Exploring an IPv6 protocol for mobile sensor network communication

by

Weiqi Zhang

Bachelor of Electronic Information Engineering, Beijing University of Post and Telecommunications

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s): Bradford G. Nickerson, Ph.D., Computer Science
Examiner Board: Wei Song, Ph.D., Computer Science,
               John DeDourek, MS, Computer Science
External Examiner: Brent R. Petersen, Ph.D.,
                   Dept. of Electrical and Computer Engineering

This thesis is accepted

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

May, 2013

©Weiqi Zhang, 2013
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 explanation 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 writing 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

iv
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
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4</td>
<td>Subset of Jetty library *.jar files used to implement a WebSocket communication.</td>
<td>13</td>
</tr>
<tr>
<td>3.6</td>
<td>The file web.xml defines Listener and startup web page.</td>
<td>13</td>
</tr>
<tr>
<td>3.7</td>
<td>Sequence diagram of starting the thread in the constructor of WSNWebSocketHandler.</td>
<td>15</td>
</tr>
<tr>
<td>3.8</td>
<td>Defining a WebSocket object in JavaScript.</td>
<td>17</td>
</tr>
<tr>
<td>3.9</td>
<td>Processing of route topology and sensor reading message received at the client.</td>
<td>19</td>
</tr>
<tr>
<td>3.5</td>
<td>Class diagram of the WSNWeb web application.</td>
<td>21</td>
</tr>
<tr>
<td>3.10</td>
<td>A screen shot of the WSNWeb application.</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>UDP interface sending packets to an IP address.</td>
<td>24</td>
</tr>
<tr>
<td>4.2</td>
<td>Sequence diagram of sendTask().</td>
<td>25</td>
</tr>
<tr>
<td>4.3</td>
<td>Sequence diagram of getNextFrag().</td>
<td>26</td>
</tr>
<tr>
<td>4.4</td>
<td>Picture of the edge router with connected USB Raven and gateway.</td>
<td>27</td>
</tr>
<tr>
<td>4.5</td>
<td>A complete UDP packet captured in Wireshark.</td>
<td>28</td>
</tr>
<tr>
<td>4.6</td>
<td>The first fragment of the UDP packet in Figure 4.5.</td>
<td>29</td>
</tr>
<tr>
<td>4.7</td>
<td>The subsequent fragment of the UDP packet in Figure 4.5.</td>
<td>29</td>
</tr>
<tr>
<td>4.8</td>
<td>The implementation of receive() event.</td>
<td>31</td>
</tr>
<tr>
<td>4.9</td>
<td>How BLIP handles first fragment when a node receives it.</td>
<td>32</td>
</tr>
<tr>
<td>4.10</td>
<td>Dual stack, integrating 6LoWPAN with IPv6, adapted from [20].</td>
<td>33</td>
</tr>
<tr>
<td>4.11</td>
<td>Wireless sensor network architecture in ITB214.</td>
<td>34</td>
</tr>
<tr>
<td>4.19</td>
<td>read_and_process reads and processes up to one 6LoWPAN packets.</td>
<td>36</td>
</tr>
<tr>
<td>4.21</td>
<td>Sequence diagram of reassembly when serial port receives 6LoWPAN packets.</td>
<td>38</td>
</tr>
<tr>
<td>4.22</td>
<td>Transmission of packets from radio to uart.</td>
<td>40</td>
</tr>
<tr>
<td>4.23</td>
<td>Transmission of packets from uart to radio.</td>
<td>40</td>
</tr>
<tr>
<td>4.12</td>
<td>Sequence diagram of setting up TCP socket and tunnel interface in the ip-driver program.</td>
<td>44</td>
</tr>
<tr>
<td>4.13</td>
<td>Sequence diagram of opening a serial port and initializing routing in ip-driver.</td>
<td>45</td>
</tr>
<tr>
<td>4.14</td>
<td>Sequence diagram of tunneling process of receiving data.</td>
<td>46</td>
</tr>
<tr>
<td>4.15</td>
<td>Sequence diagram of tun_input().</td>
<td>46</td>
</tr>
<tr>
<td>4.16</td>
<td>Sequence diagram of serial_input().</td>
<td>47</td>
</tr>
<tr>
<td>4.17</td>
<td>Definition of data type serial_source_t.</td>
<td>48</td>
</tr>
<tr>
<td>4.18</td>
<td>Definition of data type packed_lowmsg_t.</td>
<td>48</td>
</tr>
<tr>
<td>4.20</td>
<td>Definition of data type reconstruct_t.</td>
<td>48</td>
</tr>
<tr>
<td>5.1</td>
<td>Model train running on the wireless sensor network testbed in ITB214.</td>
<td>49</td>
</tr>
<tr>
<td>5.2</td>
<td>A Telosb node is carried on the model train.</td>
<td>50</td>
</tr>
<tr>
<td>5.3</td>
<td>General view of train track in ITB214.</td>
<td>51</td>
</tr>
<tr>
<td>5.4</td>
<td>Model railroad track in ITB214.</td>
<td>52</td>
</tr>
<tr>
<td>5.5</td>
<td>Model railroad track corner.</td>
<td>53</td>
</tr>
<tr>
<td>5.6</td>
<td>Packet structure of UDP over IPv6.</td>
<td>54</td>
</tr>
<tr>
<td>5.7</td>
<td>141 bytes Payload structure.</td>
<td>54</td>
</tr>
</tbody>
</table>
List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation (1st column)</th>
<th>Description (2nd column)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AHM</td>
<td>Airplane Health Management</td>
</tr>
<tr>
<td>BLIP</td>
<td>Berkeley Low-Power IP stack</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>EUI</td>
<td>Extended Unique Identifier</td>
</tr>
<tr>
<td>GHz</td>
<td>GigaHertz</td>
</tr>
<tr>
<td>HC</td>
<td>Header Compression</td>
</tr>
<tr>
<td>HTML5</td>
<td>HyperText Markup Language 5</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>ICMP</td>
<td>Internet Control Message Protocol</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IR</td>
<td>Infrared</td>
</tr>
<tr>
<td>JMRI</td>
<td>Java Model Railroad Interface</td>
</tr>
<tr>
<td>js</td>
<td>JavaScript</td>
</tr>
<tr>
<td>kB</td>
<td>kilo Bytes</td>
</tr>
<tr>
<td>LQI</td>
<td>Link Quality Identifier</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>MHz</td>
<td>MegaHertz</td>
</tr>
<tr>
<td>MTU</td>
<td>Maximum Transmission Unit</td>
</tr>
<tr>
<td>NTC</td>
<td>Number of Topology Changes</td>
</tr>
<tr>
<td>PAN</td>
<td>Personal Area Network</td>
</tr>
<tr>
<td>PL</td>
<td>Packet Loss</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>R/T</td>
<td>Received/Transmitted</td>
</tr>
<tr>
<td>RTUP</td>
<td>Routing Table Update Period</td>
</tr>
<tr>
<td>SI</td>
<td>International System of Units</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TOS</td>
<td>TinyOS</td>
</tr>
<tr>
<td>tun</td>
<td>tunnel</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver/Transmitter</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UNB</td>
<td>University of New Brunswick</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>ws</td>
<td>WebSocket</td>
</tr>
<tr>
<td>WSN</td>
<td>Wireless Sensor Network</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>6LoWPAN</td>
<td>IPv6 over Low power Wireless Personal Area Networks</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>23 Bytes</th>
<th>21 Bytes</th>
<th>40 Bytes</th>
<th>8 Bytes</th>
<th>33 Bytes</th>
<th>2 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC header</td>
<td>Security</td>
<td>IPv6 header</td>
<td>UDP header</td>
<td>Data</td>
<td>CRC</td>
</tr>
</tbody>
</table>

IEEE 802.15.4 frame size is 127 bytes

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

MAC header | Security | Adapt header | IPv6 header | UDP header | Data | CRC

Figure 2.2: Data encapsulation.

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

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 transmitted 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.
Hui et al. [14] define an encoding format for 6LoWPAN called 6LoWPAN_IPHC. 6LoWPAN_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.

![Diagram showing 6LoWPAN addressing](image)

**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.
TinyOS developers use the Nesdoc tool to display the structure and composition of nesc applications. For example, to show the structure of a 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.
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 adapted 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-
Apache Tomcat server
Jetty library *.jar files
WSNWeb
WsnServerServletContextListener.class
registered
WsnWebSocketHandler.class
wsn.html
style.css
jsoc.min.js
Util.js
web.xml
ip-driver

HTTP requests on port 8080:
wsn.html
style.css
jsoc.min.js
Util.js

Internet

Web Browser

wsn.html

Telosb sensor node

IPBaseStation

Topology changes
TCP Socket on port 6106

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:

```java
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.
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 \((\text{int})\text{recvByte[14]} \& \text{0xff}\) 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 receives 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>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>join</td>
<td>Create WebSocket connection to the server</td>
</tr>
<tr>
<td>onopen</td>
<td>implementation of WebSocket <code>onopen</code> method, sends &quot;connected&quot; to the server when successfully connected</td>
</tr>
<tr>
<td>onmessage</td>
<td>implementation of WebSocket <code>onmessage</code> method, interacts to incoming messages</td>
</tr>
<tr>
<td>onclos e</td>
<td>implementation of WebSocket <code>onclose</code> method, close the established WebSocket connection</td>
</tr>
<tr>
<td>onerror</td>
<td>implementation of WebSocket <code>onerror</code> method, display error messages</td>
</tr>
<tr>
<td>send</td>
<td>sends messages to the server</td>
</tr>
</tbody>
</table>
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()`.

![Sequence diagram of sendTask()](image)

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.
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 of Raven USB sniffer to 15 where /dev/hidraw2 is the device name of the sniffer:

```bash
./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.](image)

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

```c
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:

```c
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.
4.2 ip-driver Software Architecture

Figure 4.10 shows how 6LoWPAN 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
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:

```shell
> 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
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.

Reassembly 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:

```c
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 written 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.
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.

<table>
<thead>
<tr>
<th>interface</th>
<th>implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send</td>
<td>UartSend</td>
</tr>
<tr>
<td>Ieee154Send</td>
<td>RadioSend</td>
</tr>
<tr>
<td>Receive</td>
<td>UartReceive</td>
</tr>
<tr>
<td>Receive</td>
<td>RadioReceive</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Sensor type</th>
<th>Component in TinyOS</th>
<th>Manufacturer</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible to IR Light Sensor</td>
<td>HamamatsuS10871TsrC</td>
<td>Hamamatsu</td>
<td>Lux</td>
</tr>
<tr>
<td>Visible Light Sensor</td>
<td>HamamatsuS1087ParC</td>
<td>Hamamatsu</td>
<td>Lux</td>
</tr>
<tr>
<td>Temperature Sensor</td>
<td>SensirionSht11C</td>
<td>Sensirion</td>
<td>%</td>
</tr>
<tr>
<td>Humidity Sensor</td>
<td>SensirionSht11C</td>
<td>Sensirion</td>
<td>°C</td>
</tr>
<tr>
<td>Voltage Sensor</td>
<td>Msp430InternalVoltageC</td>
<td>Texas Instruments</td>
<td>Volts</td>
</tr>
</tbody>
</table>

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:

```c
event void ReadTSR.readDone(error_t result, uint16_t data) {
    if (result == SUCCESS) {
        stats.udp.tsr = data;
        call ReadPAR.read();
    }
}
```

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

```c
nx_struct udp_report {
    nx_uint16_6 seqno;
    nx_uint16_t sender;
    ip_statistics_t ip;
    sensor_readings_t udp;
    icmp_statistics_t icmp;
    route_statistics_t route;
};
```

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:

\[
S1087: \text{lux} = 0.769 \times 10^5 \times I \times 1000
\]  

\[4.1\]
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 kΩ, then the output current can be calculated by the following equation where \( AD_{output} \) is the sensor reading, \( V_{ref} = 1.5 \) V:

\[
I = \frac{V_{sensor}}{100000 \ \Omega}; V_{sensor} = AD_{output} \times \frac{V_{ref}}{2^{12}}
\]  
(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:

\[
humidity = -0.0000028 \times data \times data + 0.0405 \times data - 4
\]
(4.4)

\[
temperature = -40 + 0.01 \times data
\]
(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:

\[
voltage = \frac{data}{4096} \times 3
\]
(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 {
    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).

![Model train running on the wireless sensor network testbed in ITB214.](image)

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.
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.
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.
<table>
<thead>
<tr>
<th>bits</th>
<th>0 - 7</th>
<th>8 - 15</th>
<th>16 - 23</th>
<th>24 - 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Version</td>
<td>Traffic Class</td>
<td>Flow Label</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Payload Length</td>
<td>Next Header</td>
<td>Hop Limit</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>Source Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>192</td>
<td>Destination Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>320</td>
<td>Source Port Number</td>
<td>Destination Port Number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>352</td>
<td>UDP Packet Length</td>
<td>Checksum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>384</td>
<td>Payload (Payload length is 141 bytes)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.6: Packet structure of UDP over IPv6.

<table>
<thead>
<tr>
<th>seqno</th>
<th>sender</th>
<th>ip_stat</th>
<th>sensor_reading</th>
<th>ICMP_stat</th>
<th>route_stat</th>
<th>data</th>
</tr>
</thead>
</table>

Payload length is 141 bytes

Figure 5.7: 141 bytes Payload structure.
Table 5.1: UDP payload structure.

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

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 outgoing 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.

<table>
<thead>
<tr>
<th>IR light sensor reading (2 bytes)</th>
<th>visible light sensor reading (2 bytes)</th>
<th>temperature sensor reading (2 bytes)</th>
<th>humidity sensor reading (2 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>voltage sensor reading (2 bytes)</td>
<td>TOS_NODE_ID of sender (2 bytes)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

![Architecture Diagram]

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
0x5: [1.1]0x64
0x4: [1.1]0x1 [1.1]0x64
0x8: [1.0]0x64 [1.1]0x64
0x7: [1.1]0x1 [1.1]0x64
0x1: [1.1]0x64
0x64:
blip:ib214m96.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;
3. BufferedReader in;
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.

![Diagram](image)

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

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 iteration. 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:

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

The packet loss is calculated using equation (5.1),

\[
PL = 100(T - R)/T
\]  

(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:

```python
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. $T \leftarrow 0$;
2. $R \leftarrow 0$;
3. while $time\, elapsed \leq 5\, hours$ do
4.    $data \leftarrow \text{recvfrom} \, \text{"fec0::64" \, port 7000}$;
5.    print data to UDP.log;
6.    if length of $data > 0$ then
7.        $R \leftarrow R + 1$;
8.        high $\leftarrow$ first byte;
9.        low $\leftarrow$ second byte;
10.       $T \leftarrow \text{high} \times 256 + \text{low};$ /* sequence number (seqno) of transmitted packet */
11.      if $R == 1$ then /* account for the first received packet */
12.          $O \leftarrow T - 1$;
13.      end
14.      $T \leftarrow T - O$;
15.      $PL \leftarrow 100 \times (T - R)/T$;
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.

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.

<table>
<thead>
<tr>
<th>Speed Steps</th>
<th>40</th>
<th>60</th>
<th>80</th>
<th>100</th>
<th>120</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Consumed</td>
<td>15:45</td>
<td>8:09</td>
<td>5:22</td>
<td>3:58</td>
<td>3:11</td>
</tr>
<tr>
<td>Velocity (m/s)</td>
<td>0.0759</td>
<td>0.1466</td>
<td>0.2229</td>
<td>0.3015</td>
<td>0.3757</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Speed Steps</th>
<th>40</th>
<th>60</th>
<th>80</th>
<th>100</th>
<th>120</th>
</tr>
</thead>
<tbody>
<tr>
<td>Velocity (m/s)</td>
<td>0.0584</td>
<td>0.1173</td>
<td>0.1791</td>
<td>0.2469</td>
<td>0.2995</td>
</tr>
</tbody>
</table>

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 fec0:0:1 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.

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

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.

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.

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

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.

![Number of route changes vs. Train speed](image)

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


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

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 tx.total; // L2 frags received
    nx_uint8_t enc fail; // frags dropped due to send queue
```
typedef nx_struct {
    nx_uint8_t hop_limit;
    nx_uint16_t parent;
    nx_uint16_t parent_metric;
    nx_uint16_t parent_etx;
} route_statistics_t;

typedef nx_struct {
    nx_uint8_t sol_rx;
    nx_uint8_t sol_tx;
    nx_uint8_t adv_rx;
    nx_uint8_t adv tx;
    nx_uint8_t echo_rx;
    nx_uint8_t echo tx;
    nx_uint8_t unk rx;
    nx_uint16_t rx;
} icmp_statistics_t;

typedef nx_struct {
    nx_uint16_t tsr;
    nx_uint16_t par;
    nx_uint16_t exttemp;
    nx_uint16_t hum;
    nx_uint16_t volt;
    nx_uint16_t sender;
} 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

#include <6lowpan.h>
#if DBG_TRACK_FLOWS
#include "TestDriver.h"
#endif

configuration UDPMovingC {
    }
}

implementation UDPMovingP {
    UDPMovingP.Boot -> MainC;
    UDPMovingP.Leds -> LedsC;
}
components new TimerMilliC();
components IPDispatchC;

UDPMovingP.RadioControl -> IPDispatchC;
components new UdpSocketC() as Status;

UDPMovingP.Status -> Status;
UDPMovingP.StatusTimer -> TimerMilliC;

components UdpC;
UDPMovingP.IPStats -> IPDispatchC.IPStats;
UDPMovingP.UDPStats -> UdpC;
UDPMovingP.RouteStats -> IPDispatchC.RouteStats;
UDPMovingP.ICMPStats -> IPDispatchC.ICMPStats;

#ifdef SIM
components BaseStationC;
#endif
#ifdef DBGTRACKFLOW
components TestDriverP, SerialActiveMessageC as Serial;
components ICMPResponderC, IPRoutingP;
TestDriverP.Boot -> MainC;
TestDriverP.SerialControl -> Serial;
TestDriverP. ICMPPing -> ICMPResponderC. ICMPPing(unique("PING"));
TestDriverP.CmdReceive -> Serial.Receive[AM_TESTDRIVER.MSG];
TestDriverP.IPRouting -> IPRoutingP;
TestDriverP.DoneSend -> Serial.AMSend[AM_TESTDRIVER.MSG];
TestDriverP.AckSend -> Serial.AMSend[AM_TESTDRIVERACK];
TestDriverP.RadioControl -> IPDispatchC;
#endif
#ifdef DBG.FL0W8.REPORT
components TrackFlowsC;
#endif
}

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

Listing A.4: UDPMovingP.nc

```
#include <IP Dispatch.h>
#include <lib6lowpan.h>
#include <ip.h>
#include <lib6lowpan.h>
#include <ip.h>
#include "UDPReport.h"
#include "PrintfUART.h"
#define REPORT_PERIOD 75L
#define REPORT DEST "fec0::64"

module UDPMovingP {
    uses {
        interface Boot;
        interface SplitControl as RadioControl;
        interface Udp as Status;
        interface Leds;
        interface Timer<TMilli> as StatusTimer;
        interface Statistics<ip_statistics_t> as IPStats;
        //interface Statistics<udp_statistics_t> as UDPStats;
        interface Statistics<sensor_readings_t> as UDPSend;
        interface Statistics<route_statistics_t> as RouteStats;
        interface Statistics<icmp_statistics_t> as ICMPStats;
        interface Read<Read<16_t> as ReadTSR;
        interface Read<Read<16_t> as ReadPAR;
```

79
interface Read<uint16_t> as ReadExtTemp;
interface Read<uint16_t> as ReadHum;
interface Read<uint16_t> as ReadVolt;

bool timerStarted;

struct sockaddr_in6 route_dest;

event void Boot.booted() {
    timerStarted = FALSE;
    call RadioControl.start();
    call IPStats.clear();
    call RouteStats.clear();
    call ICMPStats.clear();
    printfUART_init();

    #ifdef REPORT_DEST
    route_dest.sin6_port = htonl6(7000);
    inet_pton6(REPORT_DEST, &route_dest.sin6_addr);
    //call StatusTimer.startOneShot(call Random.rand16() % (1nz_uint16_t *1024 * REPORT.PERIOD));
    call StatusTimer.startPeriodic(10000);
    #endif

    dbg("Boot", "booted: %i\n", TOS..NODE..ID);
}

event void RadioControl.startDone(error_t e) {
}

event void RadioControl.stopDone(error_t e) {
}

event void Status.recvfrom(struct sockaddr_in6 *from, void *data,
    uint16_t len, struct ip_metadata *meta) {
}

event void StatusTimer.fired() {
    if (!timerStarted) {
        timerStarted = TRUE;
    }
    call Leds.led0Toggle();
    stats.seqno++;
    stats.sender = TOS_NODE.ID;
    call IPStats.get(&stats.ip);
    call UDPStats.get(&stats.udp);
    stats.udp.sender = TOS_NODE.ID;
    call ICMPStats.get(&stats.icmp);
    call RouteStats.get(&stats.route);
    call ReadTSR.read();
    call Status.sendto(&route_dest, &stats, sizeof(stats));
}
event void ReadTSR.readDone(error_t result, uint16_t data) {
    if(result == SUCCESS) {
        stats.udp.tsr = data;
        call ReadPAR.read();
    }
}

event void ReadPAR.readDone(error_t result, uint16_t data) {
    if(result == SUCCESS) {
        stats.udp.par = data;
        call ReadExtTemp.read();
    }
}

event void ReadExtTemp.readDone(error_t result, uint16_t data) {
    if(result == SUCCESS) {
        stats.udp.exttemp = data;
        call ReadHum.read();
    }
}

event void ReadHum.readDone(error_t result, uint16_t data) {
    if(result == SUCCESS) {
        stats.udp.hum = data;
        call ReadVolt.read();
    }
}

event void ReadVolt.readDone(error_t result, uint16_t data) {
    if(result == SUCCESS) {
        stats.udp.volt = data;
    }
}
Appendix B

Stationary Node Software

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

```
Listing B.1: UDPStationaryC.nc

#include <6lowpan.h>
#ifdef DBGTRACK.FLOWS
#include "TestDriver.h"
#endif

configuration UDPStationaryC {
} implementation {
    components MainC, LedsC;
    components UDPStationaryP;
    UDPStationaryP.Boot -> MainC;
    UDPStationaryP.Leds -> LedsC;
    components new TimerMilliC();
    components IPDispatchC;
    UDPStationaryP.RadioControl -> IPDispatchC;
    components new UdpSocketC() as Status;
    UDPStationaryP.Status -> Status;
    UDPStationaryP.StatusTimer -> TimerMilliC;
    components UdpC;
    UDPStationaryP.IPStats -> IPDispatchC.IPStats;
    UDPStationaryP.UDPStats -> UdpC;
    UDPStationaryP.RouteStats -> IPDispatchC.RouteStats;
    UDPStationaryP.ICMPStats -> IPDispatchC.ICMPStats;
    components UDPShellC;
#ifdef SIM
    components BaseStationC;
#endif
#ifdef DBGTRACK.FLOWS
    components TestDriverP, SerialActiveMessageC as Serial;
    components ICMPResponderC, IPRoutingP;
#endif
    TestDriverP.Boot -> MainC;
```
41 TestDriverP. SerialControl -> Serial;
42 TestDriverP. ICMPping -> ICMPResponderC. ICMPping [ unique("PING") ];
43 TestDriverP. IPRouting -> Serial. Receive [ AM..TESTDRIVER..MSG ];
44 TestDriverP. DoneSend -> Serial. AMSend [ AM..TESTDRIVER..MSG ];
45 TestDriverP. AckSend -> Serial. AMSend [ AM..TESTDRIVER..ACK ];
46 TestDriverP. RadioControl -> IPDispatchC;
47 #ifdef
48 #if
49 #ifdef DBG.FL0W8.REPORT
50 components TrackFlowC;
51 #endif
52 }

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

Listing B.2: UDPStationaryP.nc
event void RadioControl.startDone(error_t e) {
}

event void RadioControl.stopDone(error_t e) {
}

event void Status.recvfrom(struct sockaddr_in6 *from, void *data, uint16_t len, struct ip_metadata *meta) {
}

event void StatusTimer.fired() {
  if (!timerStarted) {
    timerStarted = TRUE;
  }
}

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
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File Writer;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;

public class Monitor {

    public static void main(String[] args) {
        try {
            Socket sock = new Socket("localhost", 6106);
            BufferedReader in = new BufferedReader(new InputStreamReader(
                sock.getInputStream()));
            PrintWriter out = new PrintWriter(sock.getOutputStream(), true);
            String command = "routes";
            char[] buffer = new char[100];
            String msg = "\n"; // message received from socket
            if (in.read(buffer) == -1) { // read the welcome message
                System.err.println("IO Error!");
                System.exit(-1);
            }
            for (int i = 0; buffer[i] != '\0'; i++)
                buffer[i] = '\0';
```
Map<String, String> path = new HashMap<String, String>();

out.println(command);
while((msg = in.readLine()) != null) {
    msg = msg.trim();
    if (msg.startsWith("blip")) {
        int arrowIndex = msg.indexOf('>');
        msg = msg.substring(arrowIndex + 1).trim();
    }

    // Now we have a message that contains only the routes
    // information in String msg, we can extract routes
    // information for every node now.
    // ******************************************************************************
    if (msg.equals("0x64:")) {
        out.println(command);
    } else {
        String[] splitMsg = msg.split(":");

        // ******************************************************************************
        String node = null;
        String route = null;
        node = splitMsg[0].trim();
        if (splitMsg.length > 1) {
            route = splitMsg[1].trim();
        }

        boolean change = false;
        if (path.containsKey(node) && route != null) {
            String oldPath = path.get(node).split('&')[0];
            if (!oldPath.equals(route)) {
                change = true;

                // get the route change time
                DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
                Date date = new Date();
                String changeTime = dateFormat.format(date);

                // Write the route information to the file
                FileWriter fr = new FileWriter("route_info.log", true);
                BufferedWriter br = new BufferedWriter(fr);
                br.write("New route for node " + node + ": " + route + "\t" +
                        changeTime + "\n");
                br.close();

                String routeAndTime = route.concat("&").concat(changeTime);
                path.put(node, routeAndTime);
            } else {
                if (route != null) {
                    DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
                    Date date = new Date();
                    String changeTime = dateFormat.format(date);
                    String routeAndTime = route.concat("&").concat(changeTime);
                    path.put(node, routeAndTime);
                }
            }
        } else {
            // get rid of array index out of bound exception
            route = splitMsg[1].trim();
        }
    }
}
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
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) + '%')
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_two.py

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

88
transmitted8 = high8 * 256.0 + low8
if (addr[0] == 'fec0::1' and received1 == 1):
    offset1 = transmitted1 - 1.0
if (addr[0] == 'fec0::8' and received8 == 1):
    offset8 = transmitted8 - 1.0
if (addr[0] == 'fec0::1'):
    transmitted1 = transmitted1 - offset1
    print offset1
elif (addr[0] == 'fec0::8'):
    transmitted8 = transmitted8 - offset8
    print offset8
if (addr[0] == 'fec0::1'):
    packetLoss1 = (transmitted1 - received1) / transmitted1
elif (addr[0] == 'fec0::8'):
    packetLoss8 = (transmitted8 - received8) / transmitted8
if (addr[0] == 'fec0::1'):
    print "fec0::1 transmitted: " + str(int(transmitted1))
    print "fec0::1 received: " + str(int(received1))
    print "fec0::1 packet loss: " + str(packetLoss1*100) + '%'
    f.write("fec0::1 transmitted: " + str(int(transmitted1)) + '
')
    f.write("fec0::1 received: " + str(int(received1)) + '
')
    f.write("fec0::1 packet loss: " + str(packetLoss1*100) + '%')
    f.write('
')
elif (addr[0] == 'fec0::8'):
    print "fec0::8 transmitted: " + str(int(transmitted8))
    print "fec0::8 received: " + str(int(received8))
    print "fec0::8 packet loss: " + str(packetLoss8*100) + '%'
    f.write("fec0::8 transmitted: " + str(int(transmitted8)) + '
')
    f.write("fec0::8 received: " + str(int(received8)) + '
')
    f.write("fec0::8 packet loss: " + str(packetLoss8*100) + '%')
    f.write('
')
print "total transmitted: " + str(int(transmitted1 + transmitted8))
print "total received: " + str(int(received1 + received8))
print "total packet loss: " + str(((transmitted1 + transmitted8) - received1 - received8) / (transmitted1 + transmitted8)) * 100) + '%'
print "total received: " + str(int(transmitted1 + transmitted8)) + '
')
print "total packet loss: " + str(((transmitted1 + transmitted8) - received1 - received8) / (transmitted1 + transmitted8)) * 100) + '%')
print "time elapsed: " + str(delta)
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

```java
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.server.handler.DefaultHandler;

/**
 * Application Lifecycle Listener implementation for start/stop Embedding Jetty Server configured to manage WebSocket.
 */
public class WsnServerServletContextListener implements ServletContextListener {
    private Server server = null;

    /**
     * Start Embedding Jetty server when WEB Application is started.
     */
    public void contextInitialized(ServletContextEvent event) {
        try {
            // 1) Create a Jetty server with the 8081 port.
            this.server = new Server(8081);
            // 2) Register WsnWebSocketHandler in the Jetty server instance.
            WsnWebSocketHandler wsnWebSocketHandler = new WsnWebSocketHandler();
            wsnWebSocketHandler.setHandler(new DefaultHandler());
            server.setHandler(wsnWebSocketHandler);
            // 2) Start the Jetty server.
            server.start();
        } catch (Throwable e) {
            e.printStackTrace();
        }
    }
```
**Stop Embedding Jetty server when WEB Application is stopped.**

```java
public void contextDestroyed(ServletContextEvent event) {
    if (server != null) {
        try {
            // stop the Jetty server.
            server.stop();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

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

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.Socket;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.text.DateFormat;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.CopyOnWriteArraySet;
import javax.servlet.http.HttpServletRequest;
import org.eclipse.jetty.websocket.WebSocket;
import org.eclipse.jetty.websocket.WebSocketHandler;

public class WsnWebSocketHandler extends WebSocketHandler {

    private final Set<WsnWebSocket> webSockets = new CopyOnWriteArraySet<WsnWebSocket> ();

    private void broadcast(String msg) {
        for (WsnWebSocket _webSocket : webSockets) {
            try {
                _webSocket.connection.sendMessage(msg);
            } catch (IOException e) {
                // TODO Auto-generated catch block
                System.err.println("Error when sending message to client's websocket!");
                e.printStackTrace();
            }
        }
    }
}
```
/************************************************************
* Thread that handles the TCP Socket connection, and
* send message to all clients
************************************************************/
Runnable tcpSocketComm = new Runnable() {
    @Override
    public void run() {
        try {
            Socket sock = new Socket("ib214m06.cs.unb.ca", 6106);
            BufferedReader in = new BufferedReader(new InputStreamReader(
                sock.getInputStream()));
            PrintWriter out = new PrintWriter(sock.getOutputStream(), true);
            String command = "routes";
            char[] buffer = new char[100];
            String msg = ""; // message received from socket
            if (in.read(buffer) == -1) { // read the welcome message
                System.err.println("IO Error!");
                System.exit(-1);
            }
            for (int i = 0; buffer[i] != '\0'; i++)
                buffer[i] = '\0';
            // a hash map that stores all the nodes' routes
            Map<String, String> path = new HashMap<String, String>();
            out.println(command);
            while ((msg = in.readLine()) != null) {
                msg = msg.trim();
                if (msg.startsWith("blip")) {
                    int arrowIndex = msg.indexOf('>');
                    msg = msg.substring(arrowIndex + 1).trim();
                }
            }
            /************************************************************
             * Now we have a message that contains only the routes
             * information in String msg, we can extract routes
             * information for every node now.
             ************************************************************/
            if (msg.equals("0x64:")) {
                out.println(command);
            } else {
                String[] splitMsg = msg.split(":");
            }
            /************************************************************
             * There is a message delay when a new node joined the
             * network, so we use a null detect here to prevent
             * array index out of bound exception.
             ************************************************************/
            String node = null;
            String route = null;
            node = splitMsg[0].trim();
            if (splitMsg.length > 1) // get rid of array index out
                // of bound exception
                route = splitMsg[1].trim();
            boolean change = false;
            if (path.containsKey(node) && route != null) {
                String oldPath = path.get(node).split("&")[0];
                if (!oldPath.equals(route)) {
                    change = true;
                }
            }
            // get the route change time
            DateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss")
            /************************************************************
             * Thread that handles the TCP Socket connection, and
             * send message to all clients
             ************************************************************/
"yyyy/MM/dd HH:mm:ss";
Date date = new Date();
String changeTime = dateFormat.format(date);

/**********************
* Write the route information to the file
* ************************/
FileWriter fr = new FileWriter(
  "route_info.log", true);
BufferedWriter br = new BufferedWriter(fr);
br.write("New route for node " + node + ": "+route + \"\t\" + changeTime + \"\n\n\);
br.close();
String routeAndTime = route.concat("&").concat(
  changeTime);
path.put(node, routeAndTime);

/**********************
* Write the route information to the file
* ************************/
FileWriter fr = new FileWriter(
  "route_info.log", true);
BufferedWriter br = new BufferedWriter(fr);
br.write("New route for node " + node + ": "+route + \"\t\" + changeTime + \"\n\n\n");
br.close();
change = true;
}
else {
  if (route != null) {
    DateFormat dateFormat = new SimpleDateFormat(
      "yyyy/MM/dd HH:mm:ss");
    Date date = new Date();
    String changeTime = dateFormat.format(date);
    String routeAndTime = route.concat("&").concat(
      changeTime);
    path.put(node, routeAndTime);
    /************************
    * Write the route information to the file
    * ************************/
    FileWriter fr = new FileWriter(
      "route_info.log", true);
    BufferedWriter br = new BufferedWriter(fr);
    br.write("New route for node " + node + ": "
      + route + \"\t\" + changeTime + \"\n\n\n");
    br.close();
    change = true;
  }
}
}
if (change) {
  System.out.println("Routes change at "+path.get(node).split("&")[1]); // get
  // change
  // time
  System.out.println("New route for " + node + ": " + route);
  // broadcast to all clients
  //broadcast("Routes change at "+
  // + path.get(node).split("&")[1]);
  broadcast("New route for " + node + ": " + route);
  Set<Map.Entry<String, String>> entries = path.entrySet();
  for (Map.Entry<String, String> entry : entries) {
    String key = entry.getKey();
    String value = entry.getValue();
    String nodePath = value.split("&")[0];
    broadcast(key + ": " + nodePath);
  }
  }
} catch (IOException e1) {
  System.err.println(e1.getMessage());
}
Runnable udpSocketComm = new Runnable() {
    @Override
    public void run() {
        byte[] receiveData = new byte[141]; // UDP packet payload size is 141 bytes
        int sender;
        int seqno;
        float light;
        float temperature;
        float humidity;
        float voltage;
        DecimalFormat df = new DecimalFormat("#.00");
        try {
            InetAddress IPAddress = InetAddress.getByName("fe00::64");
            DatagramSocket udp = new DatagramSocket(7000, IPAddress);
            while (true) {
                DatagramPacket packet = new DatagramPacket(receiveData, receiveData.length);
                udp.receive(packet);
                byte[] recvByte = packet.getData();
                for (int i = 0; i < recvByte.length; i++) {
                    int value = ((int) recvByte[i]) & 0xff; // convert byte to int
                    System.out.print(value + " , ");
                    System.out.print(\n);
                }
                int senderHigh = ((int) recvByte[2]) & 0xff;
                int senderLow = ((int) recvByte[3]) & 0xff;
                sender = senderHigh + senderLow;
                System.out.println("sender is " + sender);
                //broadcast("sender is " + sender);
                int seqHigh = ((int) recvByte[0]) & 0xff;
                int seqLow = ((int) recvByte[1]) & 0xff;
                seqno = seqHigh * 256 + seqLow;
                System.out.println("packet sequence number is " + seqno);
                //broadcast("packet sequence number is " + seqno);
                int lightHigh = ((int) recvByte[13]) & 0xff;
                int lightLow = ((int) recvByte[14]) & 0xff;
                float lightRead = lightHigh * 256 + lightLow;
                light = (float) (625 * 1000000 * (lightRead * 1.5 / 4096) / 100000);
                System.out.println("light is " + df.format(light) + "lux");
                broadcast("light" + sender + "," + df.format(light));
                int tempHigh = ((int) recvByte[17]) & 0xff;
                int tempLow = ((int) recvByte[18]) & 0xff;
                int tempRead = tempHigh * 256 + tempLow;
                temperature = (float) (tempRead * 0.01 - 40);
                System.out.println("temperature is " + df.format(temperature));
                broadcast("temperature" + sender + "," + df.format(temperature));
                int humHigh = ((int) recvByte[19]) & 0xff;
                int humLow = ((int) recvByte[20]) & 0xff;
                int humRead = humHigh * 256 + humLow;
                humidity = (float) (humRead * humRead * (-0.0000028 + 0.0405 * humRead - 4));
                System.out.println("humidity is " + df.format(humidity) + "]%);" + df.format(humidity));
                broadcast("humidity" + sender + "," + df.format(humidity));
            }
        }
    }
};
int volHigh = ((int) recvByte[21]) & 0xff;
int volLow = ((int) recvByte[22]) & 0xff;
float volRead = volHigh * 256 + volLow;
voltage = volRead * 3 / 4096;
System.out.println("voltage is "+ df.format(voltage) + "V");
broadcast("voltage" + sender + "," + df.format(voltage));
}
catch (UnknownHostException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
catch (SocketException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}
public WsnWebSocketHandler() {
super();
Thread tcpSocketThread = new Thread(tcpSocketComm);
tcpSocketThread.start();
Thread udpSocketThread = new Thread(udpSocketComm);
udpSocketThread.start();
public WebSocket doWebSocketConnect(HttpServletRequest request,
    String protocol) {
    return new WsnWebSocket();
}
private class WsnWebSocket implements WebSocket.OnTextMessage {
    private Connection connection;
    public void onOpen(Connection connection) {
        // Client (Browser) WebSockets has opened a connection.
        // 1) Store the opened connection
        this.connection = connection;
        // 2) Add WsnWebSocket in the global list of WsnWebSocket
        instances
        // instance.
        webSockets.add(this);
    }
    public void onMessage(String data) {
        // Loop for each instance of WsnWebSocket to send message server to
each client WebSockets.
        try {
            for (WsnWebSocket webSocket : webSockets) {
                // send a message to the current client WebSocket.
                webSocket.connection.sendMessage(data);
            }
        } catch (IOException x) {
            // Error was detected, close the WsnWebSocket client side
            this.connection.close();
        }
    }
    public void onClose(int closeCode, String message) {
        // Remove WsnWebSocket in the global list of WsnWebSocket
        // instance.
        webSockets.remove(this);
    }
}
D.2 Client Side Source Code

Util.js contains a HashTable class in JavaScript.

Listing D.3: Util.js

```javascript
/* Hash table class */
function HashTable(obj) {
  this.length = 0;
  this.items = {};
  for (var p in obj) {
    if (obj.hasOwnProperty(p)) {
      this.items[p] = obj[p];
      this.length++;
    }
  }
  this.put = function(key, value) {
    var previous = undefined;
    if (this.items[key]) {
      previous = this.items[key];
    }
    else {
      this.length++;
    }
    this.items[key] = value;
    return previous;
  }
  this.get = function(key) {
    return this.hasItem(key) ? this.items[key] : undefined;
  }
  this.hasItem = function(key) {
    return this.items.hasOwnProperty(key);
  }
  this.remove = function(key) {
    if (this.hasItem(key)) {
      previous = this.items[key];
      this.length--;
      delete this.items[key];
      return previous;
    }
    else {
      return undefined;
    }
  }
  this.keys = function() {
    var keys = [];
    for (var k in this.items) {
      if (this.hasItem(k)) {
```
keys.push(k);
}
return keys;
}
this.values = function()
{
    var values = [];
    for (var k in this.items) {
        if (this.hasOwnProperty(k)) {
            values.push(this.items[k]);
        }
    }
    return values;
};
this.each = function(fn) {
    for (var k in this.items) {
        if (this.hasOwnProperty(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
<div id="header">
</div>

<div id="Page">
<div id="textFieldWrapper">
  <font color="white" size="4">Dynamic update of new routes</font>
</div>
<div id="sensorReadingsWrapper">
  <font color="white" size="4">Sensor readings</font>
  <div id="node1">
    <label id="node1label">node 0x1</label>
    <div id="node1temperature">
      <font color="white">temperature</font>
      <div id="temperatureValue1"></div>
    </div>
    <div id="node1humidity">
      <font color="white">humidity</font>
      <div id="humidityValue1"></div>
    </div>
    <div id="node1light">
      <font color="white">light</font>
      <div id="lightValue1"></div>
    </div>
    <div id="node1voltage">
      <font color="white">voltage</font>
      <div id="voltageValue1"></div>
    </div>
  </div>
  <div id="node8">
    <label id="node8label">node 0x8</label>
    <div id="node8temperature">
      <font color="white">temperature</font>
      <div id="temperatureValue8"></div>
    </div>
    <div id="node8humidity">
      <font color="white">humidity</font>
      <div id="humidityValue8"></div>
    </div>
    <div id="node8light">
      <font color="white">light</font>
      <div id="lightValue8"></div>
    </div>
    <div id="node8voltage">
      <font color="white">voltage</font>
      <div id="voltageValue8"></div>
    </div>
  </div>
</div>

<div id="parameterPanel">
  <h1>WSN parameters</h1>
  <p>power level: -25dBm</p>
  <p>RTUP: 0.6 seconds</p>
  <p>Speed steps: 80</p>
</div>

<div id="canvasFieldWrapper">
  <font color="white" size="4">WSN testbed in ITB214 lab</font>
  <canvas id="nodeCanvas" width="400" height="500"></canvas>
  <button id="toggleButton" type="button" onclick="toggleRoutes()">show stationary nodes routes</button>
</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("lineCanvas");
104 var nc = nodeCanvas.getContext("2d");
105 //var lc = lineCanvas.getContext("2d");
106 if (!window.WebSocket)
107 alert("WebSocket not supported by this browser");
108 function $( ) {
109 return document.getElementById(arguments[0]);
110 }
111 function $F () {
112 return document.getElementById(arguments[0]).value;
113 }
114 function keyCode(ev) {
115 if (window.event)
116 return window.event.keyCode;
117 return ev.keyCode;
118 }
119
120 /* Instantiate JSGL Panel */
121 myPanel = new jsgl.Panel(document.getElementById("canvasField"));
122
123 /* Start drawing! */
124 /* draw walls */
125 var centerWall = myPanel.createRectangle();
126 centerWall.setWidth(150);
127 centerWall.setHeight(20);
128 centerWall.setLocationXY(125, 270);
129 myPanel.addElement(centerWall);
130 centerWall.setFill().setColor("rgb(100,100,100)");
131
132 var leftWall = myPanel.createRectangle();
133 leftWall.setWidth(55);
134 leftWall.setHeight(20);
135 leftWall.setLocationXY(0, 270);
136 myPanel.addElement(leftWall);
137 leftWall.setFill().setColor("rgb(100,100,100)");
138
139 var rightWall = myPanel.createRectangle();
140 rightWall.setWidth(60);
141 rightWall.setHeight(20);
142 rightWall.setLocationXY(348, 270);
143 myPanel.addElement(rightWall);
144 rightWall.setFill().setColor("rgb(100,100,100)");
145
146 /* draw lines */
147 var polyline0 = myPanel.createPolyline();
148 var polyline1 = myPanel.createPolyline();
149 with (polyline1.getStroke()) {
150 setColor("rgb(0,0,255)"); // red color
151 setWeight(3); // 3px width
152 setDashStyle(jsgl.DashStyles.DASH); // dashed
153 setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
154 setOpacity(0.7); // 30% transparent
155 }
156 var polyline2 = myPanel.createPolyline();
157 with (polyline2.getStroke()) {
158 setColor("rgb(255,0,0)"); // red color
```
```javascript
setWeight(2); // 2px width
setDisplayStyle(jsgl.DashStyles.DASH); // dashed
setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
setOpacity(0.5); // 50% transparent
}
var polyline3 = myPanel.createPolyline();
with (polyline3.stroke) {
  setColor('rgb(255,0,0)'); // red color
  setWeight(2); // 2px width
  setDisplayStyle(jsgl.DashStyles.DASH); // dashed
  setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
  setOpacity(0.5); // 50% transparent
}
var polyline4 = myPanel.createPolyline();
with (polyline4.stroke) {
  setColor('rgb(255,0,0)'); // red color
  setWeight(2); // 2px width
  setDisplayStyle(jsgl.DashStyles.DASH); // dashed
  setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
  setOpacity(0.5); // 50% transparent
}
var polyline5 = myPanel.createPolyline();
with (polyline5.stroke) {
  setColor('rgb(255,0,0)'); // red color
  setWeight(2); // 2px width
  setDisplayStyle(jsgl.DashStyles.DASH); // dashed
  setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
  setOpacity(0.5); // 50% transparent
}
var polyline6 = myPanel.createPolyline();
with (polyline6.stroke) {
  setColor('rgb(255,0,0)'); // red color
  setWeight(2); // 2px width
  setDisplayStyle(jsgl.DashStyles.DASH); // dashed
  setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
  setOpacity(0.5); // 50% transparent
}
var polyline7 = myPanel.createPolyline();
with (polyline7.stroke) {
  setColor('rgb(255,0,0)'); // red color
  setWeight(2); // 2px width
  setDisplayStyle(jsgl.DashStyles.DASH); // dashed
  setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
  setOpacity(0.5); // 50% transparent
}
var polyline8 = myPanel.createPolyline();
with (polyline8.stroke) {
  setColor('rgb(255,0,0)'); // red color
  setWeight(2); // 2px width
  setDisplayStyle(jsgl.DashStyles.DASH); // dashed
  setEndcapType(jsgl.EndcapTypes.SQUARE); // squares on ends
  setOpacity(0.5); // 50% transparent
}
var polylines = new Array();
polylines.push(polyline0); // fake polyline
polylines.push(polyline1);
polylines.push(polyline2);
polylines.push(polyline3);
polylines.push(polyline4);
polylines.push(polyline5);
polylines.push(polyline6);
polylines.push(polyline7);
polylines.push(polyline8);
myPanel.addElement(polyline1);
/*
myPanel.addElement(polyline2);
```
myPanel.addElement(polyline3);
myPanel.addElement(polyline4);
myPanel.addElement(polyline5);
myPanel.addElement(polyline6);
myPanel.addElement(polyline7);
myPanel.addElement(polyline8);

/* Draw stationary nodes */
node2 = myPanel.createCircle();
node2.setCenterLocationXY(20, 20);
node2.setRadius(5);
node2.setStroke().setWeight(5);
node2.setStroke().setColor("rgb(255,0,0)");
myPanel.addElement(node2);

node3 = myPanel.createCircle();
node3.setCenterLocationXY(380, 20);
node3.setRadius(5);
node3.setStroke().setWeight(5);
node3.setStroke().setColor("rgb(255,0,0)");
myPanel.addElement(node3);

node4 = myPanel.createCircle();
node4.setCenterLocationXY(20, 250);
node4.setRadius(5);
node4.setStroke().setWeight(5);
node4.setStroke().setColor("rgb(255,0,0)");
myPanel.addElement(node4);

node5 = myPanel.createCircle();
node5.setCenterLocationXY(380, 250);
node5.setRadius(5);
node5.setStroke().setWeight(5);
node5.setStroke().setColor("rgb(255,0,0)");
myPanel.addElement(node5);

node6 = myPanel.createCircle();
node6.setCenterLocationXY(20, 480);
node6.setRadius(5);
node6.setStroke().setWeight(5);
node6.setStroke().setColor("rgb(255,0,0)");
myPanel.addElement(node6);

node7 = myPanel.createCircle();
node7.setCenterLocationXY(380, 480);
node7.setRadius(5);
node7.setStroke().setWeight(5);
node7.setStroke().setColor("rgb(255,0,0)");
myPanel.addElement(node7);

/* Draw moving nodes */
node1 = myPanel.createCircle();
node1.setRadius(5);
node1.setStroke().setWeight(5);
node1.setStroke().setColor("rgb(0,0,255)");
myPanel.addElement(node1);

node8 = myPanel.createCircle();
node8.setRadius(5);
node8.setStroke().setWeight(5);
node8.setStroke().setColor("rgb(0,255,0)");
myPanel.addElement(node8);

/* Draw edge router Ox64 */
node0 = myPanel.createCircle();
node0.setCenterLocationXY(50, 250);
300 node0.setRadius(5);
301 node0.setStroke().setWeight(5);
302 node0.setStroke().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("Ox1");
327 label2.setText("Ox2");
328 label3.setText("Ox3");
329 label4.setText("Ox4");
330 label5.setText("Ox5");
331 label6.setText("Ox6");
332 label7.setText("Ox7");
333 label8.setText("Ox8");
334 label0.setText("edge router: Ox64");
335
336 label0.setLocationXY(node0.getCenterX() + 20, node0.getCenterY() - 5);
337 myPanel.addElement(label0);
338
339 /** mouse events */
340 var nodelabel = function() {
341 label1.setLocationXY(nodel.getCenterX(), nodel.getCenterY() + 15);
342 myPanel.addElement(label1);
343 var nodelabelout = function() {
344 myPanel.removeElement(label1);
345 }
346 nodel.addMouseListener(nodelabel);
347 nodel.addMouseOutListener(nodelabelout);
348
349 label2.setLocationXY(node2.getCenterX(), node2.getCenterY() + 10);
350 myPanel.addElement(label2);
351
352 label3.setLocationXY(node3.getCenterX(), node3.getCenterY() + 10);
353 myPanel.addElement(label3);
354
355 label4.setLocationXY(node4.getCenterX(), node4.getCenterY() - 20);
356 myPanel.addElement(label4);
357
358 label5.setLocationXY(node5.getCenterX(), node5.getCenterY() - 20);
359 myPanel.addElement(label5);
360
361 label6.setLocationXY(node6.getCenterX(), node6.getCenterY() + 10);
362 myPanel.addElement(label6);
363
364 label7.setLocationXY(node7.getCenterX(), node7.getCenterY() + 10);
365 myPanel.addElement(label7);
366
367 102
var node8label = function() {
    label8.setLocationXY(node8.getX(), node8.getY() + 15);
    myPanel.addElement(label8);
}

var node8labelout = function() {
    myPanel.removeElement(label8);
}

node8.addMouseOverListener(node8label);
node8.addMouseOutListener(node8labelout);

/* Animator */
var animator11 = new jsgl.util.Animator(); /* node1 animator */
var animator12 = new jsgl.util.Animator();
var animator13 = new jsgl.util.Animator();
var animator14 = new jsgl.util.Animator();
var animator15 = new jsgl.util.Animator();
var animator16 = new jsgl.util.Animator();
var animator17 = new jsgl.util.Animator();
var animator18 = new jsgl.util.Animator();

/* Animator */
var animator81 = new jsgl.util.Animator(); /* node8 animator */
var animator82 = new jsgl.util.Animator();
var animator83 = new jsgl.util.Animator();
var animator84 = new jsgl.util.Animator();
var animator85 = new jsgl.util.Animator();
var animator86 = new jsgl.util.Animator();
var animator87 = new jsgl.util.Animator();
var animator88 = new jsgl.util.Animator();

var shortDuration1 = 4000;
var speed1 = 120 / 4000;
var longDuration1 = 250 / speed1;
var curveDuration1 = 0.25 * 2 * Math.PI * 50 / speed1;

var speed8 = speed1 / 1.25;
var shortDuration8 = 170 / speed8;
var longDuration8 = 300 / speed8;
var curveDuration8 = 0.25 * 2 * Math.PI * 50 / speed8;

/* node1 moving */
animator11.setStartValue(1 * Math.PI); // left top curve
animator11.setEndValue(1.5 * Math.PI);  
animator11.setDuration(curveDuration1); 
animator11.setRepeating(true);
animator12.setStartValue(0); // top short side 
animator12.setEndValue(120); 
animator12.setDuration(shortDuration1); 
animator12.setRepeating(true);
animator13.setStartValue(1.5 * Math.PI); // right top curve 
animator13.setEndValue(2 * Math.PI);  
animator13.setDuration(curveDuration1); 
animator13.setRepeating(true);
animator14.setStartValue(0); // right long side 
//animator14.setEndValue(220); 
animator14.setEndValue(250); 
animator14.setDuration(longDuration1); 
animator14.setRepeating(true);
animator15.setStartValue(0 * Math.PI); // right bottom curve 
animator15.setEndValue(0.5 * Math.PI);  
animator15.setDuration(curveDuration1); 
animator15.setRepeating(true);
setStartValue(0); // bottom short side
setEndValue(120);
setDuration(shortDuration1);
setRepeating(true);

setStartValue(0.5 * Math.PI); // left bottom curve
setEndValue(1 * Math.PI);
setDuration(curveDuration1);
setRepeating(true);

setStartValue(0); // left long side
setEndValue(220);
setDuration(longDuration1);
setRepeating(true);

animator81.setStartValue(0.5 * Math.PI); // right bottom curve
animator81.setEndValue(0 * Math.PI);
animator81.setDuration(curveDuration8);
animator81.setRepeating(true);

animator82.setStartValue(0); // right long side
animator82.setEndValue(270);
animator82.setDuration(longDuration8);
animator82.setRepeating(true);

animator83.setStartValue(2 * Math.PI); // right top curve
animator83.setEndValue(1.5 * Math.PI);
animator83.setDuration(curveDuration8);
animator83.setRepeating(true);

animator84.setStartValue(0); // top short side
animator84.setEndValue(170);
animator84.setDuration(shortDuration8);
animator84.setRepeating(true);

animator85.setStartValue(1.5 * Math.PI); // left top curve
animator85.setEndValue(1 * Math.PI);
animator85.setDuration(curveDuration8);
animator85.setRepeating(true);

animator86.setStartValue(0); // left long side
animator86.setEndValue(270);
animator86.setDuration(longDuration8);
animator86.setRepeating(true);

animator87.setStartValue(1 * Math.PI); // left bottom curve
animator87.setEndValue(0.5 * Math.PI);
animator87.setDuration(curveDuration8);
animator87.setRepeating(true);

animator88.setStartValue(0); // bottom short side
animator88.setEndValue(170);
animator88.setDuration(shortDuration8);
animator88.setRepeating(true);

var lineAnimator = new jsgl.util Animator(); // line
lineAnimator.setRepeating(true);

/* animation play */
/* node1 */
animator11.addEndListener(function () {
animator11.pause();
animator12.play();

ani\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\nator12.addEventListener(function() {
    animator12.pause();
    animator13.play();
});

animator13.addEventListener(function() {
    animator13.pause();
    animator14.play();
});

animator14.addEventListener(function() {
    animator14.pause();
    animator15.play();
});

animator15.addEventListener(function() {
    animator15.pause();
    animator16.play();
});

animator16.addEventListener(function() {
    animator16.pause();
    animator17.play();
});

animator17.addEventListener(function() {
    animator17.pause();
    animator18.play();
});

animator18.addEventListener(function() {
    animator18.pause();
    animator11.play();
});

/* node8 */
animator81.addEventListener(function() {
    animator81.pause();
    animator82.play();
});

animator82.addEventListener(function() {
    animator82.pause();
    animator83.play();
});

animator83.addEventListener(function() {
    animator83.pause();
    animator84.play();
});

animator84.addEventListener(function() {
    animator84.pause();
    animator85.play();
});

animator85.addEventListener(function() {
    animator85.pause();
    animator86.play();
});

animator86.addEventListener(function() {
    animator86.pause();
    animator87.play();
});
```javascript
animator87.addEndListener(function () {
    animator87.pause();
    animator88.play();
});

animator88.addEndListener(function () {
    animator88.pause();
    animator81.play();
});

var polylinesNodes = new Array(8); // store nodes of each polyline
for (var i = 0; i < 8; i++) { // 0 to 7 are polylines 1 to 8
    polylinesNodes[i] = new Array(9);
    for (var j = 0; j < 9; j++)
        polylinesNodes[i][j] = -1;
}

lineAnimator.addStepListener(function(t) { // line-moving listener
    for (var i = 1; i < polylines.length; i++)
        for (var j = 0; j < polylines[i].getPointsCount(); j++)
            polylines[i].setPointXYAt(nodes[polylinesNodes[i - 1][j]].getCenterX(), nodes[polylinesNodes[i - 1][j]].getCenterY(), j);
});

/* node1 */
animator11.addStepListener(function(t) { // left top curve
    node1.setCenterX(140 + 50 * Math.cos(t));
    node1.setCenterY(125 + 50 * Math.sin(t));
});

animator12.addStepListener(function(t) { // top short side
    node1.setCenterX(140 + t);
    node1.setCenterY(75);
});

animator13.addStepListener(function(t) { // right top curve
    node1.setCenterX(260 + 50 * Math.cos(t));
    node1.setCenterY(125 + 50 * Math.sin(t));
});

animator14.addStepListener(function(t) { // right long side
    node1.setCenterX(310);
    node1.setCenterY(125 + t);
});

animator15.addStepListener(function(t) { // right bottom curve
    node1.setCenterX(260 + 50 * Math.cos(t));
    node1.setCenterY(375 + 50 * Math.sin(t));
});

animator16.addStepListener(function(t) { // bottom short side
    node1.setCenterX(260 - t);
    node1.setCenterY(425);
});

animator17.addStepListener(function(t) { // left bottom curve
    node1.setCenterX(140 + 50 * Math.cos(t));
    node1.setCenterY(375 + 50 * Math.sin(t));
});

animator18.addStepListener(function(t) { // left long side
    node1.setCenterX(90);
    node1.setCenterY(375 - t);
});
```

106
/* node8 */

animator81.addStepListener(function(t) { // right bottom curve
  node8.setCenterX(285 + 50 * Math.cos(t));
  node8.setCenterY(400 + 50 * Math.sin(t));
});

animator82.addStepListener(function(t) { // right long side
  node8.setCenterX(335);
  node8.setCenterY(400 - t);
});

animator83.addStepListener(function(t) { // right top curve
  node8.setCenterX(285 + 50 * Math.cos(t));
  node8.setCenterY(100 + 50 * Math.sin(t));
});

animator84.addStepListener(function(t) { // top short side
  node8.setCenterX(285 - t);
  node8.setCenterY(50);
});

animator85.addStepListener(function(t) { // left top curve
  node8.setCenterX(115 + 50 * Math.cos(t));
  node8.setCenterY(100 + 50 * Math.sin(t));
});

animator86.addStepListener(function(t) { // left long side
  node8.setCenterX(65);
  node8.setCenterY(100 + t);
});

animator87.addStepListener(function(t) { // left bottom curve
  node8.setCenterX(115 + 50 * Math.cos(t));
  node8.setCenterY(400 + 50 * Math.sin(t));
});

animator88.addStepListener(function(t) { // bottom short side
  node8.setCenterX(115 + t);
  node8.setCenterY(450);
});

animator11.init();
animator81.init();
animator81.play();
animator81.play();
lineAnimator.init();
lineAnimator.play();

/* toggle button function */

var show = false;

function toggleRoutes() {
  if (show == false) {
    show = true;
    $('toggleBtn').innerHTML = "hide stationary nodes routes";
    myPanel.addElement(polyline2);
    myPanel.addElement(polyline3);
    myPanel.addElement(polyline4);
    myPanel.addElement(polyline5);
    myPanel.addElement(polyline6);
    myPanel.addElement(polyline7);
  } else {
    show = false;
    $('toggleBtn').innerHTML = "show stationary nodes routes";
    myPanel.removeElement(polyline2);
    myPanel.removeElement(polyline3);
    myPanel.removeElement(polyline4);
    myPanel.removeElement(polyline5);
    myPanel.removeElement(polyline6);
  }
}
myPanel.removeElement(polyline7);
}

var room = {
    join : function() {
        //this._username = name;
        //var location = document.location.toString().replace('http://',
        //    'ws://').replace('https://', 'wss://');
        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;
    },
    _onopen : function() {
        room._send('connected');
    },
    _onmessage : function(m) {
        if (m.data) {
            var text = m.data.toString();
            /* new routes information */
            if (text.search("New") !== -1) {
                var spanText = document.createElement('span');
                spanText.className = 'text';
                spanText.innerHTML = text;
                var lineBreak = document.createElement('br');
                textField.appendChild(spanText);
                textField.appendChild(lineBreak);
                textField.scrollTop = textField.scrollHeight - textField.clientHeight;
                var route = text;
                var routeArray = new Array();
                var count = 0;
                var xindex = route.indexOf("0x");
                var endindex;
                while (xindex !== -1) {
                    var numberStr;
                    if (count == 0) {
                        endindex = route.indexOf(";");
                        numberStr = route.substring(xindex + 2, endindex);
                        number = parseInt(numberStr);
                        routeArray[count] = number;
                        route = route.substring(endindex + 2);
                    } else {
                        endindex = route.indexOf(" ");
                        if (endindex == -1) {
                            routeArray[count] = 64;
                            break;
                        } else {
                            numberStr = route.substring(xindex + 2, endindex);
                            number = parseInt(numberStr);
                            routeArray[count] = number;
                            route = route.substring(endindex + 1);
                        }
                    }
                    count = count + 1;
                    xindex = route.indexOf("0x");
                }
                if (routeArray.length > 0) {
                    var currentRoute = ";
                    }
for (var i = 1; i < routeArray.length; i++) {
    currentRoute = currentRoute + routeArray[i] + ",";
} 
Htable.put(routeArray[0], currentRoute);
polyline1.clearPoints();
polyline2.clearPoints();
polyline3.clearPoints();
polyline4.clearPoints();
polyline5.clearPoints();
polyline6.clearPoints();
polyline7.clearPoints();
polyline8.clearPoints();
for (var i = 1; i < routeArray.length; i++) {
    // clear the stored nodes of each polyline
    for (var j = 0; j < 9; j++)
        polylinesNodes[i][j] = -1;
}
var keys = Htable.keys();
for (var i = 0; i < Htable.length; i++) {
    var routeArray = Htable.get(keys[i]).split(",");
    var nodeAndRoute = new Array();
    nodeAndRoute[0] = parseInt(keys[i]);
    for (var j = 0; j < routeArray.length - 1; j++)
        if (parseInt(routeArray[j]) === 64)
            nodeAndRoute[j + 1] = 0;
        else
            nodeAndRoute[j + 1] = parseInt(routeArray[j]);
    for (var k = 0; k < nodeAndRoute.length; k++)
        polylines[parseInt(keys[i][])]
            .addPointXY(nodes[nodeAndRoute[k]][].getCenterX(),
            nodes[nodeAndRoute[k]].getCenterY());
        polylinesNodes[parseInt(keys[i][]) - 1][k] = nodeAndRoute[k];
}
/* sensor readings */
else {
    var commaIndex = text.indexOf(",");
    var typeStr = text.substring(0, commaIndex);
    var valueStr = text.substring(commaIndex + 1);
    if (typeStr === "temperature") { /* node1 temperature */
        var temp1Field = $('temperatureValue1');
        var spanText = document.createElement('span');
        spanText.className = 'text';
        spanText.style.color = "yellow";
        spanText.innerHTML = valueStr + "\degree" + "C";
        if (temp1Field.hasChildNodes())
            temp1Field.replaceChild(spanText, 
            temp1Field.childNodes[0]);
        else
            temp1Field.appendChild(spanText);
    } else if (typeStr === "humidity") { /* node1 humidity */
        var humidity1Field = $('humidityValue1');
        var spanText = document.createElement('span');
        spanText.className = 'text';
        spanText.style.color = "yellow";
        spanText.innerHTML = valueStr + ";";
        if (humidity1Field.hasChildNodes())
            humidity1Field.replaceChild(spanText,
humidity1Field.childNodes[0]);

} else
humidity1Field.appendChild(spanText);

} else if (typeStr === "voltage1") {  /* node1 voltage*/
var voltage1Field = $( 'voltageValue1' );
var spanText = document.createElement('span');
spanText.className = 'text';
spanText.style.color = "yellow";
spanText.innerHTML = valueStr + '"V";
if (voltage1Field.hasChildNodes())
voltage1Field.replaceChild(spanText,
voltage1Field.childNodes[0]);
else
voltage1Field.appendChild(spanText);

} else if (typeStr === "light1") {  /* node1 light*/
var light1Field = $( 'lightValue1' );
var spanText = document.createElement('span');
spanText.className = 'text';
spanText.style.color = "yellow";
spanText.innerHTML = valueStr + '"lux";
if (light1Field.hasChildNodes())
light1Field.replaceChild(spanText,
light1Field.childNodes[0]);
else
light1Field.appendChild(spanText);

} else if (typeStr === "temperature8") {  /* node8 temperature*/
var temp8Field = $( 'temperatureValue8' );
var spanText = document.createElement('span');
spanText.className = 'text';
spanText.style.color = "yellow";
spanText.innerHTML = valueStr + '"deg" +"°C";
if (temp8Field.hasChildNodes())
temp8Field.replaceChild(spanText,
temp8Field.childNodes[0]);
else
temp8Field.appendChild(spanText);

} else if (typeStr === "humidity8") {  /* node8 humidity*/
var humidity8Field = $( 'humidityValue8' );
var spanText = document.createElement('span');
spanText.className = 'text';
spanText.style.color = "yellow";
spanText.innerHTML = valueStr + '"%";
if (humidity8Field.hasChildNodes())
humidity8Field.replaceChild(spanText,
humidity8Field.childNodes[0]);
else
humidity8Field.appendChild(spanText);

} else if (typeStr === "light8") {  /* node8 light*/
var light8Field = $( 'lightValue8' );
var spanText = document.createElement('span');
spanText.className = 'text';
spanText.style.color = "yellow";
spanText.innerHTML = valueStr + '"lux";
if (light8Field.hasChildNodes())
light8Field.replaceChild(spanText,
light8Field.childNodes[0]);
else
light8Field.appendChild(spanText);

} else if (typeStr === "voltage8") {  /* node8 voltage*/
var voltage8Field = $( 'voltageValue8' );
var spanText = document.createElement('span');
spanText.className = 'text';
spanText.style.color = "yellow";
spanText.innerHTML = valueStr + '"V";
if (voltage8Field.hasChildNodes())
voltage8Field.replaceChild(spanText,
voltage8Field.childNodes[0]);
else
voltage8Field.appendChild(spanText);
voltage8Field.appendChild(spanText);

_onclose : function(m) {
    this._ws = null;
    $('textField').innerHTML = ''; 
},
_onerror : function(e) {
    alert(e);
},
_send : function(message) {
    if (this._ws) 
        this._ws.send(message);
}

/* method for getting route information*/
room.join();
</script>

</div>
</div>

</script>

</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
School of Information and Communication Engineering
Beijing University of Post and Telecommunications
Beijing, China

Technical Report:

Poster:

Publications:
Conference Presentations: