionSht11C() as Sensor3;
12  components new DemoSensorC() as Sensor4;
13
14  UDPMovingP.ReadTSR -> Sensor1.Read;
15  UDPMovingP.ReadPAR -> Sensor2.Read;
16  UDPMovingP.ReadExtTemp -> Sensor3.Temperature;
17  UDPMovingP.ReadHum -> Sensor3.Humidity;
18  UDPMovingP.ReadVolt -> Sensor4.Read;
19
20  components MainC, LedsC;
21  components UDPMovingP;
22
23  UDPMovingP.Boot -> MainC;
24  UDPMovingP.Leds -> LedsC;
25
```

```
26    components new TimerMilliC();
27    components IPDispatchC;
28
29    UDPMovingP.RadioControl -> IPDispatchC;
30    components new UdpSocketC() as Status;
31
32    UDPMovingP.Status -> Status;
33
34    UDPMovingP.StatusTimer -> TimerMilliC;
35
36    components UdpC;
37    UDPMovingP.IPStats -> IPDispatchC.IPStats;
38    UDPMovingP.UDPStats -> UdpC;
39    UDPMovingP.RouteStats -> IPDispatchC.RouteStats;
40    UDPMovingP.ICMPStats -> IPDispatchC.ICMPStats;
41
42 #ifdef SIM
43    components BaseStationC;
44 #endif
45 #ifdef DBG_TRACK_FLOWS
46    components TestDriverP, SerialActiveMessageC as Serial;
47    components ICMPResponderC, IPRoutingP;
48    TestDriverP.Boot -> MainC;
49    TestDriverP.SerialControl -> Serial;
50    TestDriverP.ICMPPing -> ICMPResponderC.ICMPPing[unique("PING")];
51    TestDriverP.CmdReceive -> Serial.Receive[AM_TESTDRIVER_MSG];
52    TestDriverP.IPRouting -> IPRoutingP;
53    TestDriverP.DoneSend -> Serial.AMSend[AM_TESTDRIVER_MSG];
54    TestDriverP.AckSend -> Serial.AMSend[AM_TESTDRIVER_ACK];
55    TestDriverP.RadioControl -> IPDispatchC;
56 #endif
57 #ifdef DBG_FLOWS_REPORT
58    components TrackFlowsC;
59 #endif
60 }
```

`UDPMovingP.nc` composes the module `UDPMovingP` for the moving node application.

Listing A.4: `UDPMovingP.nc`

```
1  #include <IPDispatch.h>
2  #include <lib6lowpan.h>
3  #include <ip.h>
4  #include <lib6lowpan.h>
5  #include <ip.h>
6  #include "UDPReport.h"
7  #include "PrintfUART.h"
8
9  #define REPORT_PERIOD 75L
10 #define REPORT_DEST   "fec0::64"
11
12 module UDPMovingP {
13   uses {
14     interface Boot;
15     interface SplitControl as RadioControl;
16     interface UDP as Status;
17
18     interface Leds;
19
20     interface Timer<TMilli> as StatusTimer;
21
22     interface Statistics<ip_statistics_t> as IPStats;
23     //interface Statistics<udp_statistics_t> as UDPStats;
24     interface Statistics<sensor_readings_t> as UDPStats;
25     interface Statistics<route_statistics_t> as RouteStats;
26     interface Statistics<icmp_statistics_t> as ICMPStats;
27
28     interface Read<uint16_t> as ReadTSR;
29     interface Read<uint16_t> as ReadPAR;
```

```
30      interface Read<uint16_t> as ReadExtTemp;
31      interface Read<uint16_t> as ReadHum;
32      interface Read<uint16_t> as ReadVolt;
33
34    }
35
36  } implementation {
37
38    bool timerStarted;
39    nx_struct udp_report stats;
40    struct sockaddr_in6 route_dest;
41
42
43    event void Boot.booted() {
44      call RadioControl.start();
45      timerStarted = FALSE;
46
47      call IPStats.clear();
48      call RouteStats.clear();
49      call ICMPStats.clear();
50      printfUART_init();
51
52  #ifdef REPORT_DEST
53      route_dest.sin6_port = hton16(7000);
54      inet_pton6(REPORT_DEST, &route_dest.sin6_addr);
55      //call StatusTimer.startOneShot(call Random.rand16() % (1nx_uint16_t *1024 *
            REPORT_PERIOD));
56      call StatusTimer.startPeriodic(10000);
57  #endif
58
59      dbg("Boot", "booted: %i\n", TOS_NODE_ID);
60      call Status.bind(7001);
61    }
62
63    event void RadioControl.startDone(error_t e) {
64
65    }
66
67    event void RadioControl.stopDone(error_t e) {
68
69    }
70
71    event void Status.recvfrom(struct sockaddr_in6 *from, void *data,
72                               uint16_t len, struct ip_metadata *meta) {
73
74    }
75
76    event void StatusTimer.fired() {
77
78      if (!timerStarted) {
79        //call StatusTimer.startPeriodic(1024 * REPORT_PERIOD);
80
81        timerStarted = TRUE;
82      }
83      call Leds.led0Toggle();
84      stats.seqno++;
85      stats.sender = TOS_NODE_ID;
86
87      call IPStats.get(&stats.ip);
88      call UDPStats.get(&stats.udp);
89      stats.udp.sender = TOS_NODE_ID;
90      call ICMPStats.get(&stats.icmp);
91      call RouteStats.get(&stats.route);
92      call ReadTSR.read();
93
94      call Status.sendto(&route_dest, &stats, sizeof(stats));
95    }
96
```

```
97    event void ReadTSR.readDone(error_t result, uint16_t data) {
98    if(result == SUCCESS) {
99      stats.udp.tsr = data;
100     call ReadPAR.read();
101   }
102   }
103
104   event void ReadPAR.readDone(error_t result, uint16_t data) {
105   if(result == SUCCESS) {
106     stats.udp.par = data;
107     call ReadExtTemp.read();
108   }
109   }
110
111   event void ReadExtTemp.readDone(error_t result, uint16_t data) {
112   if(result == SUCCESS) {
113     stats.udp.exttemp = data;
114     call ReadHum.read();
115   }
116   }
117
118   event void ReadHum.readDone(error_t result, uint16_t data) {
119   if(result == SUCCESS) {
120     stats.udp.hum = data;
121     call ReadVolt.read();
122   }
123   }
124
125   event void ReadVolt.readDone(error_t result, uint16_t data) {
126   if(result == SUCCESS) {
127     stats.udp.volt = data;
128   }
129   }
130 }
```

Appendix B

Stationary Node Software

`UDPMovingC.nc` contains the configuration for the stationary node application.

Listing B.1: `UDPStationaryC.nc`

```
1  #include <6lowpan.h>
2  #ifdef DBG_TRACK_FLOWS
3  #include "TestDriver.h"
4  #endif
5
6  configuration UDPStationaryC {
7
8  } implementation {
9
10   components MainC, LedsC;
11   components UDPStationaryP;
12
13   UDPStationaryP.Boot -> MainC;
14   UDPStationaryP.Leds -> LedsC;
15
16   components new TimerMilliC();
17   components IPDispatchC;
18
19   UDPStationaryP.RadioControl -> IPDispatchC;
20   components new UdpSocketC() as Status;
21
22   UDPStationaryP.Status -> Status;
23
24   UDPStationaryP.StatusTimer -> TimerMilliC;
25
26   components UdpC;
27   UDPStationaryP.IPStats -> IPDispatchC.IPStats;
28   UDPStationaryP.UDPStats -> UdpC;
29   UDPStationaryP.RouteStats -> IPDispatchC.RouteStats;
30   UDPStationaryP.ICMPStats -> IPDispatchC.ICMPStats;
31
32   components UDPShellC;
33
34 #ifdef SIM
35   components BaseStationC;
36 #endif
37 #ifdef DBG_TRACK_FLOWS
38   components TestDriverP, SerialActiveMessageC as Serial;
39   components ICMPResponderC, IPRoutingP;
40   TestDriverP.Boot -> MainC;
```

```
41    TestDriverP.SerialControl -> Serial;
42    TestDriverP.ICMPPing -> ICMPResponderC.ICMPPing[unique("PING")];
43    TestDriverP.CmdReceive -> Serial.Receive[AM_TESTDRIVER_MSG];
44    TestDriverP.IPRouting -> IPRoutingP;
45    TestDriverP.DoneSend -> Serial.AMSend[AM_TESTDRIVER_MSG];
46    TestDriverP.AckSend -> Serial.AMSend[AM_TESTDRIVER_ACK];
47    TestDriverP.RadioControl -> IPDispatchC;
48  #endif
49  #ifdef DBG_FLOWS_REPORT
50    components TrackFlowsC;
51  #endif
52  }
```

`UDPStationaryP.nc` composes the module `UDPStationaryP` for the moving node application.

Listing B.2: `UDPStationaryP.nc`

```
1   #include <IPDispatch.h>
2   #include <lib6lowpan.h>
3   #include <ip.h>
4   #include <lib6lowpan.h>
5   #include <ip.h>
6
7   #include "UDPReport.h"
8   #include "PrintfUART.h"
9
10  #define REPORT_PERIOD 75L
11  #define REPORT_DEST   "fec0::64"
12
13  module UDPStationaryP {
14    uses {
15      interface Boot;
16      interface SplitControl as RadioControl;
17      interface UDP as Status;
18      interface Leds;
19      interface Timer<TMilli> as StatusTimer;
20      interface Statistics<ip_statistics_t> as IPStats;
21      //interface Statistics<udp_statistics_t> as UDPStats;
22      interface Statistics<sensor_readings_t> as UDPStats;
23      interface Statistics<route_statistics_t> as RouteStats;
24      interface Statistics<icmp_statistics_t> as ICMPStats;
25    }
26
27  } implementation {
28
29    bool timerStarted;
30    nx_struct udp_report stats;
31    struct sockaddr_in6 route_dest;
32
33
34    event void Boot.booted() {
35      call RadioControl.start();
36      timerStarted = FALSE;
37
38      call IPStats.clear();
39      call RouteStats.clear();
40      call ICMPStats.clear();
41      printfUART_init();
42
43  #ifdef REPORT_DEST
44      route_dest.sin6_port = hton16(7000);
45      inet_pton6(REPORT_DEST, &route_dest.sin6_addr);
46      call StatusTimer.startPeriodic(10000);
47  #endif
48      call Status.bind(7001);
49    }
```

```
50
51    event void RadioControl.startDone(error_t e) {
52
53    }
54
55    event void RadioControl.stopDone(error_t e) {
56
57    }
58
59    event void Status.recvfrom(struct sockaddr_in6 *from, void *data,
60                               uint16_t len, struct ip_metadata *meta) {
61    }
62
63    event void StatusTimer.fired() {
64
65      if (!timerStarted) {
66        timerStarted = TRUE;
67      }
68    }
69  }
```

Appendix C

Testing Software

C.1 Java Monitor Program

`Monitor.java` keeps track on the route topology update, and store the new routes in the file `route_info.log`.

Listing C.1: `Monitor.java`

```
1  import java.io.BufferedReader;
2  import java.io.BufferedWriter;
3  import java.io.FileWriter;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.io.PrintWriter;
7  import java.net.Socket;
8  import java.text.DateFormat;
9  import java.text.SimpleDateFormat;
10 import java.util.Date;
11 import java.util.HashMap;
12 import java.util.Map;
13
14 public class Monitor {
15
16   public static void main(String[] args) {
17     try {
18       Socket sock = new Socket("localhost", 6106);
19       BufferedReader in = new BufferedReader(new InputStreamReader(
20           sock.getInputStream()));
21       PrintWriter out = new PrintWriter(sock.getOutputStream(), true);
22
23       String command = "routes";
24       char[] buffer = new char[100];
25       String msg = "";  // message received from socket
26
27       if (in.read(buffer) == -1){ // read the welcome message
28           System.err.println("IO Error!");
29           System.exit(-1);
30       }
31       for(int i = 0; buffer[i] != '\0'; i++)
32         buffer[i] = '\0';
```

```
33
34          // a hash map that stores all the nodes' routes
35          Map<String, String> path = new HashMap<String, String>();
36
37          out.println(command);
38          while((msg = in.readLine()) != null) {
39            msg = msg.trim();
40            if (msg.startsWith("blip")) {
41              int arrowIndex = msg.indexOf('>');
42              msg = msg.substring(arrowIndex + 1).trim();
43            }
44
45            /*****************************************************
46             *  Now we have a message that contains only the routes
47             *  information in String msg, we can extract routes
48             *  information for every node now.
49             *****************************************************/
50
51            if (msg.equals("0x64:")) {
52              out.println(command);
53            }
54            else {
55              String[] splitMsg = msg.split(":");
56
57              /*****************************************************
58               *  There is a message delay when a new node joined
59               *  the network, so we use a null detect here to prevent
60               *  array index out of bound exception.
61               *****************************************************/
62              String node = null;
63              String route = null;
64              node = splitMsg[0].trim();
65              if (splitMsg.length > 1)  // get rid of array index out of bound exception
66                route = splitMsg[1].trim();
67
68              boolean change = false;
69              if (path.containsKey(node) && route != null) {
70                String oldPath = path.get(node).split("&")[0];
71                if (!oldPath.equals(route)) {
72                  change = true;
73
74                  // get the route change time
75                  DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
76                  Date date = new Date();
77                  String changeTime = dateFormat.format(date);
78
79                  /*************************************
80                   * Write the route information to the file
81                   * ************************************/
82                  FileWriter fr = new FileWriter("route_info.log", true);
83                  BufferedWriter br = new BufferedWriter(fr);
84                  br.write("New route for node " + node + ": " + route + "\t" +
                        changeTime + "\n");
85                  br.close();
86
87                  String routeAndTime = route.concat("&").concat(changeTime);
88                  path.put(node, routeAndTime);
89                }
90              }
91              else {
92                if (route != null) {
93                  DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
94                  Date date = new Date();
95                  String changeTime = dateFormat.format(date);
96                  String routeAndTime = route.concat("&").concat(changeTime);
97                  path.put(node, routeAndTime);
98
99                  /*************************************
```

```
100                 * Write the route information to the file
101                 * ************************************/
102                FileWriter fr = new FileWriter("route_info.log", true);
103                BufferedWriter br = new BufferedWriter(fr);
104                br.write("New route for node " + node + ": " + route + " \t" +
                       changeTime + "\n");
105                br.close();
106                change = true;
107              }
108            }
109            if (change) {
110              System.out.println("Routes change at " + path.get(node).split("&")[1]);
                     // get change time
111              System.out.println("New route for " + node + ": " + route);
112            }
113          }
114        }
115      }catch (IOException e1) {
116        System.err.println(e1.getMessage());
117      }
118    }
119  }
```

C.2 Python Monitor Program

`UDPMonitor.py` measures packet loss and stores received UDP packets in the file `UDP.log` in one moving node testing.

Listing C.2: `UDPMonitor.py`

```
1  import socket
2  from datetime import datetime
3
4  port = 7000
5
6  if __name__ == '__main__':
7
8    s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
9    s.bind(('fec0::64', port))
10   transmitted = 0.0
11   received = 0.0
12   packetLoss = 0.0
13   offset = 0.0
14   timeStamp = ""
15   start = datetime.now()
16   end = datetime.now()
17   delta = end - start
18   while delta.seconds < 18000:
19     end = datetime.now()
20     delta = end - start
21     data, addr = s.recvfrom(65535)
22     if (len(data) > 0):
23       timeStamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
24       f = open('logs/UDP.log', 'a')
25       received = received + 1
26       print addr
27       print map(ord, data)
28       f.write(timeStamp + '\n')
29       f.write(str(map(ord, data)) + '\n')
```

```
30        high = map(ord, data)[0]
31        low = map(ord, data)[1]
32        transmitted = high * 256.0 + low
33        if (received == 1):
34          offset = transmitted - 1.0
35        transmitted = transmitted - offset
36        packetLoss = (transmitted - received) / transmitted
37        print "transmitted: " + str(int(transmitted))
38        print "received: " + str(int(received))
39        print "packet loss:" + str(packetLoss*100) + '%'
40        print "time elapsed: " + str(delta)
41        f.write("transmitted: " + str(int(transmitted)) + '\n')
42        f.write("received: " + str(int(received)) + '\n')
43        f.write("packet loss: " + str(packetLoss * 100) + '%' + '\n')
44        f.write('\n')
45        f.close()
```

`UDPMonitor_two.py` measures packet loss and stores received UDP packets in the file `UDP.log` in two moving node testing.

Listing C.3: `UDPMonitor_tow.py`

```
1  import socket
2  from datetime import datetime
3
4  port = 7000
5
6  if __name__ == '__main__':
7
8    s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
9    s.bind(('fec0::64', port))
10   transmitted1 = 0.0
11   received1 = 0.0
12   packetLoss1 = 0.0
13   offset1 = 0.0
14   transmitted8 = 0.0
15   received8 = 0.0
16   packetLoss8 = 0.0
17   totalPacketLoss = 0.0
18   offset8 = 0.0
19   timeStamp = ""
20   start = datetime.now()
21   end = datetime.now()
22   delta = end - start
23   while delta.seconds < 18000:
24     end = datetime.now()
25     delta = end - start
26     data, addr = s.recvfrom(65535)
27     if (len(data) > 0):
28       timeStamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
29       f = open('logs/UDP.log', 'a')
30       if (addr[0] == 'fec0::1'):
31         received1 = received1 + 1
32       elif (addr[0] == 'fec0::8'):
33         received8 = received8 + 1
34       print addr
35       print map(ord, data)
36       f.write(timeStamp + '\n')
37       f.write(str(map(ord, data)) + '\n')
38       if (addr[0] == 'fec0::1'):
39         high1 = map(ord, data)[0]
40         low1 = map(ord, data)[1]
41         transmitted1 = high1 * 256.0 + low1
42       elif (addr[0] == 'fec0::8'):
43         high8 = map(ord, data)[0]
44         low8 = map(ord, data)[1]
```

```
45          transmitted8 = high8 * 256.0 + low8
46        if (addr[0] == 'fec0::1' and received1 == 1):
47          offset1 = transmitted1 - 1.0
48        if (addr[0] == 'fec0::8' and received8 == 1):
49          offset8 = transmitted8 - 1.0
50        if (addr[0] == 'fec0::1'):
51          transmitted1 = transmitted1 - offset1
52          print offset1
53        elif (addr[0] == 'fec0::8'):
54          transmitted8 = transmitted8 - offset8
55          print offset8
56        if (addr[0] == 'fec0::1'):
57          packetLoss1 = (transmitted1 - received1) / transmitted1
58        elif (addr[0] == 'fec0::8'):
59          packetLoss8 = (transmitted8 - received8) / transmitted8
60        if (addr[0] == 'fec0::1'):
61          print "fec0::1 transmitted: " + str(int(transmitted1))
62          print "fec0::1 received: " + str(int(received1))
63          print "fec0::1 packet loss:" + str(packetLoss1*100) + '%'
64          f.write("fec0::1 transmitted: " + str(int(transmitted1)) + '\n')
65          f.write("fec0::1 received: " + str(int(received1)) + '\n')
66          f.write("fec0::1 packet loss: " + str(packetLoss1 * 100) + '%' + '\n')
67        elif (addr[0] == 'fec0::8'):
68          print "fec0::8 transmitted: " + str(int(transmitted8))
69          print "fec0::8 received: " + str(int(received8))
70          print "fec0::8 packet loss:" + str(packetLoss8*100) + '%'
71          f.write("fec0::8 transmitted: " + str(int(transmitted8)) + '\n')
72          f.write("fec0::8 received: " + str(int(received8)) + '\n')
73          f.write("fec0::8 packet loss: " + str(packetLoss8 * 100) + '%' + '\n')
74        print "total transmitted: " + str(int(transmitted1 + transmitted8))
75        print "total received: " + str(int(received1 + received8))
76        print "total packet loss: " + str(((transmitted1 + transmitted8 - received1 -
              received8)
77        / (transmitted1 + transmitted8)) * 100) + '%'
78        f.write("total transmitted: " + str(int(transmitted1 + transmitted8)) + '\n')
79        f.write("total received: " + str(int(received1 + received8)) + '\n')
80        f.write("total packet loss: " + str(((transmitted1 + transmitted8 - received1
              - received8)
81        / (transmitted1 + transmitted8)) * 100) + '%' + '\n')
82        f.write('\n')
83        print "time elapsed: " + str(delta)
84        f.close()
```

Appendix D

WSNWeb Application

D.1 Server Side Source Code

`WsnServerServletContextListener.java` is the application lifecycle listener implementation for start and stop Embedding Jetty Server configured to manage WebSocket.

Listing D.1: `WsnServerServletContextListener.java`

```
1  import javax.servlet.ServletContextEvent;
2  import javax.servlet.ServletContextListener;
3
4  import org.eclipse.jetty.server.Server;
5  import org.eclipse.jetty.server.handler.DefaultHandler;
6
7  /**
8   * Application Lifecycle Listener implementation for start/stop Embedding Jetty
9   * Server configured to manage WebSocket.
10  */
11 public class WsnServerServletContextListener implements ServletContextListener {
12
13   private Server server = null;
14
15   /**
16    * Start Embedding Jetty server when WEB Application is started.
17    *
18    */
19   public void contextInitialized(ServletContextEvent event) {
20     try {
21       // 1) Create a Jetty server with the 8081 port.
22       this.server = new Server(8081);
23       // 2) Register WsnWebSocketHandler in the Jetty server instance.
24       WsnWebSocketHandler wsnWebSocketHandler = new WsnWebSocketHandler();
25       wsnWebSocketHandler.setHandler(new DefaultHandler());
26       server.setHandler(wsnWebSocketHandler);
27       // 2) Start the Jetty server.
28       server.start();
29     } catch (Throwable e) {
30       e.printStackTrace();
```

```
31      }
32    }
33
34    /**
35     * Stop Embedding Jetty server when WEB Application is stopped.
36     */
37    public void contextDestroyed(ServletContextEvent event) {
38      if (server != null) {
39        try {
40          // stop the Jetty server.
41          server.stop();
42        } catch (Exception e) {
43          e.printStackTrace();
44        }
45      }
46    }
47
48  }
```

`WsnWebSocketHandler.java` handles WebSocket connections and creates two threads to send route topology and sensor readings information to all clients.

Listing D.2: `WsnWebSocketHandler.java`

```
1  import java.io.BufferedReader;
2  import java.io.BufferedWriter;
3  import java.io.FileWriter;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.io.PrintWriter;
7  import java.net.DatagramPacket;
8  import java.net.DatagramSocket;
9  import java.net.InetAddress;
10 import java.net.Socket;
11 import java.net.SocketException;
12 import java.net.UnknownHostException;
13 import java.text.DateFormat;
14 import java.text.DecimalFormat;
15 import java.text.SimpleDateFormat;
16 import java.util.Date;
17 import java.util.HashMap;
18 import java.util.Map;
19 import java.util.Set;
20 import java.util.concurrent.CopyOnWriteArraySet;
21 import javax.servlet.http.HttpServletRequest;
22 import org.eclipse.jetty.websocket.WebSocket;
23 import org.eclipse.jetty.websocket.WebSocketHandler;
24
25 public class WsnWebSocketHandler extends WebSocketHandler {
26
27   private final Set<WsnWebSocket> webSockets = new CopyOnWriteArraySet<WsnWebSocket
         >();
28
29   private void broadcast(String msg) {
30     for (WsnWebSocket _webSocket : webSockets) {
31       try {
32         _webSocket.connection.sendMessage(msg);
33       } catch (IOException e) {
34         // TODO Auto-generated catch block
35         System.err
36             .println("Error when sending message to client's websocket!");
37         e.printStackTrace();
38       }
39     }
40   }
41
```

```
42  /*******************************************************
43   * Thread that handles the TCP Socket connection, and
44   * send message to all clients
45   *******************************************************/
46
47  Runnable tcpSocketComm = new Runnable() {
48
49    @Override
50    public void run() {
51      try {
52        Socket sock = new Socket("ib214m06.cs.unb.ca", 6106);
53        BufferedReader in = new BufferedReader(new InputStreamReader(
54            sock.getInputStream()));
55        PrintWriter out = new PrintWriter(sock.getOutputStream(), true);
56
57        String command = "routes";
58        char[] buffer = new char[100];
59        String msg = ""; // message received from socket
60
61        if (in.read(buffer) == -1) { // read the welcome message
62          System.err.println("IO Error!");
63          System.exit(-1);
64        }
65        for (int i = 0; buffer[i] != '\0'; i++)
66          buffer[i] = '\0';
67
68        // a hash map that stores all the nodes' routes
69        Map<String, String> path = new HashMap<String, String>();
70
71        out.println(command);
72        while ((msg = in.readLine()) != null) {
73          msg = msg.trim();
74          if (msg.startsWith("blip")) {
75            int arrowIndex = msg.indexOf('>');
76            msg = msg.substring(arrowIndex + 1).trim();
77          }
78
79          /*******************************************************
80           * Now we have a message that contains only the routes
81           * information in String msg, we can extract routes
82           * information for every node now.
83           *******************************************************/
84
85          if (msg.equals("0x64:")) {
86            out.println(command);
87          } else {
88            String[] splitMsg = msg.split(":");
89
90            /*******************************************************
91             * There is a message delay when a new node joined the
92             * network, so we use a null detect here to prevent
93             * array index out of bound exception.
94             *******************************************************/
95            String node = null;
96            String route = null;
97            node = splitMsg[0].trim();
98            if (splitMsg.length > 1) // get rid of array index out
99                                     // of bound exception
100             route = splitMsg[1].trim();
101
102           boolean change = false;
103           if (path.containsKey(node) && route != null) {
104             String oldPath = path.get(node).split("&")[0];
105             if (!oldPath.equals(route)) {
106               change = true;
107
108               // get the route change time
109               DateFormat dateFormat = new SimpleDateFormat(
```

```
110                     "yyyy/MM/dd HH:mm:ss");
111                 Date date = new Date();
112                 String changeTime = dateFormat.format(date);
113
114                 /*************************************
115                  * Write the route information to the file
116                  * ************************************/
117                 FileWriter fr = new FileWriter(
118                     "route_info.log", true);
119                 BufferedWriter br = new BufferedWriter(fr);
120                 br.write("New route for node " + node + ": "
121                     + route + "\t" + changeTime + "\n");
122                 br.close();
123
124                 String routeAndTime = route.concat("&").concat(
125                     changeTime);
126                 path.put(node, routeAndTime);
127               }
128             } else {
129               if (route != null) {
130                 DateFormat dateFormat = new SimpleDateFormat(
131                     "yyyy/MM/dd HH:mm:ss");
132                 Date date = new Date();
133                 String changeTime = dateFormat.format(date);
134                 String routeAndTime = route.concat("&").concat(
135                     changeTime);
136                 path.put(node, routeAndTime);
137
138                 /*************************************
139                  * Write the route information to the file
140                  * ************************************/
141                 FileWriter fr = new FileWriter(
142                     "route_info.log", true);
143                 BufferedWriter br = new BufferedWriter(fr);
144                 br.write("New route for node " + node + ": "
145                     + route + " \t" + changeTime + "\n");
146                 br.close();
147
148                 change = true;
149
150               }
151             }
152             if (change) {
153               System.out.println("Routes change at "
154                   + path.get(node).split("&")[1]); // get
155                                                    // change
156                                                    // time
157               System.out.println("New route for " + node + ": "
158                   + route);
159               // broadcast to all clients
160               //broadcast("Routes change at "
161               //    + path.get(node).split("&")[1]);
162               broadcast("New route for " + node + ": " + route);
163               Set<Map.Entry<String, String>> entries = path
164                   .entrySet();
165               for (Map.Entry<String, String> entry : entries) {
166                 String key = entry.getKey();
167                 String value = entry.getValue();
168                 String nodePath = value.split("&")[0];
169                 broadcast(key + ": " + nodePath);
170               }
171
172             }
173
174           }
175         }
176       } catch (IOException e1) {
177         System.err.println(e1.getMessage());
```

```
178         }
179       }
180     };
181
182     /*******************************************************
183      * Thread that handles the UDP Socket connection, and
184      * send message to all clients
185      *******************************************************/
186
187     Runnable udpSocketComm = new Runnable() {
188       @Override
189       public void run() {
190         byte[] receiveData = new byte[141]; // UDP packet payload size is 141 bytes
191         int sender;
192         int seqno;
193         float light;
194         float temperature;
195         float humidity;
196         float voltage;
197         DecimalFormat df = new DecimalFormat("#.00");
198         try {
199           InetAddress IPAddress = InetAddress.getByName("fec0::64");
200           DatagramSocket udp = new DatagramSocket(7000, IPAddress);
201           while (true) {
202             DatagramPacket packet = new DatagramPacket(receiveData,
203                 receiveData.length);
204             udp.receive(packet);
205             byte[] recvByte = packet.getData();
206             for (int i = 0; i < recvByte.length; i++) {
207               int value = ((int) recvByte[i]) & 0xff; // convert byte
208                                                       // to int
209               System.out.print(value + ",");
210             }
211             System.out.print('\n');
212
213             int senderHigh = ((int) recvByte[2]) & 0xff;
214             int senderLow = ((int) recvByte[3]) & 0xff;
215             sender = senderHigh + senderLow;
216             System.out.println("sender is " + sender);
217             //broadcast("sender is " + sender);
218
219             int seqHigh = ((int) recvByte[0]) & 0xff;
220             int seqLow = ((int) recvByte[1]) & 0xff;
221             seqno = seqHigh * 256 + seqLow;
222             System.out.println("packet sequence number is " + seqno);
223             //broadcast("packet sequence number is " + seqno);
224
225             int lightHigh = ((int) recvByte[13]) & 0xff;
226             int lightLow = ((int) recvByte[14]) & 0xff;
227             float lightRead = lightHigh * 256 + lightLow;
228             light = (float) (625 * 1000000 * (lightRead * 1.5 / 4096) / 100000);
229             System.out.println("light is " + df.format(light) + "lux");
230             broadcast("light" + sender + "," + df.format(light));
231
232             int tempHigh = ((int) recvByte[17]) & 0xff;
233             int tempLow = ((int) recvByte[18]) & 0xff;
234             int tempRead = tempHigh * 256 + tempLow;
235             temperature = (float) (tempRead * 0.01 - 40);
236             System.out.println("temperature is " + df.format(temperature));
237             broadcast("temperature" + sender + "," + df.format(temperature));
238
239             int humHigh = ((int) recvByte[19]) & 0xff;
240             int humLow = ((int) recvByte[20]) & 0xff;
241             int humRead = humHigh * 256 + humLow;
242             humidity = (float) (humRead * humRead * (-0.0000028)
243                 + 0.0405 * humRead - 4);
244             System.out.println("humidity is " + df.format(humidity) + "%");
245             broadcast("humidity" + sender + "," + df.format(humidity));
```

```
246
247             int volHigh = ((int) recvByte[21]) & 0xff;
248             int volLow = ((int) recvByte[22]) & 0xff;
249             float volRead = volHigh * 256 + volLow;
250             voltage = volRead * 3 / 4096;
251             System.out.println("voltage is " + df.format(voltage) + "V");
252             broadcast("voltage" + sender + "," + df.format(voltage));
253           }
254         } catch (UnknownHostException e) {
255           // TODO Auto-generated catch block
256           e.printStackTrace();
257         } catch (SocketException e) {
258           // TODO Auto-generated catch block
259           e.printStackTrace();
260         } catch (IOException e) {
261           // TODO Auto-generated catch block
262           e.printStackTrace();
263         }
264       }
265     };
266
267     public WsnWebSocketHandler() {
268       super();
269       Thread tcpSocketThread = new Thread(tcpSocketComm);
270       tcpSocketThread.start();
271       Thread udpSocketThread = new Thread(udpSocketComm);
272       udpSocketThread.start();
273     }
274
275     public WebSocket doWebSocketConnect(HttpServletRequest request,
276         String protocol) {
277       return new WsnWebSocket();
278     }
279
280     private class WsnWebSocket implements WebSocket.OnTextMessage {
281
282       private Connection connection;
283
284       public void onOpen(Connection connection) {
285         // Client (Browser) WebSockets has opened a connection.
286         // 1) Store the opened connection
287         this.connection = connection;
288         // 2) Add WsnWebSocket in the global list of WsnWebSocket
289         // instances
290         // instance.
291         webSockets.add(this);
292       }
293
294       public void onMessage(String data) {
295         // Loop for each instance of WsnWebSocket to send message server to
296         // each client WebSockets.
297         try {
298           for (WsnWebSocket webSocket : webSockets) {
299             // send a message to the current client WebSocket.
300             webSocket.connection.sendMessage(data);
301           }
302         } catch (IOException x) {
303           // Error was detected, close the WsnWebSocket client side
304           this.connection.close();
305         }
306       }
307
308       public void onClose(int closeCode, String message) {
309         // Remove WsnWebSocket in the global list of WsnWebSocket
310         // instance.
311         webSockets.remove(this);
312       }
313     }
```

```
314 }
```

D.2 Client Side Source Code

`Util.js` contains a HashTable class in JavaScript.

Listing D.3: `Util.js`

```
1  /*Hash table class*/
2  function HashTable(obj)
3  {
4      this.length = 0;
5      this.items = {};
6      for (var p in obj) {
7          if (obj.hasOwnProperty(p)) {
8              this.items[p] = obj[p];
9              this.length++;
10         }
11     }
12
13     this.put = function(key, value)
14     {
15         var previous = undefined;
16         if (this.hasItem(key)) {
17             previous = this.items[key];
18         }
19         else {
20             this.length++;
21         }
22         this.items[key] = value;
23         return previous;
24     };
25
26     this.get = function(key) {
27         return this.hasItem(key) ? this.items[key] : undefined;
28     };
29
30     this.hasItem = function(key)
31     {
32         return this.items.hasOwnProperty(key);
33     };
34
35     this.remove = function(key)
36     {
37         if (this.hasItem(key)) {
38             previous = this.items[key];
39             this.length--;
40             delete this.items[key];
41             return previous;
42         }
43         else {
44             return undefined;
45         }
46     };
47
48     this.keys = function()
49     {
50         var keys = [];
51         for (var k in this.items) {
52             if (this.hasItem(k)) {
```

```
                keys.push(k);
            }
        }
        return keys;
    };

    this.values = function()
    {
        var values = [];
        for (var k in this.items) {
            if (this.hasItem(k)) {
                values.push(this.items[k]);
            }
        }
        return values;
    };

    this.each = function(fn) {
        for (var k in this.items) {
            if (this.hasItem(k)) {
                fn(k, this.items[k]);
            }
        }
    };

    this.clear = function()
    {
        this.items = {};
        this.length = 0;
    };
}
```

`wsn.html` is the user interface of WSNWeb application, which contains the code for receiving route topology and sensor readings information from the server and animation.

Listing D.4: `wsn.html`

```
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:vml="urn:schemas-microsoft-com:vml">
<head>
<title>WSN Web</title>
<link href="style.css" rel="stylesheet" type="text/css" media="screen" />
<script type="text/javascript" src="./jsgl.min.js"></script>
<script type="text/javascript" src="Util.js"></script>
</head>
<body>
  <div id="wrapper">
    <div id="header" class="container">
      <div id="logo">
        <img src="images/UNBlogo.png" alt="UNB Logo" width="260" height="90" />
      </div>
      <div id="menu">
        <ul>
          <li><a href="http://131.202.243.6:8080/HomePage">Homepage</a></li>
          <li><a href="http://131.202.243.6:8080/WSNWeb">WSNWeb</a></li>
        </ul>
      </div>
    </div>
    <div id="apptitle">
      <h1 class="blocktext">Mobile Sensor Network Communication</h1>

    </div>

```

```
28      <!-- end #header -->
29      <div id="Page">
30        <div id="textFieldWrapper">
31          <font color="white" size="4">Dynamic update of new routes</font>
32          <div id="textField"></div>
33        </div>
34        <div id="sensorReadingsWrapper">
35          <font color="white" size="4">Sensor readings</font>
36          <div id="sensorReadings">
37            <div id="node1">
38              <label id="node1label">node 0x1</label>
39              <div id="node1temperature">
40                <font color="white">temperature</font>
41                <div id="temperatureValue1"></div>
42              </div>
43              <div id="node1humidity">
44                <font color="white">humidity</font>
45                <div id="humidityValue1"></div>
46              </div>
47              <div id="node1light">
48                <font color="white">light</font>
49                <div id="lightValue1"></div>
50              </div>
51              <div id="node1voltage">
52                <font color="white">voltage</font>
53                <div id="voltageValue1"></div>
54              </div>
55            </div>
56            <div id="node8">
57              <label id="node8label">node 0x8</label>
58              <div id="node8temperature">
59                <font color="white">temperature</font>
60                <div id="temperatureValue8"></div>
61              </div>
62              <div id="node8humidity">
63                <font color="white">humidity</font>
64                <div id="humidityValue8"></div>
65              </div>
66              <div id="node8light">
67                <font color="white">light</font>
68                <div id="lightValue8"></div>
69              </div>
70              <div id="node8voltage">
71                <font color="white">voltage</font>
72                <div id="voltageValue8"></div>
73              </div>
74            </div>
75          </div>
76          <div id="parameterPanel">
77            <h1>WSN parameters</h1>
78            <p>power level: -25dBm <br><br>
79               RTUP: 0.6 seconds<br><br>
80               Speed steps: 80</p>
81          </div>
82        </div>
83        <div id="canvasFieldWrapper">
84          <font color="white" size="4">WSN testbed in ITB214 lab</font>
85          <div id="canvasField">
86            <canvas id="nodeCanvas" width="400" height="500"></canvas>
87            <!-- canvas id="lineCanvas" width="600" height="400"></canvas> -->
88          </div>
89          <div id="toggleButton">
90            <button id="toggleBt" type="button" onclick="toggleRoutes()">show
91              stationary nodes routes</button>
92          </div>
93        </div>
94
95      </div>
```

```
96  </div>
97  <script type="text/javascript">
98    /* hash map for storing current route information*/
99    var map = {};
100   var Htable = new HashTable(map);
101   /* drawing */
102   var nodeCanvas = document.getElementById("nodeCanvas");
103   //var lineCanvas = document.getElementById("nodeCanvas");
104   var nc = nodeCanvas.getContext("2d");
105   //var lc = lineCanvas.getContext("2d");
106
107   if (!window.WebSocket)
108     alert("WebSocket not supported by this browser");
109
110   function $() {
111     return document.getElementById(arguments[0]);
112   }
113   function $F() {
114     return document.getElementById(arguments[0]).value;
115   }
116
117   function getKeyCode(ev) {
118     if (window.event)
119       return window.event.keyCode;
120     return ev.keyCode;
121   }
122
123   /* Instantiate JSGL Panel. */
124   myPanel = new jsgl.Panel(document.getElementById("canvasField"));
125
126   /* Start drawing! */
127
128   /* draw walls */
129   var centerWall = myPanel.createRectangle();
130   centerWall.setWidth(150);
131   centerWall.setHeight(20);
132   centerWall.setLocationXY(125, 270);
133   myPanel.addElement(centerWall);
134   centerWall.getFill().setColor("rgb(100,100,100)");
135
136   var leftWall = myPanel.createRectangle();
137   leftWall.setWidth(55);
138   leftWall.setHeight(20);
139   leftWall.setLocationXY(0, 270);
140   myPanel.addElement(leftWall);
141   leftWall.getFill().setColor("rgb(100,100,100)");
142
143   var rightWall = myPanel.createRectangle();
144   rightWall.setWidth(60);
145   rightWall.setHeight(20);
146   rightWall.setLocationXY(348, 270);
147   myPanel.addElement(rightWall);
148   rightWall.getFill().setColor("rgb(100,100,100)");
149
150   /* draw lines */
151
152   var polyline0 = myPanel.createPolyline();
153   var polyline1 = myPanel.createPolyline();
154   with (polyline1.getStroke()) {
155     setColor('rgb(0,0,255)'); // red color
156     setWeight(3); // 3px width
157     setDashStyle(jsgl.DashStyles.DASH); // dashed
158     setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
159     setOpacity(0.7); // 30% transparent
160   }
161   var polyline2 = myPanel.createPolyline();
162   with (polyline2.getStroke()) {
163     setColor('rgb(255,0,0)'); // red color
```

```
164         setWeight(2); // 2px width
165         setDashStyle(jsgl.DashStyles.DASH); // dashed
166         setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
167         setOpacity(0.5); // 50% transparent
168       }
169       var polyline3 = myPanel.createPolyline();
170       with (polyline3.getStroke()) {
171         setColor('rgb(255,0,0)'); // red color
172         setWeight(2); // 2px width
173         setDashStyle(jsgl.DashStyles.DASH); // dashed
174         setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
175         setOpacity(0.5); // 50% transparent
176       }
177       var polyline4 = myPanel.createPolyline();
178       with (polyline4.getStroke()) {
179         setColor('rgb(255,0,0)'); // red color
180         setWeight(2); // 2px width
181         setDashStyle(jsgl.DashStyles.DASH); // dashed
182         setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
183         setOpacity(0.5); // 50% transparent
184       }
185       var polyline5 = myPanel.createPolyline();
186       with (polyline5.getStroke()) {
187         setColor('rgb(255,0,0)'); // red color
188         setWeight(2); // 2px width
189         setDashStyle(jsgl.DashStyles.DASH); // dashed
190         setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
191         setOpacity(0.5); // 50% transparent
192       }
193       var polyline6 = myPanel.createPolyline();
194       with (polyline6.getStroke()) {
195         setColor('rgb(255,0,0)'); // red color
196         setWeight(2); // 2px width
197         setDashStyle(jsgl.DashStyles.DASH); // dashed
198         setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
199         setOpacity(0.5); // 50% transparent
200       }
201       var polyline7 = myPanel.createPolyline();
202       with (polyline7.getStroke()) {
203         setColor('rgb(255,0,0)'); // red color
204         setWeight(2); // 2px width
205         setDashStyle(jsgl.DashStyles.DASH); // dashed
206         setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
207         setOpacity(0.5); // 50% transparent
208       }
209       var polyline8 = myPanel.createPolyline();
210       with (polyline8.getStroke()) {
211         setColor('rgb(0,0,255)'); // red color
212         setWeight(3); // 3px width
213         setDashStyle(jsgl.DashStyles.DASH); // dashed
214         setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
215         setOpacity(0.7); // 30% transparent
216       }
217
218       var polylines = new Array();
219       polylines.push(polyline0); // fake polyline
220       polylines.push(polyline1);
221       polylines.push(polyline2);
222       polylines.push(polyline3);
223       polylines.push(polyline4);
224       polylines.push(polyline5);
225       polylines.push(polyline6);
226       polylines.push(polyline7);
227       polylines.push(polyline8);
228
229       myPanel.addElement(polyline1);
230       /*
231       myPanel.addElement(polyline2);
```

```
232     myPanel.addElement(polyline3);
233     myPanel.addElement(polyline4);
234     myPanel.addElement(polyline5);
235     myPanel.addElement(polyline6);
236     myPanel.addElement(polyline7);
237      */
238     myPanel.addElement(polyline8);
239
240     /* Draw stationary nodes */
241     node2 = myPanel.createCircle();
242     node2.setCenterLocationXY(20, 20);
243     node2.setRadius(5);
244     node2.getStroke().setWeight(5);
245     node2.getStroke().setColor("rgb(255,0,0)");
246     myPanel.addElement(node2);
247
248     node3 = myPanel.createCircle();
249     node3.setCenterLocationXY(380, 20);
250     node3.setRadius(5);
251     node3.getStroke().setWeight(5);
252     node3.getStroke().setColor("rgb(255,0,0)");
253     myPanel.addElement(node3);
254
255     node4 = myPanel.createCircle();
256     node4.setCenterLocationXY(20, 250);
257     node4.setRadius(5);
258     node4.getStroke().setWeight(5);
259     node4.getStroke().setColor("rgb(255,0,0)");
260     myPanel.addElement(node4);
261
262     node5 = myPanel.createCircle();
263     node5.setCenterLocationXY(380, 250);
264     node5.setRadius(5);
265     node5.getStroke().setWeight(5);
266     node5.getStroke().setColor("rgb(255,0,0)");
267     myPanel.addElement(node5);
268
269     node6 = myPanel.createCircle();
270     node6.setCenterLocationXY(20, 480);
271     node6.setRadius(5);
272     node6.getStroke().setWeight(5);
273     node6.getStroke().setColor("rgb(255,0,0)");
274     myPanel.addElement(node6);
275
276     node7 = myPanel.createCircle();
277     node7.setCenterLocationXY(380, 480);
278     node7.setRadius(5);
279     node7.getStroke().setWeight(5);
280     node7.getStroke().setColor("rgb(255,0,0)");
281     myPanel.addElement(node7);
282
283     /* Draw moving nodes */
284
285     node1 = myPanel.createCircle();
286     node1.setRadius(5);
287     node1.getStroke().setWeight(5);
288     node1.getStroke().setColor("rgb(0,0,255)");
289     myPanel.addElement(node1);
290
291     node8 = myPanel.createCircle();
292     node8.setRadius(5);
293     node8.getStroke().setWeight(5);
294     node8.getStroke().setColor("rgb(0,255,0)");
295     myPanel.addElement(node8);
296
297     /* Draw edge router 0x64 */
298     node0 = myPanel.createCircle();
299     node0.setCenterLocationXY(50, 250);
```

```
300     node0.setRadius(5);
301     node0.getStroke().setWeight(5);
302     node0.getStroke().setColor("rgb(100,100,100)");
303     myPanel.addElement(node0);
304
305     var nodes = new Array();
306     nodes.push(node0); // edge router
307     nodes.push(node1); // moving node1
308     nodes.push(node2);
309     nodes.push(node3);
310     nodes.push(node4);
311     nodes.push(node5);
312     nodes.push(node6);
313     nodes.push(node7);
314     nodes.push(node8); // moving node8
315
316     var label1 = myPanel.createLabel();
317     var label2 = myPanel.createLabel();
318     var label3 = myPanel.createLabel();
319     var label4 = myPanel.createLabel();
320     var label5 = myPanel.createLabel();
321     var label6 = myPanel.createLabel();
322     var label7 = myPanel.createLabel();
323     var label8 = myPanel.createLabel();
324     var label0 = myPanel.createLabel();
325
326     label1.setText("0x1");
327     label2.setText("0x2");
328     label3.setText("0x3");
329     label4.setText("0x4");
330     label5.setText("0x5");
331     label6.setText("0x6");
332     label7.setText("0x7");
333     label8.setText("0x8");
334     label0.setText("edge router: 0x64");
335
336     label0.setLocationXY(node0.getCenterX() + 20, node0.getCenterY() - 5);
337     myPanel.addElement(label0);
338
339     /* mouse events */
340     var node1label = function() {
341       label1.setLocationXY(node1.getCenterX(), node1.getCenterY() + 15);
342       myPanel.addElement(label1);
343     };
344     var node1labelout = function() {
345       myPanel.removeElement(label1);
346     }
347     node1.addMouseOverListener(node1label);
348     node1.addMouseOutListener(node1labelout);
349
350     label2.setLocationXY(node2.getCenterX(), node2.getCenterY() + 10);
351     myPanel.addElement(label2);
352
353     label3.setLocationXY(node3.getCenterX(), node3.getCenterY() + 10);
354     myPanel.addElement(label3);
355
356     label4.setLocationXY(node4.getCenterX(), node4.getCenterY() - 20);
357     myPanel.addElement(label4);
358
359     label5.setLocationXY(node5.getCenterX(), node5.getCenterY() - 20);
360     myPanel.addElement(label5);
361
362     label6.setLocationXY(node6.getCenterX(), node6.getCenterY() + 10);
363     myPanel.addElement(label6);
364
365     label7.setLocationXY(node7.getCenterX(), node7.getCenterY() + 10);
366     myPanel.addElement(label7);
367
```

```
368     var node8label = function() {
369       label8.setLocationXY(node8.getCenterX(), node8.getCenterY() + 15);
370       myPanel.addElement(label8);
371     };
372     var node8labelout = function() {
373       myPanel.removeElement(label8);
374     }
375     node8.addMouseOverListener(node8label);
376     node8.addMouseOutListener(node8labelout);
377
378     /* Animator */
379
380     var animator11 = new jsgl.util.Animator(); /* node1 animator */
381     var animator12 = new jsgl.util.Animator();
382     var animator13 = new jsgl.util.Animator();
383     var animator14 = new jsgl.util.Animator();
384     var animator15 = new jsgl.util.Animator();
385     var animator16 = new jsgl.util.Animator();
386     var animator17 = new jsgl.util.Animator();
387     var animator18 = new jsgl.util.Animator();
388
389     /* Animator */
390
391     var animator81 = new jsgl.util.Animator(); /* node8 animator */
392     var animator82 = new jsgl.util.Animator();
393     var animator83 = new jsgl.util.Animator();
394     var animator84 = new jsgl.util.Animator();
395     var animator85 = new jsgl.util.Animator();
396     var animator86 = new jsgl.util.Animator();
397     var animator87 = new jsgl.util.Animator();
398     var animator88 = new jsgl.util.Animator();
399
400     var shortDuration1 = 4000;
401     var speed1 = 120 / 4000;
402     var longDuration1 = 250 / speed1;
403     var curveDuration1 = 0.25 * 2 * Math.PI * 50 / speed1;
404
405     var speed8 = speed1 / 1.25;
406     var shortDuration8 = 170 / speed8;
407     var longDuration8 = 300 / speed8;
408     var curveDuration8 = 0.25 * 2 * Math.PI * 50 / speed8;
409
410     /* node1 moving */
411     animator11.setStartValue(1 * Math.PI); // left top curve
412     animator11.setEndValue(1.5 * Math.PI);
413     animator11.setDuration(curveDuration1);
414     animator11.setRepeating(true);
415
416     animator12.setStartValue(0); // top short side
417     animator12.setEndValue(120);
418     animator12.setDuration(shortDuration1);
419     animator12.setRepeating(true);
420
421     animator13.setStartValue(1.5 * Math.PI); // right top curve
422     animator13.setEndValue(2 * Math.PI);
423     animator13.setDuration(curveDuration1);
424     animator13.setRepeating(true);
425
426     animator14.setStartValue(0); // right long side
427     //animator14.setEndValue(220);
428     animator14.setEndValue(250);
429     animator14.setDuration(longDuration1);
430     animator14.setRepeating(true);
431
432     animator15.setStartValue(0 * Math.PI); // right bottom curve
433     animator15.setEndValue(0.5 * Math.PI);
434     animator15.setDuration(curveDuration1);
435     animator15.setRepeating(true);
```

```
436
437     animator16.setStartValue(0); // bottom short side
438     animator16.setEndValue(120);
439     animator16.setDuration(shortDuration1);
440     animator16.setRepeating(true);
441
442     animator17.setStartValue(0.5 * Math.PI); // left bottom curve
443     animator17.setEndValue(1 * Math.PI);
444     animator17.setDuration(curveDuration1);
445     animator17.setRepeating(true);
446
447     animator18.setStartValue(0); // left long side
448     //animator18.setEndValue(220);
449     animator18.setEndValue(250);
450     animator18.setDuration(longDuration1);
451     animator18.setRepeating(true);
452
453     /* node8 moving */
454     animator81.setStartValue(0.5 * Math.PI); // right bottom curve
455     animator81.setEndValue(0 * Math.PI);
456     animator81.setDuration(curveDuration8);
457     animator81.setRepeating(true);
458
459     animator82.setStartValue(0); // right long side
460     //animator82.setEndValue(270);
461     animator82.setEndValue(300);
462     animator82.setDuration(longDuration8);
463     animator82.setRepeating(true);
464
465     animator83.setStartValue(2 * Math.PI); // right top curve
466     animator83.setEndValue(1.5 * Math.PI);
467     animator83.setDuration(curveDuration8);
468     animator83.setRepeating(true);
469
470     animator84.setStartValue(0); // top short side
471     animator84.setEndValue(170);
472     animator84.setDuration(shortDuration8);
473     animator84.setRepeating(true);
474
475     animator85.setStartValue(1.5 * Math.PI); // left top curve
476     animator85.setEndValue(1 * Math.PI);
477     animator85.setDuration(curveDuration8);
478     animator85.setRepeating(true);
479
480     animator86.setStartValue(0); // left long side
481     //animator86.setEndValue(270);
482     animator86.setEndValue(300);
483     animator86.setDuration(longDuration8);
484     animator86.setRepeating(true);
485
486     animator87.setStartValue(1 * Math.PI); // left bottom curve
487     animator87.setEndValue(0.5 * Math.PI);
488     animator87.setDuration(curveDuration8);
489     animator87.setRepeating(true);
490
491     animator88.setStartValue(0); // bottom short side
492     animator88.setEndValue(170);
493     animator88.setDuration(shortDuration8);
494     animator88.setRepeating(true);
495
496     var lineAnimator = new jsgl.util.Animator(); // line
497     lineAnimator.setRepeating(true);
498
499     /* animation play */
500     /* node1 */
501     animator11.addEndListener(function() {
502       animator11.pause();
503       animator12.play();
```

```
504     });
505
506     animator12.addEndListener(function() {
507       animator12.pause();
508       animator13.play();
509     });
510
511     animator13.addEndListener(function() {
512       animator13.pause();
513       animator14.play();
514     });
515
516     animator14.addEndListener(function() {
517       animator14.pause();
518       animator15.play();
519     });
520
521     animator15.addEndListener(function() {
522       animator15.pause();
523       animator16.play();
524     });
525
526     animator16.addEndListener(function() {
527       animator16.pause();
528       animator17.play();
529     });
530
531     animator17.addEndListener(function() {
532       animator17.pause();
533       animator18.play();
534     });
535
536     animator18.addEndListener(function() {
537       animator18.pause();
538       animator11.play();
539     });
540
541     /* node8 */
542     animator81.addEndListener(function() {
543       animator81.pause();
544       animator82.play();
545     });
546
547     animator82.addEndListener(function() {
548       animator82.pause();
549       animator83.play();
550     });
551
552     animator83.addEndListener(function() {
553       animator83.pause();
554       animator84.play();
555     });
556
557     animator84.addEndListener(function() {
558       animator84.pause();
559       animator85.play();
560     });
561
562     animator85.addEndListener(function() {
563       animator85.pause();
564       animator86.play();
565     });
566
567     animator86.addEndListener(function() {
568       animator86.pause();
569       animator87.play();
570     });
571
```

```
572     animator87.addEndListener(function() {
573       animator87.pause();
574       animator88.play();
575     });
576
577     animator88.addEndListener(function() {
578       animator88.pause();
579       animator81.play();
580     });
581
582     var polylinesNodes = new Array(8); // store nodes of each polyline
583     for ( var i = 0; i < 8; i++) { // 0 to 7 are polylines 1 to 8
584       polylinesNodes[i] = new Array(9);
585       for ( var j = 0; j < 9; j++)
586         polylinesNodes[i][j] = -1;
587     }
588
589     lineAnimator.addStepListener(function(t) { // line-moving listener
590       for ( var i = 1; i < polylines.length; i++) {
591         for ( var j = 0; j < polylines[i].getPointsCount(); j++) {
592           polylines[i].setPointXYAt(nodes[polylinesNodes[i - 1][j]]
593               .getCenterX(), nodes[polylinesNodes[i - 1][j]]
594               .getCenterY(), j);
595         }
596       }
597     });
598
599     /* node1 */
600     animator11.addStepListener(function(t) { // left top curve
601       node1.setCenterX(140 + 50 * Math.cos(t));
602       node1.setCenterY(125 + 50 * Math.sin(t));
603     });
604
605     animator12.addStepListener(function(t) { // top short side
606       node1.setCenterX(140 + t);
607       node1.setCenterY(75);
608     });
609
610     animator13.addStepListener(function(t) { // right top curve
611       node1.setCenterX(260 + 50 * Math.cos(t));
612       node1.setCenterY(125 + 50 * Math.sin(t));
613     });
614
615     animator14.addStepListener(function(t) { // right long side
616       node1.setCenterX(310);
617       node1.setCenterY(125 + t);
618     });
619
620     animator15.addStepListener(function(t) { // right bottom curve
621       node1.setCenterX(260 + 50 * Math.cos(t));
622       node1.setCenterY(375 + 50 * Math.sin(t));
623     });
624
625     animator16.addStepListener(function(t) { // bottom short side
626       node1.setCenterX(260 - t);
627       node1.setCenterY(425);
628     });
629
630     animator17.addStepListener(function(t) { // left bottom curve
631       node1.setCenterX(140 + 50 * Math.cos(t));
632       node1.setCenterY(375 + 50 * Math.sin(t));
633     });
634
635     animator18.addStepListener(function(t) { // left long side
636       node1.setCenterX(90);
637       node1.setCenterY(375 - t);
638     });
639
```

```
640     /* node8 */
641     animator81.addStepListener(function(t) { // right bottom curve
642       node8.setCenterX(285 + 50 * Math.cos(t));
643       node8.setCenterY(400 + 50 * Math.sin(t));
644     });
645
646     animator82.addStepListener(function(t) { // right long side
647       node8.setCenterX(335);
648       node8.setCenterY(400 - t);
649     });
650
651     animator83.addStepListener(function(t) { // right top curve
652       node8.setCenterX(285 + 50 * Math.cos(t));
653       node8.setCenterY(100 + 50 * Math.sin(t));
654     });
655
656     animator84.addStepListener(function(t) { // top short side
657       node8.setCenterX(285 - t);
658       node8.setCenterY(50);
659     });
660
661     animator85.addStepListener(function(t) { // left top curve
662       node8.setCenterX(115 + 50 * Math.cos(t));
663       node8.setCenterY(100 + 50 * Math.sin(t));
664     });
665
666     animator86.addStepListener(function(t) { // left long side
667       node8.setCenterX(65);
668       node8.setCenterY(100 + t);
669     });
670
671     animator87.addStepListener(function(t) { // left bottom curve
672       node8.setCenterX(115 + 50 * Math.cos(t));
673       node8.setCenterY(400 + 50 * Math.sin(t));
674     });
675
676     animator88.addStepListener(function(t) { // bottom short side
677       node8.setCenterX(115 + t);
678       node8.setCenterY(450);
679     });
680
681     animator11.init();
682     animator81.init();
683     animator11.play();
684     animator81.play();
685     lineAnimator.init();
686     lineAnimator.play();
687
688     /* toggle button function */
689     var show = false;
690     function toggleRoutes() {
691       if (show == false) {
692         show = true;
693         $('toggleBt').innerHTML = "hide stationary nodes routes";
694         myPanel.addElement(polyline2);
695         myPanel.addElement(polyline3);
696         myPanel.addElement(polyline4);
697         myPanel.addElement(polyline5);
698         myPanel.addElement(polyline6);
699         myPanel.addElement(polyline7);
700       } else {
701         show = false;
702         $('toggleBt').innerHTML = "show stationary nodes routes";
703         myPanel.removeElement(polyline2);
704         myPanel.removeElement(polyline3);
705         myPanel.removeElement(polyline4);
706         myPanel.removeElement(polyline5);
707         myPanel.removeElement(polyline6);
```

```
708           myPanel.removeElement(polyline7);
709         }
710       }
711
712       var room = {
713         join : function() {
714           //this._username = name;
715           //var location = document.location.toString().replace('http://',
716           //     'ws://').replace('https://', 'wss://');
717           var location = "ws://131.202.243.6:8081/";
718           this._ws = new WebSocket(location);
719           this._ws.onopen = this._onopen;
720           this._ws.onmessage = this._onmessage;
721           this._ws.onclose = this._onclose;
722           this._ws.onerror = this._onerror;
723         },
724
725         _onopen : function() {
726           room._send('connected');
727         },
728
729         _onmessage : function(m) {
730           if (m.data) {
731             var text = m.data.toString();
732             /* new routes information */
733             if (text.search("New") != -1) {
734               var textField = $('textField');
735               var spanText = document.createElement('span');
736               spanText.className = 'text';
737               spanText.innerHTML = text;
738               var lineBreak = document.createElement('br');
739               textField.appendChild(spanText);
740               textField.appendChild(lineBreak);
741               textField.scrollTop = textField.scrollHeight
742                   - textField.clientHeight;
743               var route = text;
744               var routeArray = new Array();
745               var count = 0;
746               var xindex = route.indexOf("0x");
747               var endindex;
748               while (xindex != -1) {
749                 var numberStr;
750                 var number;
751                 if (count == 0) {
752                   endindex = route.indexOf(":");
753                   numberStr = route.substring(xindex + 2,
754                       endindex);
755                   number = parseInt(numberStr);
756                   routeArray[count] = number;
757                   route = route.substring(endindex + 2);
758                 } else {
759                   endindex = route.indexOf(" ");
760                   if (endindex == -1) {
761                     routeArray[count] = 64;
762                     break;
763                   } else {
764                     numberStr = route.substring(xindex + 2,
765                         endindex);
766                     number = parseInt(numberStr);
767                     routeArray[count] = number;
768                     route = route.substring(endindex + 1);
769                   }
770                 }
771                 count = count + 1;
772                 xindex = route.indexOf("0x");
773               }
774               if (routeArray.length > 0) {
775                 var currentRoute = "";
```

```
776                 for ( var i = 1; i < routeArray.length; i++) {
777                   currentRoute = currentRoute + routeArray[i]
778                       + ",";
779                 }
780                 Htable.put(routeArray[0], currentRoute);
781
782                 polyline1.clearPoints();
783                 polyline2.clearPoints();
784                 polyline3.clearPoints();
785                 polyline4.clearPoints();
786                 polyline5.clearPoints();
787                 polyline6.clearPoints();
788                 polyline7.clearPoints();
789                 polyline8.clearPoints();
790
791                 for ( var i = 1; i < 8; i++) { // clear the stored nodes of each
                        polyline
792                   for ( var j = 0; j < 9; j++)
793                     polylinesNodes[i][j] = -1;
794                 }
795
796                 var keys = Htable.keys();
797                 for ( var i = 0; i < Htable.length; i++) {
798                   var routeArray = Htable.get(keys[i]).split(",");
799                   var nodeAndRoute = new Array();
800                   nodeAndRoute[0] = parseInt(keys[i]);
801                   for ( var j = 0; j < routeArray.length - 1; j++) {
802                     if (parseInt(routeArray[j]) == 64)
803                       nodeAndRoute[j + 1] = 0;
804                     else
805                       nodeAndRoute[j + 1] = parseInt(routeArray[j]);
806                   }
807                   for ( var k = 0; k < nodeAndRoute.length; k++) {
808                     polylines[parseInt(keys[i])]
809                         .addPointXY(nodes[nodeAndRoute[k]]
810                             .getCenterX(),
811                             nodes[nodeAndRoute[k]]
812                                 .getCenterY());
813                     polylinesNodes[parseInt(keys[i]) - 1][k] = nodeAndRoute[k];
814                   }
815                 }
816
817               }
818             }
819             /* sensor readings */
820             else {
821               var commaIndex = text.indexOf(",");
822               var typeStr = text.substring(0, commaIndex);
823               var valueStr = text.substring(commaIndex + 1);
824               if (typeStr == "temperature1") { /* node1 temperature */
825                 var temp1Field = $('temperatureValue1');
826                 var spanText = document.createElement('span');
827                 spanText.className = 'text';
828                 spanText.style.color = "yellow";
829                 spanText.innerHTML = valueStr + "&deg" + "C";
830                 if (temp1Field.hasChildNodes())
831                   temp1Field.replaceChild(spanText,
832                       temp1Field.childNodes[0]);
833                 else
834                   temp1Field.appendChild(spanText);
835               } else if (typeStr == "humidity1") { /* node1 humidity */
836                 var humidity1Field = $('humidityValue1');
837                 var spanText = document.createElement('span');
838                 spanText.className = 'text';
839                 spanText.style.color = "yellow";
840                 spanText.innerHTML = valueStr + "%";
841                 if (humidity1Field.hasChildNodes())
842                   humidity1Field.replaceChild(spanText,
```

```
843                     humidity1Field.childNodes[0]);
844                 else
845                   humidity1Field.appendChild(spanText);
846               } else if (typeStr == "voltage1") { /* node1 voltage*/
847                 var voltage1Field = $('voltageValue1');
848                 var spanText = document.createElement('span');
849                 spanText.className = 'text';
850                 spanText.style.color = "yellow";
851                 spanText.innerHTML = valueStr + "V";
852                 if (voltage1Field.hasChildNodes())
853                   voltage1Field.replaceChild(spanText,
854                       voltage1Field.childNodes[0]);
855                 else
856                   voltage1Field.appendChild(spanText);
857               } else if (typeStr == "light1") { /* node1 voltage*/
858                 var light1Field = $('lightValue1');
859                 var spanText = document.createElement('span');
860                 spanText.className = 'text';
861                 spanText.style.color = "yellow";
862                 spanText.innerHTML = valueStr + "lux";
863                 if (light1Field.hasChildNodes())
864                   light1Field.replaceChild(spanText,
865                       light1Field.childNodes[0]);
866                 else
867                   light1Field.appendChild(spanText);
868               } else if (typeStr == "temperature8") { /* node8 temperature*/
869                 var temp8Field = $('temperatureValue8');
870                 var spanText = document.createElement('span');
871                 spanText.className = 'text';
872                 spanText.style.color = "yellow";
873                 spanText.innerHTML = valueStr + "&deg" + "C";
874                 if (temp8Field.hasChildNodes())
875                   temp8Field.replaceChild(spanText,
876                       temp8Field.childNodes[0]);
877                 else
878                   temp8Field.appendChild(spanText);
879               } else if (typeStr == "humidity8") { /* node8 humidity */
880                 var humidity8Field = $('humidityValue8');
881                 var spanText = document.createElement('span');
882                 spanText.className = 'text';
883                 spanText.style.color = "yellow";
884                 spanText.innerHTML = valueStr + "%";
885                 if (humidity8Field.hasChildNodes())
886                   humidity8Field.replaceChild(spanText,
887                       humidity8Field.childNodes[0]);
888                 else
889                   humidity8Field.appendChild(spanText);
890               } else if (typeStr == "light8") { /* node1 voltage*/
891                 var light8Field = $('lightValue8');
892                 var spanText = document.createElement('span');
893                 spanText.className = 'text';
894                 spanText.style.color = "yellow";
895                 spanText.innerHTML = valueStr + "lux";
896                 if (light8Field.hasChildNodes())
897                   light8Field.replaceChild(spanText,
898                       light8Field.childNodes[0]);
899                 else
900                   light8Field.appendChild(spanText);
901               } else if (typeStr == "voltage8") { /* node8 voltage */
902                 var voltage8Field = $('voltageValue8');
903                 var spanText = document.createElement('span');
904                 spanText.className = 'text';
905                 spanText.style.color = "yellow";
906                 spanText.innerHTML = valueStr + "V";
907                 if (voltage8Field.hasChildNodes())
908                   voltage8Field.replaceChild(spanText,
909                       voltage8Field.childNodes[0]);
910                 else
```

```
911                 voltage8Field.appendChild(spanText);
912               }
913             }
914           }
915         },
916
917         _onclose : function(m) {
918           this._ws = null;
919           $('textField').innerHTML = '';
920         },
921
922         _onerror : function(e) {
923           alert(e);
924         },
925
926         _send : function(message) {
927           if (this._ws)
928             this._ws.send(message);
929         }
930       };
931
932       /* method for getting route information*/
933       room.join();
934     </script>
935
936     <div id="footer-content">
937       <div id="footer-bg" class="container">
938         <div id="column1">
939           <h2>Contact</h2>
940           <p>
941             Weiqi Zhang: weiqqi1028@gmail.com<br>Bradford G. Nickerson: bgn@unb.ca<br>
                  University of New
942             Brunswick <br> Faculty of Computer Science <br>
943             Fredericton, NB. <br> Canada, E3B 5A3
944           </p>
945         </div>
946         <div id="column2">
947           <h2>Acknowledgements</h2>
948           <p>University of New Brunswick Faculty of Computer Science for
949             their support on this research.</p>
950         </div>
951         <div id="column3">
952           <h2>Associated Links</h2>
953           <ul>
954             <li><a
955               href="http://bullseye.xbow.com:81/Products/productdetails.aspx?sid=252">
956                 The Telosb sensor node</a></li>
957             <li><a
958               href="http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-17
                    ">
959                 The WebSocket Protocol</a></li>
960             <li><a
961               href="http://www.jsgl.org/doku.php?id=home">
962                 JavaScript Graphics Library (JSGL)</a></li>
963             <li><a href="http://diveintohtml5.info/canvas.html"> HTML5
964                 Canvas Element</a></li>
965             <li><a href="http://tomcat.apache.org/"> Apache Tomcat
966                 Webserver</a></li>
967           </ul>
968         </div>
969       </div>
970     </div>
971
972 </body>
973 </html>
```

Vita

Candidate's full name: Weiqi Zhang

University attended:
September 2010 - October 2013
Faculty of Computer Science
University of New Brunswick
Fredericton, New Brunswick, Canada

September 2006 - June 2010
Bachelor of Engineering
Shool of Information and Communication Engineering
Beijing University of Post and Telecommunications
Beijing, China

Technical Report:
Weiqi Zhang and Bradford G. Nickerson, "Wireless Sensor Network Communication Protocols", UNB Faculty of Computer Science, TR11-208, May 2011, Fredericton, N.B., Canada.

Poster:
Weiqi Zhang and Bradford G. Nickerson, "Poster: 6LoWPAN for mobile wireless sensor network communication", 10th UNB Computer Science Research Exposition, April 12, 2013, Fredericton, N.B., Canada.

Publications:
Conference Presentations: