

---

# COMP 4203 Project Report

## An Implementation of Internet Networking Using XBee Modules

---

Tekin Özbek  
Benjamin Yan

**Disclaimer:** The work presented in this report is new, and prepared only for the course COMP 4203, Winter 2015, Carleton University.

April 8, 2015

### CONTENTS

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Background . . . . .	3
1.2	Problem Definition . . . . .	3
1.3	Project Results . . . . .	3
1.4	Report Outline . . . . .	4
<b>2</b>	<b>Detailed Background</b>	<b>4</b>
2.1	XBee Modules . . . . .	4
2.2	TUN Devices . . . . .	6
2.3	Data Encapsulation . . . . .	6
2.3.1	XBee Fragments . . . . .	6
2.3.2	Wireless Security . . . . .	8
2.3.3	Compression . . . . .	9

2.4	Summary . . . . .	9
<b>3</b>	<b>System and Implementation Details</b>	<b>10</b>
3.1	Description of the System . . . . .	10
3.1.1	Component Breakdown . . . . .	10
3.1.2	XBLink . . . . .	10
3.2	Configuration of XBee Modules . . . . .	13
3.3	Sample Usage . . . . .	14
3.3.1	Network Applications . . . . .	14
3.4	Simple Performance Analysis . . . . .	15
3.4.1	Range Performance . . . . .	15
3.4.2	Speed Performance . . . . .	16
<b>4</b>	<b>Summary</b>	<b>18</b>
4.1	Possible Future Directions . . . . .	19

# 1 INTRODUCTION

## 1.1 BACKGROUND

The Internet is a well-established global system of computer networks, with a standard set of protocols that govern the communications and interactions between the connected computers and devices. The various devices and systems connected to the Internet are linked by a variety of technologies, ranging from electronic and wireless to optic networks [1].

The Internet has become a crucial part of the everyday lives of most North Americans, and we rely on it for a variety of services and resources. Although there are many inexpensive off-the-shelf solutions to connect a personal computer or device to the Internet, we were interested in exploring the possibility of creating computer networks that communicate using the standard Internet protocol suite [2], and are linked by an alternative method.

## 1.2 PROBLEM DEFINITION

Given the above information, we seek to develop a framework for building computer networks using XBee modules, which are small low-power radio transceivers designed to connect devices over short ranges [3]. There are a number of different models in the XBee family, each with different specifications in power, range, and compatibility; all XBee modules can either transmit raw radio signals, or communicate with each other using a packet-based application programming interface (API).

Our specific goal is to use two XBee modules to create a wireless point-to-point connection between two computers. Connection between each pair of computer and XBee module is established over USB, facilitated by a small circuit board that interfaces the XBee modules with the USB connector [4]. To create an Internet connection between the computers, we use a TUN kernel device that simulates a network layer device and transmits network layer packets between the computer and XBee module [5]. In other words, network layer packets are encapsulated in the XBee module packets, and an internet connection is virtualized between the two computers.

## 1.3 PROJECT RESULTS

Over the course of this project, we have developed a lightweight C++ program that allows us to establish a permanent wireless connection between two computers, using the XBee modules. The C++ program allows the two computers to address each other directly using IP addresses, and to communicate using standard network layer protocols to access / provide services for each other. This includes ping over the network, establishing SSH connections, transmitting files over SFTP, and serving webpages.

To achieve these results, we designed our own data encapsulation scheme that encapsulates IP packets in one or more XBee packets, due to the small size limit of the XBee packets. Reading and writing data to the XBee modules was done through an open-source library called libxbee [6]. We had considered using data compression to decrease the transmission size, but this proved to be time-consuming, and provided no benefits.

## 1.4 REPORT OUTLINE

The remainder of the report is organized as follows: In Section 2, we present the background information about the various devices, protocols, and tools used in this project. In Section 3, we describe the way that we have designed and implemented our system, the way that the XBee modules have been setup, and we discuss some examples of usage. In Section 4, we give a brief summary of our achievements and conclude the paper.

# 2 DETAILED BACKGROUND

In this section, we provide the reader with the detailed background knowledge required to understand:

- what an XBee module is.
- what a TUN device is.
- how we encapsulate IP packets from one computer to another, using the XBee packet API.

## 2.1 XBEE MODULES

The XBee family of wireless modules are small radio transceivers, developed and sold by Digi International [7]. They are based on the IEEE 802.15.4 standard, which is designed for low power and low speed devices, and operate at about 10m ranges with speeds of around 250kbit/s, although many devices do not reach this speed [8]. There are two types of network nodes under this standard: a full-function device (FFD), which, as the name suggests, has full functionality, and can act as “PAN coordinators” in the network; and a reduced-function device (RFD), which are simpler devices that cannot act as coordinators.

The standard allows the creation of two types of networks: a star network, where all communication between nodes is transmitted through a central Coordinator; or a peer-to-peer network, where nodes can communicate with each other directly [8]. In either network type, at least one full-function device must act as a coordinator; multiple coordinators can also be networked together. Examples of these networks are shown in Figure 2.2.

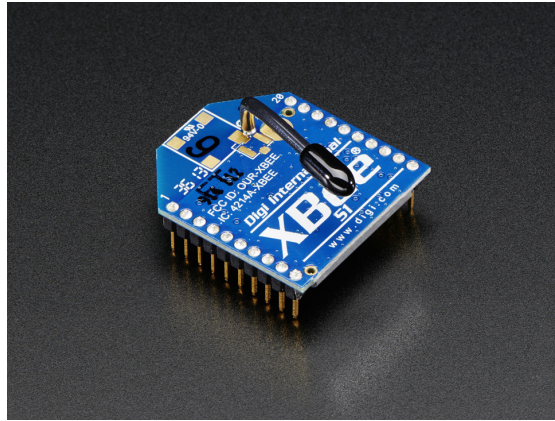


Figure 2.1: Example of an XBee module

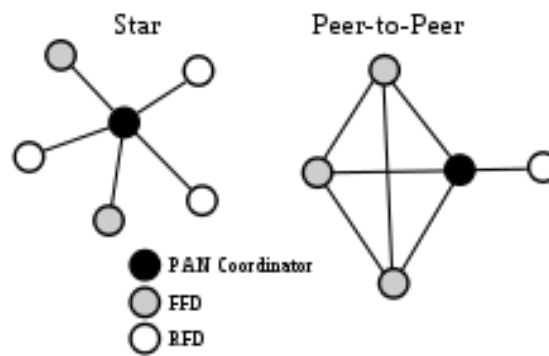


Figure 2.2: Star and peer-to-peer topologies allowed by IEEE 802.15.4 [8]

The XBee modules are full-function devices, so they can each act as a coordinator. Since we are working with only 2 devices, we arbitrarily choose one to be the coordinator, and set up a simple peer-to-peer network between the two devices. The manufacturer's specifications state that the RF line of sight range for our XBee model is about 100m. We verified that this is roughly accurate; this is shown in Section 3.4.

To interface the XBee modules with computers, we have also obtained an USB adapter for each module, shown in figure 2.3 [9]. Interfacing with the module through a USB connection serves two purposes: first, we can configure the devices through the USB connection, and set parameters such as coordinator address, and baud rate; this can be done using the X-CTU program from the XBee manufacturer (Figure 2.4). Second, we can send and receive data to the XBee modules over USB. Transmitting data over the usb device is particularly convenient in Linux systems, because we can easily read from and write to the USB device file [10]. We were required to install the FTDI driver on the computers, so that the USB devices can be recognized [4].

Simply being able to transmit and receive data using the XBee modules is not enough for our

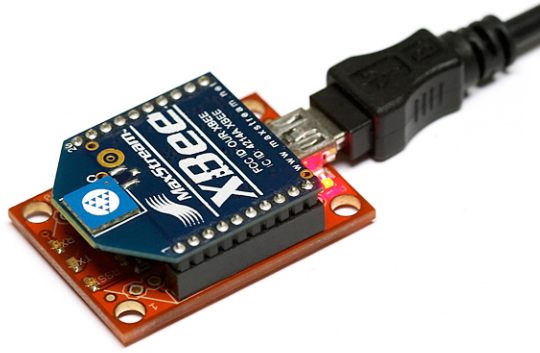


Figure 2.3: Example of a XBee USB adapter circuit

objectives, however. Next, we discuss how we can virtualize the network layer using the XBee wireless connection, with TUN devices.

## 2.2 TUN DEVICES

A TUN device is a virtual network kernel device that simulates the network layer, and is treated in the same way as a network adapter, such as an Ethernet device [5]. Once the TUN device is set up, a user-space program can be attached to the device. For our purposes, we can assign an IP address to the device, and enable it, using the Linux *ip* command. This means that any time that the kernel refers to this IP address, data transmission is done through the user-space program, to the TUN device, instead of the Ethernet or wireless device on the computer [12].

Similar to TUN devices are TAP devices, which stands for “network tap” [5]. The difference between a TUN device and a TAP device is in the layer: a TUN device works at the network layer, and transmits IP packets, while a TAP device works at the link layer, and transmits Ethernet frames [12]. Since we are interested in working directly with IP packets, we use TUN devices only, not TAP devices. We create a TUN device by opening a file under the path `/dev/net/tun`, and setting the file’s flags for a TUN device.

## 2.3 DATA ENCAPSULATION

### 2.3.1 XBEE FRAGMENTS

Although the IEEE 802.15.4 standard specifies a maximum packet size of 127 bytes, the XBee Series 2 Pro modules that we are using only allows a maximum payload of 72 bytes, due to significant number of bytes used in the packet header [13]. Since we are sending IP packets, which, for IPv4, have a size range between 20 and 65,535 bytes [14], it is very likely that one

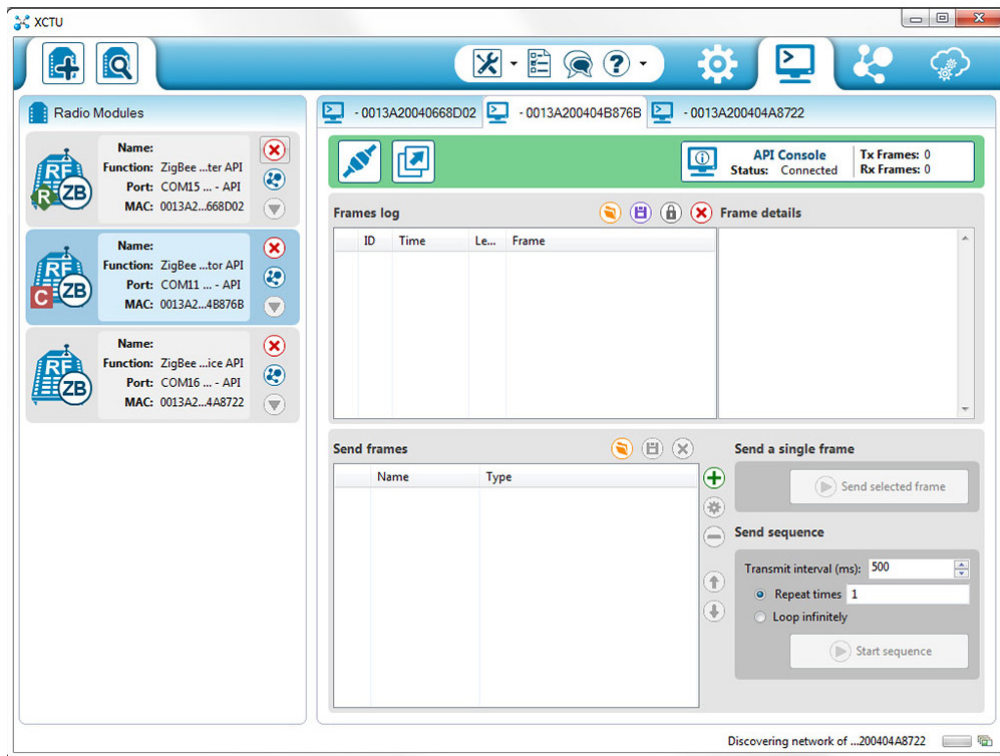


Figure 2.4: X-CTU program interface for configuring XBee modules and networks [11]

XBee fragment is not enough to contain the entire IP packet. Therefore, we need to be able to transmit a single IP packet over several XBee fragments.

To do so, we have developed an encapsulation scheme for the 72byte XBee fragment, which contains a header portion and a payload portion, shown in Figure 2.5. The first 2 bytes of the header is a randomly generated ID, unique for each IP packet transmitted. The next 4 bytes represent the length of the fragment payload. The following 2 bytes represent the fragment index number in the sequence for that IP packet. The remainder of the fragment contains the actual payload, which is a portion of the IP packet.

In the case that we transmit several IP packets, each packet over several XBee fragments, it is possible that the packets may arrive out of order. This is the reason that we employ a 2 byte ID number, and a 2 byte fragment number. Figure 2.6 shows a potential scenario where a number of fragments representing several different IP packets arrive out of order, and we must rely on the ID and fragment number to correctly reconstruct the original message on the receiving device.

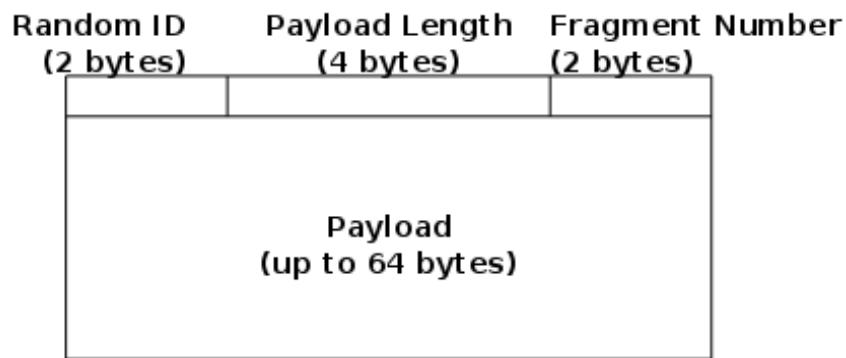


Figure 2.5: XBee fragment header and payload

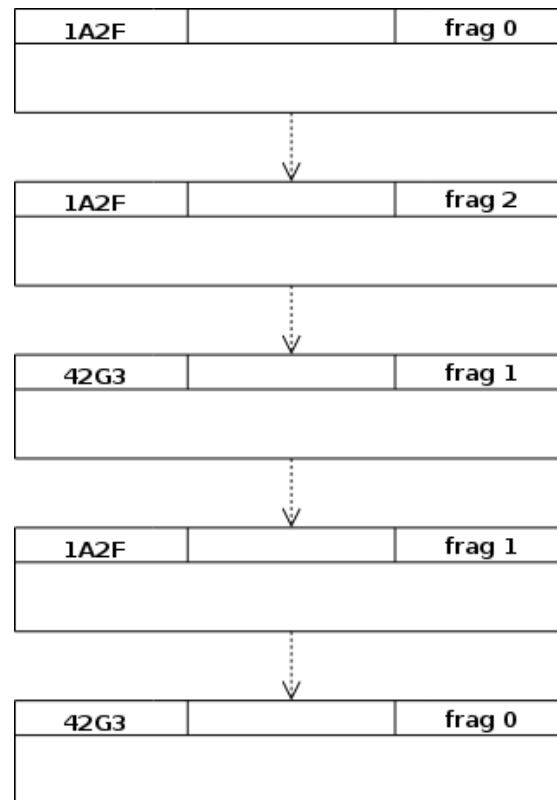


Figure 2.6: XBee fragments arriving out of order

### 2.3.2 WIRELESS SECURITY

It should be noted here that our implementation does not contain any security features. Currently, any security issue should be handled in the application layer. Programs like SSH and SFTP, or the HTTPS protocol already encrypt their contents before sending them over TCP.



Our design is highly susceptible to man-in-the-middle attacks. This type of attack could be carried out by an adversary who intercepts our radio transmissions, modifies the information, and sends the modified data. It is also possible for an adversary to simply forge XBee fragments, using valid headers and payload, and broadcast the fragments to our XBee modules.

To protect against this attack, we could use a secure encryption algorithm to encrypt the entire XBee fragment, and send the encrypted fragment. In this case, we already have a fragment number and a secret random ID inside the encrypted fragment, so the scheme would not be vulnerable to replay attacks. Alternatively, we could encrypt the IP packet, and send portions of the encrypted packet as payload in the XBee fragments. This method is less secure, because the header portion would be unencrypted, so an adversary could forge the header and perform a replay attack. In either case, we should also ensure the authenticity of each fragment by computing a message authentication code (MAC) for the fragment. This would consume additional bytes in each message.

Ultimately, we chose not to employ encryption in our design, because it would take up additional space in our already-limited fragment size. Since the goal of this project is only to explore the possibility of computer networking using XBee modules, the security aspect is not as crucial. Additionally, due to the range limitations on the XBee modules, and the novelty in their use as a networking interface, it seems unlikely that an adversary could perform an attack undetected, under our noses.

### 2.3.3 COMPRESSION

The speed of the network is of course slow, compared to wireless internet connections. We wanted to make the transmissions as fast as possible. One of our options was to compress the messages before sending them, and then send the compressed data in fragments. We implemented a prototype version using the LZ4 data compression library; we chose LZ4 because it is supposedly a fast compression library. The default MTU is 1500 bytes and our program spends a considerable amount of time compressing and decompressing each message without much gain in speeds. Also, due to this compression, additional memory allocation and deallocation also wastes time. In the end, using compression for such small messages turned out to be slower than if it were sent uncompressed.

## 2.4 SUMMARY

In this section, we have discussed, in detail, the background information for XBee modules, TUN devices, and our method for encapsulating IP packets. Next, we discuss how we implemented our design, and provide examples of the system in use.

## 3 SYSTEM AND IMPLEMENTATION DETAILS

In the previous section, we have provided the reader with the necessary background knowledge for each component of our project. Here, we present the details of the implementation and usage. In section 3.1, we discuss the details of the system design; in section 3.2, we discuss the detailed configuration of the XBee modules; and in section 3.3, we provide examples of our system in use, over the network connection created between two computers.

### 3.1 DESCRIPTION OF THE SYSTEM

#### 3.1.1 COMPONENT BREAKDOWN

As mentioned in the previous section, we create a network connection between two computers, using XBee modules at the link layer. At the application level on each computer, the network is virtualized so that applications would operate normally, and interact with the other node over a preset IP address. The translation from application layer to link layer is facilitated by a TUN device in the network layer. This relationship is visualized in Figure 3.1, in which data is sent from an application on Node 1 to an application on Node 2. Labels on each arrow show the data format passed between each subsystem.

In Figure 3.2, we show the network components and their subsystems, as well as the protocols used to communicate between the subsystems. Note that data at each layer of the network is virtualized such that each subsystem's behaviour and data processing does not need to be modified. In other words, applications on each node operate exactly as they would on a normal Internet connection.

#### 3.1.2 XBLINK

To enable quick setup of the XBee-based network between two computers, we have developed a program called XBLink to create a TUN device, and handle the communication from applications on the node to the XBee modules. The C++ program can be compiled using any C++11-compliant compiler (e.g., g++ 4.7+) and runs under standard Linux systems with access to `/dev/net/tun` and `/dev/urandom`. The library *libxbee* must be installed for XBLink to communicate with the devices.

Once the program is compiled, it is run as follows, with a number of command line arguments:

```
./xblink usb_dev xbee_model baud addr tun_dev
```

The argument `usb_dev` is the path to the XBee USB device; `xbee_model` refers to the specific type of XBee module used; `baud` refers to the baud rate preset on the XBee module; `addr` refers to the destination address of the XBee module on the *other* node; and `tun_dev` is a user-chosen name to assign to the TUN device to be created.

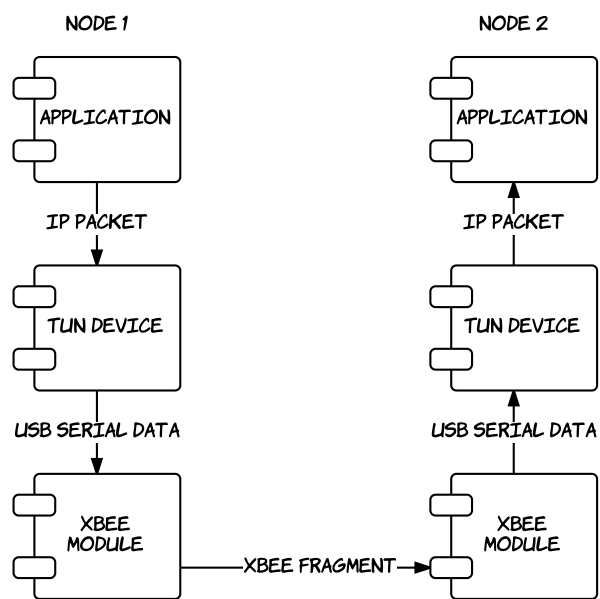


Figure 3.1: Data is sent from Node 1 to Node 2 through the various subsystems

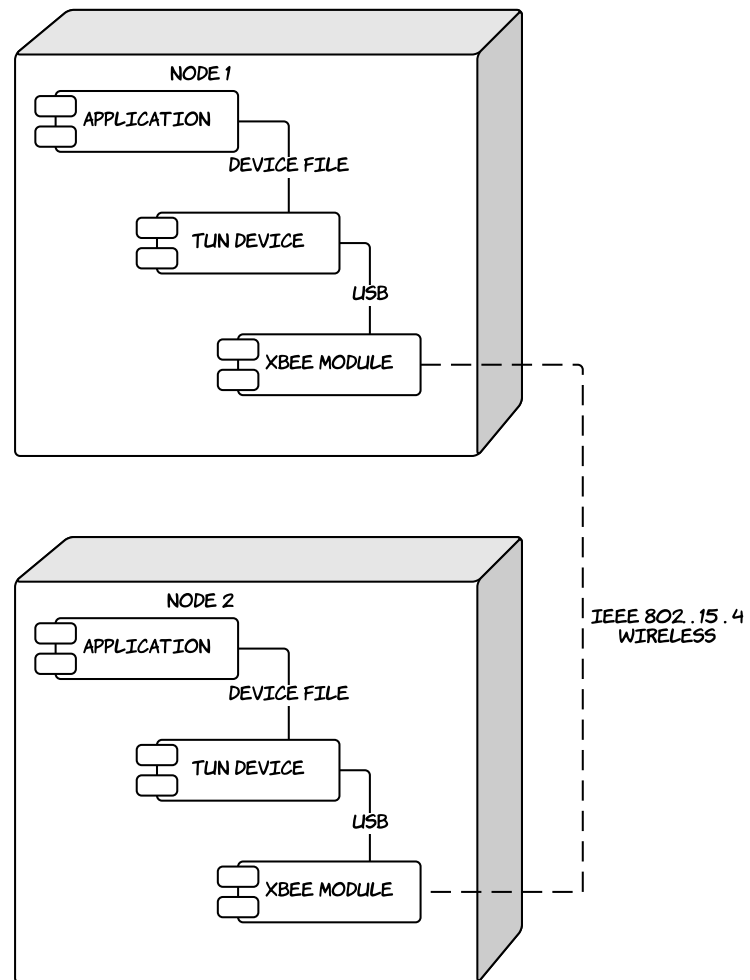


Figure 3.2: Component distribution in the XBee-based network

As a specific example, we create a TUN device named “tun77”, using the following command:

```
./xblink /dev/ttyUSB0 xbeeZB 115200 0x0013A200409E0550 tun77
```

Here, our XBee USB adapter is located at /dev/ttyUSB0; our XBee model is ZB; the baud rate has been set at 115200; and the address of the XBee module that we’re connecting to is at 0x0013A200409E0550. Note that the XBLink program runs for the duration of the connection, and outputs useful data, such as XBee fragment info.

Once the TUN device is created and running, we must assign an IP address to it. This is done by first enabling the device as a network interface:

```
sudo ip link set tun77 up
```

This Linux command simply activates the TUN device that we have created as a network interface. Next, we assign an IP address range to it:

```
sudo ip addr add 10.0.0.2/24 dev tun77
```

This command assigns the IP addresses 10.0.0.2/24 to the TUN device. Now, all internet traffic that is routed through this address range will be accessed through the TUN device. For example, if we ping the address, then UDP packets will be sent through the device to the other computer. This means that we have now successfully connected our nodes over the network.

In this section, we have described the system components in our implementation, and outlined the procedures of setting up the XBee-based network. In section 3.2, we describe the way that the XBee modules have been configured to work effectively in our system.

### 3.2 CONFIGURATION OF XBEE MODULES

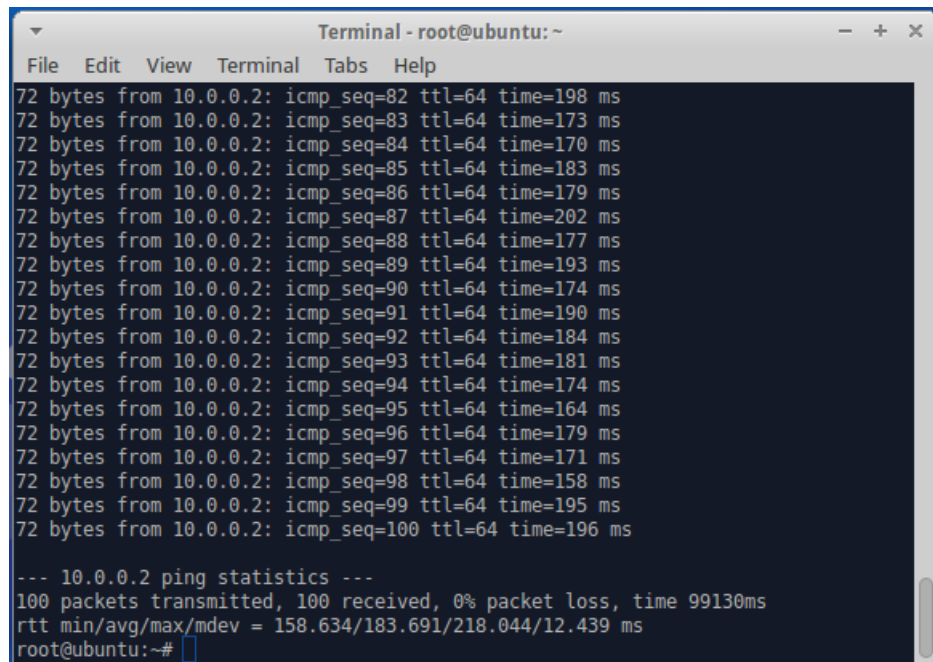
XBee modules must be configured properly in order for them to interact with the library, *libxbee*, which allows us to programmatically communicate with the devices. First and foremost, any module that interacts with the library must enable API mode. The modules must be set to hardware flow control by enabling the AP settings RTS (request-to-send) and CTS (clear-to-send). The recommended baud rate is any rate  $\leq 57600$ . Higher baud rates do work in the beginning, but eventually packets start to time out.

If you are using XBee Series 2 Pro (also known as ZigBee) modules, one of the modules have to be set as a PAN coordinator and the other ones should be router or end-point devices with sleep timers set to OFF (end-points may not have this option and they will fall asleep stop responding after a while). To change the device role (coordinator, router or end-point), a new firmware will be installed by X-CTU.

### 3.3 SAMPLE USAGE

#### 3.3.1 NETWORK APPLICATIONS

In this section, we give some examples of usage of our system. First, we simply ping the other node, using Linux's built in ping command, shown in Figure 3.3. In this figure, we see the result of ping, with some statistics on packets sent, and transmission speed. In this example, each packet had a size of 72 bytes (plus header), no packets were lost, and 100 pings required a total time of about 1.5min.

A terminal window titled "Terminal - root@ubuntu: ~" showing the output of a ping command. The output lists 100 individual ping results, each showing 72 bytes from 10.0.0.2 with varying ICMP sequence numbers and round-trip times. Below the individual results, it shows a summary: "100 packets transmitted, 100 received, 0% packet loss, time 99130ms" and "rtt min/avg/max/mdev = 158.634/183.691/218.044/12.439 ms". The prompt "root@ubuntu:~#" is visible at the bottom.

```
72 bytes from 10.0.0.2: icmp_seq=82 ttl=64 time=198 ms
72 bytes from 10.0.0.2: icmp_seq=83 ttl=64 time=173 ms
72 bytes from 10.0.0.2: icmp_seq=84 ttl=64 time=170 ms
72 bytes from 10.0.0.2: icmp_seq=85 ttl=64 time=183 ms
72 bytes from 10.0.0.2: icmp_seq=86 ttl=64 time=179 ms
72 bytes from 10.0.0.2: icmp_seq=87 ttl=64 time=202 ms
72 bytes from 10.0.0.2: icmp_seq=88 ttl=64 time=177 ms
72 bytes from 10.0.0.2: icmp_seq=89 ttl=64 time=193 ms
72 bytes from 10.0.0.2: icmp_seq=90 ttl=64 time=174 ms
72 bytes from 10.0.0.2: icmp_seq=91 ttl=64 time=190 ms
72 bytes from 10.0.0.2: icmp_seq=92 ttl=64 time=184 ms
72 bytes from 10.0.0.2: icmp_seq=93 ttl=64 time=181 ms
72 bytes from 10.0.0.2: icmp_seq=94 ttl=64 time=174 ms
72 bytes from 10.0.0.2: icmp_seq=95 ttl=64 time=164 ms
72 bytes from 10.0.0.2: icmp_seq=96 ttl=64 time=179 ms
72 bytes from 10.0.0.2: icmp_seq=97 ttl=64 time=171 ms
72 bytes from 10.0.0.2: icmp_seq=98 ttl=64 time=158 ms
72 bytes from 10.0.0.2: icmp_seq=99 ttl=64 time=195 ms
72 bytes from 10.0.0.2: icmp_seq=100 ttl=64 time=196 ms

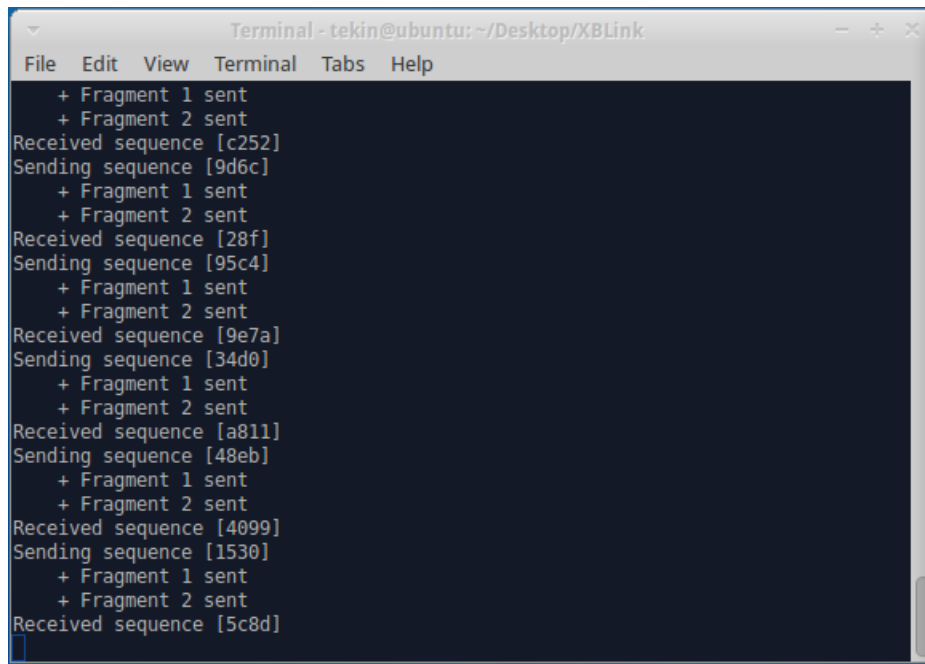
--- 10.0.0.2 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99130ms
rtt min/avg/max/mdev = 158.634/183.691/218.044/12.439 ms
root@ubuntu:~#
```

Figure 3.3: Running the program ping over the network

In Figure 3.4, we see the output of the XBLink program during execution of the ping program, which displays information about sent and received fragments and packets. In this example, packets only needed to be split over two fragments, as expected, since two fragments can contain a total payload size of 128 bytes.

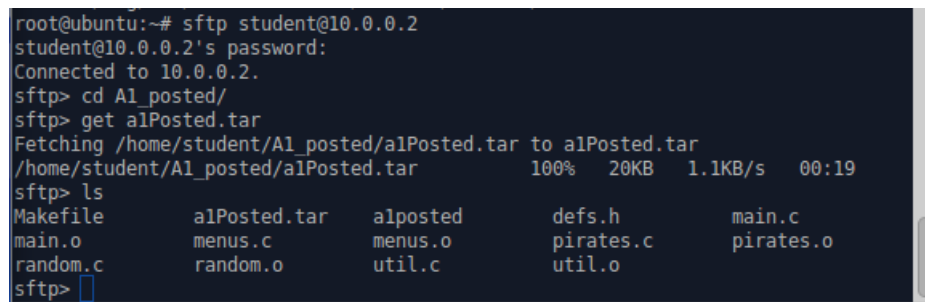
We are also able to perform secure file transfers between the two nodes. Figure 3.5 shows the successful transfer of a 19KB file over SFTP, at a data rate of about 1KB/s. We were able to run the program using default setup, without adjustments for the slower transmission speeds.

Finally, we show that we are able to create a secure connection from one node to the other, using SSH. Figure 3.6 shows that we are able to login to the other node under the "student" account, and view the home directory. Of course, this is not a surprise, since we were able to perform secure file transfers using SFTP, which uses SSH as the underlying protocol.

A terminal window titled "Terminal - tekin@ubuntu: ~/Desktop/XBLink" showing the output of the XBLink program. The output consists of a series of messages indicating the sending and receiving of data fragments and sequences. The messages are: "+ Fragment 1 sent", "+ Fragment 2 sent", "Received sequence [c252]", "Sending sequence [9d6c]", "+ Fragment 1 sent", "+ Fragment 2 sent", "Received sequence [28f]", "Sending sequence [95c4]", "+ Fragment 1 sent", "+ Fragment 2 sent", "Received sequence [9e7a]", "Sending sequence [34d0]", "+ Fragment 1 sent", "+ Fragment 2 sent", "Received sequence [a811]", "Sending sequence [48eb]", "+ Fragment 1 sent", "+ Fragment 2 sent", "Received sequence [4099]", "Sending sequence [1530]", "+ Fragment 1 sent", "+ Fragment 2 sent", and "Received sequence [5c8d]".

```
Terminal - tekin@ubuntu: ~/Desktop/XBLink
+ Fragment 1 sent
+ Fragment 2 sent
Received sequence [c252]
Sending sequence [9d6c]
+ Fragment 1 sent
+ Fragment 2 sent
Received sequence [28f]
Sending sequence [95c4]
+ Fragment 1 sent
+ Fragment 2 sent
Received sequence [9e7a]
Sending sequence [34d0]
+ Fragment 1 sent
+ Fragment 2 sent
Received sequence [a811]
Sending sequence [48eb]
+ Fragment 1 sent
+ Fragment 2 sent
Received sequence [4099]
Sending sequence [1530]
+ Fragment 1 sent
+ Fragment 2 sent
Received sequence [5c8d]
```

Figure 3.4: Console output of the XBLink program during a network connection

A terminal window showing an SFTP session. The user is root@ubuntu and connects to student@10.0.0.2. The password is entered. The user then navigates to the /A1\_posted/ directory and uses the 'get' command to download a file named 'a1Posted.tar'. The terminal shows the progress of the file transfer, indicating it is 100% complete with a size of 20KB and a transfer rate of 1.1KB/s. Finally, the user runs the 'ls' command to list the files in the current directory, showing a directory listing of various files and subdirectories.

```
root@ubuntu:~# sftp student@10.0.0.2
student@10.0.0.2's password:
Connected to 10.0.0.2.
sftp> cd A1_posted/
sftp> get a1Posted.tar
Fetching /home/student/A1_posted/a1Posted.tar to a1Posted.tar
/home/student/A1_posted/a1Posted.tar      100% 20KB 1.1KB/s 00:19
sftp> ls
Makefile      a1Posted.tar  alposted      defs.h        main.c
main.o        menus.c       menus.o       pirates.c     pirates.o
random.c      random.o     util.c        util.o
sftp>
```

Figure 3.5: Transferring a file over SFTP

### 3.4 SIMPLE PERFORMANCE ANALYSIS

#### 3.4.1 RANGE PERFORMANCE

First, we were interested in verifying that the range of the XBee modules is as advertised. To do this, we send 10 pings between the two modules at various distances, with no obstructions in the line of sight, and record the average roundtrip time. The ping packet sizes are the default value of 64 bytes. The data is shown in Figure 3.7. We can see that the performance is steady between 0 and 90m ranges, and the average roundtrip time remains at about 500ms. However, around 100m ranges or more, the roundtrip time increases significantly. This is due to increasing percentages of packet loss, which is caused by the weaker signal at these ranges.

```
root@ubuntu:~# ssh student@10.0.0.2
student@10.0.0.2's password:
Welcome to Ubuntu 12.04.5 LTS (GNU/Linux 3.2.0-68-generic-pae i686)

 * Documentation:  https://help.ubuntu.com/

New release '14.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Mar 29 12:49:08 2015 from 10.0.0.1

student @ COMP2401 : Sun Mar 29 01:25:08
~ 1 $ ls
4108Proj  A1_posted  A3      Documents  examples.desktop  Pictures  Templates
A1        A2         Desktop Downloads  Music             Public   Videos

student @ COMP2401 : Sun Mar 29 01:25:27
~ 2 $
```

Figure 3.6: Logging in to a remote node using SSH

So, we conclude that reliable performance can be maintained at close ranges below 90-100m, with this particular XBee model.

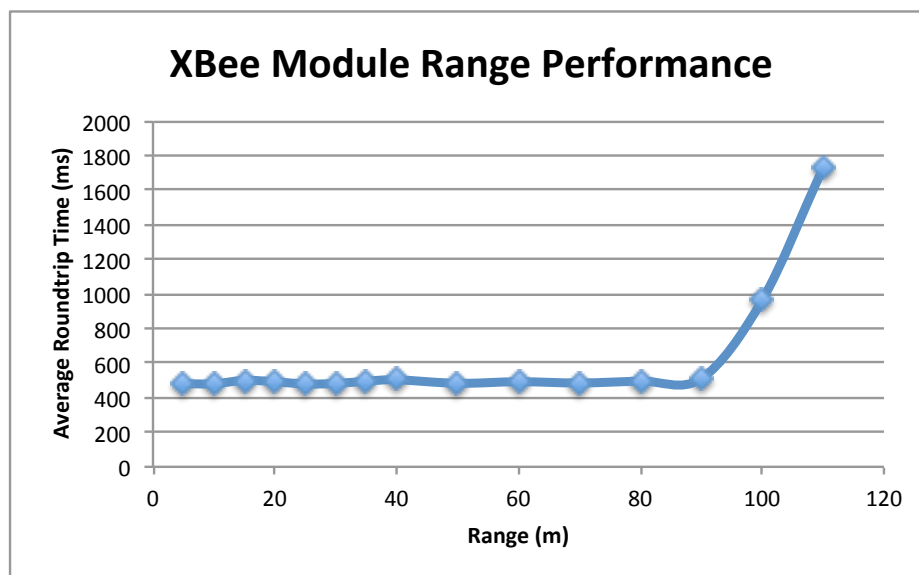


Figure 3.7: XBee module performance at various ranges

### 3.4.2 SPEED PERFORMANCE

To gain a rough idea of the speed of the XBee network, we perform a number of ping operations at different packet sizes, and analyze the data rates for each packet size. Figure 3.8



shows the various packet sizes used, in bytes, and the required average roundtrip time. The graph also shows the percentage of packets that were lost during transmission. We notice that the transfer rates are steady around 450B/s, when the packet sizes are small. However, at a packet size of 512B or greater, we see a significant drop in performance, as the transfer rate dips to 150B/s. The rate does appear to increase beyond this point, to approach the steady rate seen at low packet sizes.

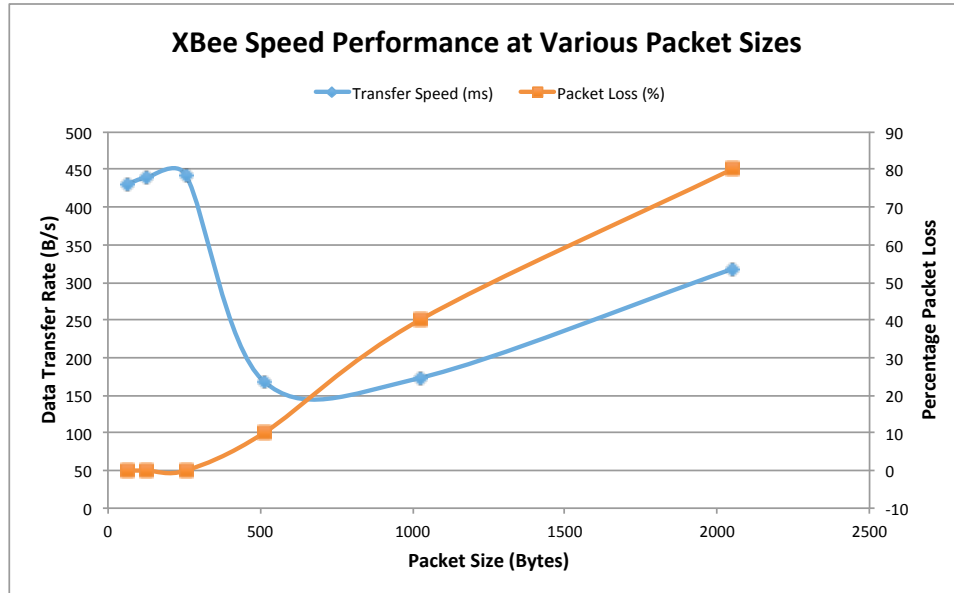


Figure 3.8: A comparison of data transfer performance at various packet sizes

From the figure, we immediately notice that higher number of packets were lost for larger packet sizes. It is possible that the ping program does not wait long enough for the XBee modules to send all the fragments of the packet, and erroneously considers the packets lost. We know that by default, a packet is sent every second by ping. It is also interesting that the transmission durations appear to increase over time, for larger packet sizes. This further supports the hypothesis above, since the modules have not completely transmitted all of the fragments in the first packet, before another packet is queued. If this is indeed the reason for the lost packets, then that means that the system should work for programs other than ping, where an immediate response from the other node is not required. In fact, we have seen that transferring files on the order of kilobytes works without loss.

To test this hypothesis, we compute the roundtrip time required to send a number of large packets, at an interval of 1 second each. We can approximate this using the following model: the first packet is sent at a rate of 500B/s, so, given a packet of size  $P$ , the roundtrip time for the first packet equals:

$$RTT_1 = \frac{2P}{500B/s} \quad (3.1)$$

For subsequent packets, if each packet can be transmitted in under 1 second, then the roundtrip

time is the same as the initial packet. Otherwise, assuming that each packet requires more than 1 second to transmit, the time taken between the start of the transmission and the completion of the roundtrip equals the time required to transmit the previous packet, minus 1 second for the delay, plus the time required to transmit the subsequent packet. This is a recurrence relation:

$$RTT_n = RTT_{n-1} - 1 + \frac{2P}{500B/s} \quad (3.2)$$

Given this recurrence relation, we can model the roundtrip time required for each packet size, as measured by ping. Figure 3.9 shows that the model's roundtrip times for each packet number, as well as the average time, all match well with the measured values. This further suggests that our above hypothesis is correct, and that no data is actually lost in the transmission of large packets.

Roundtrip Time (ms) Computed by Ping							Roundtrip Time (ms) Based on Recurrence Model						
Packet Size (B)	64	128	256	512	1024	2048	Packet Size (B)	64	128	256	512	1024	2048
Packet 1	294	577	1154	2314	4639	9455	Packet 1	256	512	1024	2048	4096	8192
2	296	582	1158	3254	7518	16348	2	256	512	1024	3096	7192	15384
3	300	586	1157	4193	10396		3	256	512	1024	4144	10288	
4	302	596	1146	5147	13274		4	256	512	1024	5192	13384	
5	292	589	1151	6076	16155		5	256	512	1024	6240	16480	
6	300	577	1158	7021	19051		6	256	512	1024	7288	19576	
7	300	585	1158	7968			7	256	512	1024	8336		
8	291	578	1164	8893			8	256	512	1024	9384		
9	297	584	1170	9862			9	256	512	1024	10432		
10	298	577	1160				10	256	512	1024			
Average Time	297	583.1	1158	6080.9	11839	12902	Average Time	256	512	1024	6240	11836	11788
Loss (%)	0	0	0	10	40	80							

Figure 3.9: A comparison between data rates from measurement and recurrence model

Disregarding measurements with packet losses, we see that the computed average data transfer rate is about 440B/s. However, this does not consider that there are extra headers in each packet, which can significantly increase the data size for small packets. Either way, we see that the data rate is relatively steady for all packet sizes below 256B. The rate is slower than the maximum theoretical rate allowed for XBee devices, and we suspect that it is possible to increase the data rate by tweaking our implementation.

Thus, we have presented examples of how our system can be used for basic Internet networking, and analyzed the performance of the network. Next, we summarize our achievements and conclude the report.

## 4 SUMMARY

For this project, we have designed and implemented a networking system, based on the XBee radio modules. We are able to connect two computers over the Internet, each using a single

XBee module, connected to the computer via USB. We are able to run a variety of Internet and network applications over this connection, and perform different operations, such as pinging, and file transfers. We have also conducted a basic performance test, to discover that the system may drop packets when they are large, but this is not conclusive. Next, we examine some potential future areas of development.

#### 4.1 POSSIBLE FUTURE DIRECTIONS

XBLink can be useful in many applications. For example, two mobile laptops (e.g. used in field work) that want to communicate with each other where there are no wireless area networks available, can use XBLink to create a network between them.

As mentioned above, our system is designed with simplicity and speed in mind, and does not incorporate any security features. We have considered some modifications that would increase the security of the system, but they are not exhaustive. For the system to be used in real-world applications, the security aspect must be carefully considered and implemented.

Another area of potential future work involves expanding the network to multiple nodes. This is easily doable for a small number of nodes, since we can use the existing topology that uses one XBee module as the coordinator. However, as the number of nodes grows large, it will become infeasible for a single module to act as the coordinator for all nodes, both due to range and complexity constraints. In this case, a more distributed topology with many coordinators would be appropriate.

#### REFERENCES

- [1] Wikipedia.org. Internet. Electronic:<http://en.wikipedia.org/wiki/Internet>.
- [2] Wikipedia.org. Internet protocol suite. Electronic:[http://en.wikipedia.org/wiki/Internet\\_protocol\\_suite](http://en.wikipedia.org/wiki/Internet_protocol_suite).
- [3] Digi International Inc. Xbee. Electronic:<http://www.digi.com/xbee/>.
- [4] Future Technology Devices International Ltd. Future technology devices international ltd. Electronic:<http://www.ftdichip.com/index.html>.
- [5] Wikipedia.org. Tun/tap. Electronic:<http://en.wikipedia.org/wiki/TUN/TAP>.
- [6] Google Project Hosting. libxbee. Electronic:<https://code.google.com/p/libxbee/>.
- [7] Wikipedia.org. Xbee. Electronic:<http://en.wikipedia.org/wiki/XBee>.
- [8] Wikipedia.org. Ieee 802.15.4. Electronic:[http://en.wikipedia.org/wiki/IEEE\\_802.15.4](http://en.wikipedia.org/wiki/IEEE_802.15.4).

- [9] SparkFun Electronics. Xbee explorer usb. Electronic:<https://www.sparkfun.com/products/retired/8687>.
- [10] Wikipedia.org. Device file. Electronic:[http://en.wikipedia.org/wiki/Device\\_file](http://en.wikipedia.org/wiki/Device_file).
- [11] Digi International Inc. Xctu. Electronic:<http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/xctu>.
- [12] Davide Brini. Tun/tap interface tutorial. Electronic:<http://backreference.org/2010/03/26/tuntap-interface-tutorial/>.
- [13] Digi International Inc. Sending data through an 802.15.4 network latency timing. Electronic:<http://www.digi.com/support/kbase/kbaseresultdet1?id=3065>.
- [14] Wikipedia.org. Ipv4. Electronic:<http://en.wikipedia.org/wiki/IPv4>.