

T05213

**Performance Evaluation of different Bluetooth voice packets on SCO Links.**



T-5213

By

**Abdur Rasheed (04/FET/MSEE/F05)**

**Danish Rafiq Khan (05/FET/MSEE/F05)**

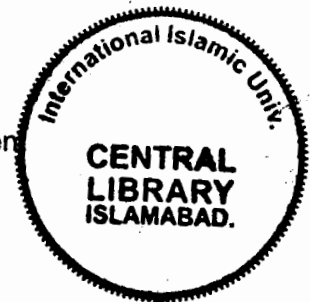
**Supervised by**

**Assistant Professor Raza Hasan Abedi**

This dissertation is submitted to I.I.U in partial fulfillment

Of the requirement for the degree of

MS Electronic Engineering



**Department of Electronics Engineering**

**Faculty of Engineering and Technology**

**INTERNATIONAL ISLAMIC UNIVERSITY ISLAMABAD (IIUI)**

**2008**

## Certificate of Approval

It is certified that we have read the project report submitted by ~~Abdul~~ Rasheed [04-FET/MSEE/F05 ] and Danish Rafiq Khan [ 05-FET/MSEE/F05 ]. It is our judgment that this project is of sufficient standard to warrant its acceptance by the International Islamic University, Islamabad for Degree of MS Electronics Engineering (MSEE).



---

**External Examiner**

**Dr. Abdul Jalil**

Associate Professor

Department of Information & Computer Science,  
PIEAS, Islamabad



---

**Internal Examiner**

**Dr. Tanweer Ahmed Cheema**

Assistant Professor

Department of Electronic Engineering  
Faculty of Engineering & Technology,  
IIU Islamabad



---

**Supervisor**

**Raza Hasan Abedi**

Assistant Professor

Department of Electronic Engineering  
Faculty of Engineering & Technology,  
IIU Islamabad

## DECLARATION

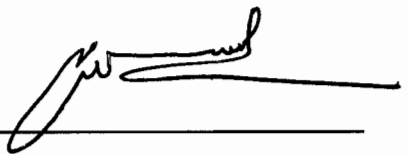
I/W, hereby declare that this software, neither as a whole nor as a part thereof has been copied out from any source. It is further declare that we have developed this software and the accompanied report entirely on the basis of our personal effort made under the guidance of our teachers.

If any part of this report to be copied out or found to be reported, We shall standby the consequences, no portion of this work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.



**Abdur/Rasheed**

**[04-FET/MSEE/F05 ]**



**Danish Rafiq Khan**

**[05-FET/MSEE/F05 ]**

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

In the name of Allah (SWT) the most beneficent, the most merciful

**An MS Final Year Project submitted to the**  
**Department of Electronics Engineering**  
**International Islamic University Islamabad**  
**In partial fulfillment of the requirements**  
**For the award of the degree of**  
**MS in Electronics Engineering**

**Dedicated To,**

**Our Parents,**

**Who always supported us and prayed for our success**

# **Acknowledgement**

First of all we would like to take this opportunity to our humble gratitude to Almighty Allah who enabled us to accomplish this task. Who helped us in every crucial stage and bestowed us the stamina to achieve our aim.

We are also fortunate to have Assistant Professor Sir Raza Hasan Abedi as our MS. Advisor. Their ability to ask the right question and attention to detail has never ceased to surprise us. They were always able to make time for us, even if we just walked into office without any notice. We would also like to mention their patience, giving us inspiration and hope when we were stuck at dead-ends.

Special thanks go to our parents and friends for their support throughout our entire academic career.

**Danish Rafiq Khan**

**&**

**Abdur Rasheed**

# Project in Brief

**Project Title:** Performance Evaluation of different Bluetooth voice packets on SCO links

**Undertaken By:** Abdur Rasheed [ 04-FET/MSEE/F05 ]  
Danish Rafiq Khan [ 05-FET/MSEE/F05 ]

**Supervised By:** Assistant professor Raza Hasan Abedi

**Date Started:** September 2007

**Date Completed:** August 2008

**Tools Used:** OMNeT++, Visual C++, MS Excel, Word

**Operating System used:** Microsoft Windows XP Professional



# ABSTRACT

The performance analysis of different Bluetooth voice packets on Synchronous Connection Oriented (SCO) links. In this way one packet is sent from one device to another device via Bluetooth on SCO links.

Different graphs and tables are shown here for the best analysis of the Bluetooth voice packets. The delay is added to voice packets so that to see that how much time it will take to reach at the destination. The time span of different voice packets are compared with one another for the best analysis on SCO link.

We proposed OMNeT Software to implement this research work. The different techniques are used to show the best analysis of the Bluetooth voice packets.

In this research the piconet is explained for Bluetooth transmission. This fully wireless scenario can be achieved because of the master/slave nature of the Bluetooth technology. All devices are peers, identified by their own unique 48-bit address, and can be assigned as a master either by function or user intervention. A master can connect up to seven slaves at the same time, forming a piconet. In the piconet one master (transmitter) and more than one slaves (receiver) up to seven connectable devices, Bluetooth layers and the protocols are also explained. The simple act of utilizing Bluetooth technology as cable replacement removes the problem of the actual physical connections. Interference can impact both the quality of an audio (Synchronous Connection Oriented [SCO]) connection or the throughput of a data (Asynchronous Connectionless [ACL]) connection. High levels of interference can interrupt communications for long enough to cause the protocol stack to timeout and abandon the link altogether. Although this is addressed in the Bluetooth specification with a frequency-hopping scheme which does provide robustness, it is still a serious consideration for some applications.

The results of different voice packets obtained from the OMNeT software are compared with one another and hence the best performance is analyzed.

# TABLE OF CONTENTS

<b>1</b>	<b>Introduction and Overview .....</b>	<b>2</b>
1.1	Wireless communication .....	3
1.2	Bluetooth concepts .....	3
1.3	OMNet Overview .....	3
<b>2</b>	<b>Literature Review .....</b>	<b>5</b>
<b>3</b>	<b>Bluetooth Overview .....</b>	<b>7</b>
3.1	Origins of Bluetooth .....	8
3.2	Bluetooth Overview .....	8
3.3	Frequency Hopping and Channels .....	10
3.4	Bluetooth Data Rates and Data Packets .....	11
3.5	Range / Distance Covered .....	12
3.6	Bluetooth Communication Topology/ Connectivity .....	13
3.6.1	Piconet Topology .....	13
3.6.2	Connecting Procedures and Modes .....	13
3.6.3	Inquiry/Discovering Procedure .....	13
3.6.4	Paging/Connecting Procedure .....	14
3.6.5	Connected Mode .....	14
3.6.6	Active Mode .....	14
3.6.7	Hold Mode .....	14
3.6.8	Sniff Mode .....	14

3.6.9	Parked Mode .....	15
3.7	<i>Bluetooth Profiles</i> .....	15
3.8	Layers of Bluetooth Architecture .....	15
3.8.1	Protocol Layers Stack.....	17
3.8.2	Service Discovery Protocol (SDP).....	17
3.8.3	RFCOM Protocol .....	17
3.8.4	L2CAP (Logical Link Control and Adaptation Protocol).....	17
3.8.5	Host to Controller Interface (HCI).....	18
3.8.6	Link Manager Protocol Layer (LMP).....	18
3.8.7	Baseband layer/ Link Controller .....	18
3.8.8	Radio Layer (RF).....	19
3.8.9	Bluetooth Controller .....	19
3.9	Security of Bluetooth and its Security Modes .....	19
<b>4</b>	<b>Introduction to OMNeT Simulator</b> .....	<b>21</b>
4.1	What is OMNeT++ .....	22
4.1.1	Components .....	22
4.1.2	Platforms .....	23
4.1.3	Licensing For OMNet ++ .....	23
4.2	Installation of the OMNeT++ .....	23
4.3	OMNeT++ Module Structure .....	24
4.4	Simulation Modeling In OMNet ++ .....	25
4.4.1	Library Modules .....	25
4.4.2	Network Moduling .....	25

4.4.3	Network Description (NED).....	26
4.5	User Interfaces .....	26
4.5.1	TKenv .....	26
4.5.2	Cmdenv .....	28
4.6	NED Overview .....	28
4.6.1	Reserved words .....	28
4.6.2	Identifiers .....	28
4.6.3	Case Sensitivity.....	29
4.6.4	Comments .....	29
4.6.5	The import Directive .....	29
4.7	Simple Module definitions .....	29
4.7.1	Compound module Definitions .....	30
4.7.2	Network Definitions .....	30
4.7.3	Generating NED Files .....	31
4.7.4	Building the network from C++ code.....	31
<b>5</b>	<b>Experimental Analysis and Results.....</b>	<b>32</b>
5.1	Experimental Setup.....	32
5.2	Compiling the Simulation .....	34
5.3	Running the Simulation .....	35
5.4	Scenario 1 .....	37
5.4.1	Delay(0,0) .....	39
5.5	Scenario 2 .....	40
5.5.1	Delay(50,100) .....	43

5.5.2	Delay(250,150) .....	47
5.5.3	Data rate(1500,200) .....	51
5.6	Formula Used .....	51
<b>6</b>	<b>Conclusion</b> .....	<b>53</b>
<b>7</b>	<b>Bibliography</b> .....	<b>55</b>
<b>8</b>	<b>Appendices</b> .....	<b>57</b>
	Source Code .....	<b>Appendix A</b>
	Vector Data Generated .....	<b>Appendix B</b>
	Scalar Data Generated.....	<b>Appendix C</b>

## LIST OF FIGURES

<b>Figure 3.1:</b>	<b>Typical Bluetooth Network A Scatter-net .....</b>	<b>10</b>
<b>Figure 3.2:</b>	<b>Bluetooth packet .....</b>	<b>12</b>
<b>Figure 3.3:</b>	<b>Bluetooth Operating Range .....</b>	<b>13</b>
<b>Figure 3.4:</b>	<b>Protocol Stack .....</b>	<b>16</b>
<b>Figure 3.5:</b>	<b>Taxonomy of Bluetooth Security Modes .....</b>	<b>20</b>
<b>Figure 4.1:</b>	<b>Module structure .....</b>	<b>24</b>
<b>Figure 4.2:</b>	<b>Tkenv User Interface in Omnet++ .....</b>	<b>27</b>
<b>Figure 5.1:</b>	<b>Scenario 1.....</b>	<b>34</b>
<b>Figure 5.2:</b>	<b>Scenario 2 .....</b>	<b>34</b>
<b>Figure 5.3:</b>	<b>Compiling the simulation .....</b>	<b>35</b>
<b>Figure 5.4:</b>	<b>Running the simulation .....</b>	<b>36</b>
<b>Figure 5.5:</b>	<b>Running the simulation interface .....</b>	<b>36</b>
<b>Figure 5.6:</b>	<b>End to End delay (device1 &amp; 2 with 0ms delay).....</b>	<b>37</b>
<b>Figure 5.8:</b>	<b>Simulation duration (with 0ms delay).....</b>	<b>38</b>
<b>Figure 5.9:</b>	<b>End to End delay (device1 &amp; 2 with 50,100 ms delay).....</b>	<b>40</b>
<b>Figure 5.10:</b>	<b>Simulation duration (with 50 &amp; 100 ms delay).....</b>	<b>42</b>

<b>Figure 5.11:</b>	<b>End to End delay (device1 &amp; 2 with 250,150 delay).....</b>	<b>44</b>
<b>Figure 5.12:</b>	<b>Simulation duration (with 250 &amp; 150 delay).....</b>	<b>46</b>
<b>Figure 5.13:</b>	<b>End to End delay (device1 &amp; 2 with 1500,200 data rate).....</b>	<b>48</b>
<b>Figure 5.14:</b>	<b>Simulation duration (with 1500 &amp; 200 data rate).....</b>	<b>50</b>

## LIST OF TABLES

<b>Table 2.1 :</b>	<b>A summary of comparisons between Bluetooth and IEEE 802.11.....</b>	<b>5</b>
<b>Table 3.1 :</b>	<b>ISM Band allocations .....</b>	<b>8</b>
<b>Table 3.2 :</b>	<b>Key Characteristics of Bluetooth Technology Descriptions .....</b>	<b>9</b>
<b>Table 3.3 :</b>	<b>Bluetooth frequency range and channels .....</b>	<b>10</b>
<b>Table 3.4 :</b>	<b>Bluetooth Data Rates .....</b>	<b>11</b>
<b>Table 3.5 :</b>	<b>Device Classes of Power Management .....</b>	<b>12</b>
<b>Table 3.7 :</b>	<b>Bluetooth Core Protocol Stack .....</b>	<b>15</b>
<b>Table 5.1 :</b>	<b>Device 1 and Device 2 delay 0ms .....</b>	<b>31</b>
<b>Table 5.2 :</b>	<b>Delay vs Time .....</b>	<b>41</b>
<b>Table 5.3 :</b>	<b>Device 1 and Device 2 delay 50ms,100ms respectively .....</b>	<b>43</b>
<b>Table 5.4 :</b>	<b>Delay vs Time .....</b>	<b>44</b>
<b>Table 5.5 :</b>	<b>Device 1 and Device 2 delay 250ms,150ms respectively .....</b>	<b>47</b>
<b>Table 5.6 :</b>	<b>Delay vs Time .....</b>	<b>48</b>
<b>Table 5.7 :</b>	<b>Device 1 and Device 2 data rate 1500,200 respectively .....</b>	<b>51</b>



# **Chapter 1**

## **Introduction and Overview**

# 1 INTRODUCTION AND OVERVIEW

This research is performance evaluation of Bluetooth voice packets on Synchronous Connection Oriented (SCO) Links. In this research voice packet has been send from one device to another via Bluetooth Technology on a SCO links in a piconet environment. The graphical result and tables are also shown firstly the performance is evaluated in the delay free environment and secondly the performance is evaluated in the presence of delay environment and then compare the performance. The delay is also included in the research for the best analysis of the voice packets transferred from one device to another. In this way to conclude that how delay affects the performance of one device sending packets to other device in piconet.

In the Bluetooth technologies there are two types of links Asynchronous Connection Less (ACL) and Synchronous Connection Oriented (SCO). New technologies have emerged that allow wireless communication. The IEEE 802.11b or Wi-Fi is becoming the choice for the networking community as it supports features that enable it to perform handovers between access points and it can effectively become a transparent wireless network expanding the static wired network. IEEE 802.11b has the maximum data capacity of 11 Mbps as IEEE 802.11a and its competitor HyperLAN2 have the greater data rates but it is not cost effective in the customer's point of view but now technologies that have wireless connectivity, low-power, cost effectiveness and ad-hoc connectivity between devices are becoming popular like Bluetooth.

As now a days different devices are being introduced having Bluetooth support in today's world the wireless network and Bluetooth enabled handheld devices are growing rapidly and networks continue to increase the problem of interference, delay, noise and distortion in the data when communicating with each other and operating at the same frequency bands. As telecommunication industry is growing the hurdles in the communication field motivated research work in the Bluetooth voice packets performance on the SCO link.

Experiments performed simulation based comparison using the OMNet simulator and evaluated the performance of voice packets in the presence of delay and interference.

## **1.1 Wireless Communication**

Wireless communication, enable one or more devices to communicate without physical connections without requiring network or peripheral cabling. Wireless communication use radio frequency transmissions, microwave or infrared communication as the means for transmitting data, where as wired technologies use cables. Wireless technologies range from complex systems, such as WiFi, WiMAX, cell phones, GPS and satellite television to simple devices such as wireless headphones, microphones, (IR) devices such as remote controls, some cordless computer keyboards and mouse, cordless phones and wireless hi-fi stereo headsets. Wireless communications is a rapidly growing segment of the communications industry, as it provides high-speed high-quality information exchange between portable devices located anywhere in the world.

## **1.2 Bluetooth**

Bluetooth is a wireless technology for creating personal networks operating in the 2.4 GHz unlicensed band, with a range of 10 to 100 meters. Networks are usually called PAN formed ad-hoc from portable devices such as cellular phones, handhelds like headset and laptops. Bluetooth offers different benefits to the users like ad-hoc network, file sharing, cable replacement, internet connectivity, wireless synchronization, multiplayer games, multimedia contents sharing and much more applications.

## **1.3 OMNeT Overview**

OMNeT++ is basically a collection of software tools and libraries which you can use to build your own simulation models. The simulation structure provides the following:

1. An OMNeT++ model is build from componets ( modules) which communicate by exchanging messages . Modules can be nested, that is several modules can be grouped together to form a compound module. When creating the model you need to map your system into a hierarchy of communicating modules.
2. Define the model structure in the NED language. You can edit NED in a text editor or in GNED, the graphical editor of OMNeT++.

3. The active components of the model (simple modules) have to be programmed in C++, using the simulation kernel and class library.
4. It provides a suitable `omnetpp.ini` to store OMNeT++ configuration and parameters to your model. A config file can describe several simulation runs with different parameters.
5. Build the simulation program and run it. Link the code with the OMNeT++ simulation kernel and one of the user interfaces OMNeT++ provides. There are command (line batch) and interactive graphical interfaces.
6. Simulation results are written into output vector and output scalar files. You can use Plove and Scalars to visualize them. For more thorough statistical analysis, you can use standalone packages such as R, Octave or Matlab, or even spreadsheet.

This report consists of six chapters, 1st chapter is introduction and overview, 2nd chapter describes the overview of the related work has accomplished in this field. Third Chapter highlights Bluetooth its architecture, connectivity and protocols. Fourth chapter is an introduction and overview of OMNeT simulator, its installation and network modeling features. Chapter 5 reveals experimental analysis and simulation results carried out in OMNeT (OMNet++ 3.3). In the last chapter an effort is made to conclude findings with respect to the experimental results that may be helpful to the researchers to improve the performance of Bluetooth device.

## Chapter 2

### 2 LITERATURE REVIEW

In [1] author has discussed in this paper that the Radio is the interface between the on-air channel medium and the Base band. The Base band layer is responsible for channel coding and decoding. It digitizes the signals received by the radio for passing up the stack and it formats the data it receives from the Link Controller for transmission over the channel. The Link Controller is responsible for establishing and maintaining the links between Bluetooth units. The Link Manager Protocol (LMP) handles piconet management and link configuration. It also includes procedures for enforcing link security, such as encryption and authentication procedures.

In [2] author has discussed about the Bluetooth distributed voice access protocol (DVAP). In DVAP, Bluetooth mobiles collect connectivity information which is provided to the base station infrastructure. Mobile node migration is then used to reduce blocking when a Bluetooth base station is carrying a full load of active SCO links. DVAP is intended for use in systems where deployments generate partial overlapping radio coverage situations. As in standard Bluetooth access point designs, in DVAP, Bluetooth enabled devices use the standard Bluetooth inquiry procedures to initially associate with a base station, where it is parked.

In [3] author has discussed the principal difference between the Bluetooth and IEEE 802.11b standards that the Bluetooth is connection-oriented while 802.11b is connectionless. This implies that Bluetooth units need to set up connections before they can send any data, while for 802.11b, units can directly send data to any other unit in range. Table 2.1 shows a summary of the more detailed comparisons between the two technologies.

Support	Bluetooth	IEEE 802.11b
Media Access Control	Based on controlling unit (master)	Random-access-oriented
Neighbor Discovery	Standardized discovery using the INQUIRY procedure	No defined way to discover unknown devices (may use broadcasting), but known devices can be directly addressed
Multihop PAN's	Involves scatternets -	Straightforward - no piconet

<b>Support</b>	<b>Bluetooth</b>	<b>IEEE 802.11b</b>
	interconnected piconets	architecture, all nodes are peers
Power Consumption	Uses polling between masters and slaves, offers power saving modes	Units may receive packets from other units at any time, so receivers need to be active for long periods.

**Table 2.1 : A summary of comparisons between Bluetooth and IEEE 802.11**

In [4] author has discussed the Bluetooth SCO link as a standard communication link for high quality real time voice data. In case, if the high quality is not required SCO link can be replaced by ACL link. Although ACL shows a slight delay as compared to the HV2 packet type, but perform better in case of HV3 packet. Another observation shows that the transmission of voice over IP (TCP) connection when it is in communication with a slave of piconet, ACL outperforms than that of HV2 and HV3 packets of the SCO link.

In [5] author referred to as the emergence of several radio technologies, such as Bluetooth and IEEE 802.11, operating in the 2.4 GHz unlicensed ISM frequency band, may lead to signal interference and result in significant performance degradation when devices are collocated in the same environment using a slower hop rate for Bluetooth (i.e. longer packet sizes) may cause less interference to WLAN. Second, Bluetooth voice represents the worst type of interference for WLAN. In addition, the WLAN performance seems to degrade as the Bluetooth offered load is increased. Finally, the use of error correcting block codes in the Bluetooth payload does not improve performance. The errors caused by interference are often too many to correct.

In [6] author has discussed the soft-blocking problem in the Bluetooth system increases as the number of slaves and piconets increases. To determine the soft-blocking problem, measure the throughput as the function of data rate, number of transition channels and number of access point that is slaves. The throughput performance severely degraded as the number of access points increases.

# **Chapter 3**

## **Bluetooth Overview**

### 3.1 Origins of Bluetooth

Bluetooth began as an open standard project in 1994 by Ericsson in Sweden. It was originally named multi-communicator (MC) link the goal was to develop a wireless communication standard that would support short-range voice and data transfers between multiple devices. Four years later, in 1998, four other companies, IBM, Intel, Nokia, and Toshiba, joined with Ericsson to form a special interest group (SIG) SIG which serves as the governing body of specification renamed the standard to Bluetooth. Today, the group consists of nine companies 3Com, Lucent Technologies, Microsoft, Motorola, IBM, Intel, Nokia, Toshiba and of course Ericsson there are also hundreds of associate and adopter member companies in the telecommunication and computing industries. The Bluetooth SIG manages on going technical working group for short-range wireless specification for connecting mobile devices Bluetooth provides connectivity among devices like Computers, wireless headsets, printers, personal digital assistants (PDAs), mobile phones, cordless phones, pagers, cameras, PC cards, fax machines, MP3, MP4 players and laptops.

### 3.2 Bluetooth Overview

Bluetooth is an open standard for short-range digital radio. It provides low-cost, low-power, and low-profile technology that offer fast and reliable transmission for both voice and data communications creating small wireless networks on an ad hoc basis. The goal of Bluetooth is to connect different devices (PDAs, cell phones, printers, faxes, etc.) together wirelessly in a small environment such as an office or home and forming a PAN on a temporary and random basis.

Country	Frequency Band (MHz)	Number Of Channels
United States	2,400-2,483.5	79
Europe	2,400-2,483.5	79
Spain	2,444.5-2,475	23
France	2,446.5-2,483.5	23
Japan	2,471-2,497	23

Table 3.1 ISM Band allocations

Bluetooth operates in the ISM (industrial, scientific, medical applications) band which is unlicensed band with different frequency and number of channel allocation as shown in Table 3.1. In Bluetooth network eight devices can be connected with each other where one device will be master and rest of the seven will be acting as slaves forming a "piconet". Any slave device in



a piconet can act as a master in other piconet forming a network called scatter-net as depicted in Figure 3.1. Master can assign active member address (3 bit), parked member address (8 bit) and access request address (8 bit) to any slaves participating in the piconet and also able to change the mode from active to sniff, hold and park at any time during the communication. Master can be identified by 48 bit address as each device has its own unique address. Master is managing and controlling the whole piconet network communication like channel, frequency hopping and power level. Table 3.2 shows some characteristics of Bluetooth technology.

<b>Characteristics</b>	<b>Description</b>
Physical Layer	Frequency Hoping Spread Spectrum (FHSS)
Frequency Band	2.4 – 2.4835 GHz (ISM band)
Hop Frequency	1,600 hops/sec
Data rate	1 Mbps (raw). Higher bit rates are anticipated.
Data and Network Security	Three modes of security (none, link- level, and service level)
Operating Range	About 10 meters can be extended to 100 meters.
Throughput	Up to approximately 720 kbps.
Positive Aspects	No wire and cables for many interfaces. Ability to penetrate walls and other obstacles. Costs are decreasing and Low power and minimal hardware
Negative Aspects	Possibility for interference with other ISM band technologies. Relatively low data rates. Signals leak outside desired boundaries

**Table 3.2 Key Characteristics of Bluetooth Technology Descriptions**

Bluetooth provides two different types of physical links:

**Asynchronous connectionless links (ACLs)** are most often used for data communication and is point to multipoint link between master and all slaves participating on the piconet. Packet retransmission corrects error packets. Master can establish ACL link on a per slot basis to any slave.

**Synchronous connection-oriented (SCO)** links create a circuit-switched, scheduled, point-to-point link between a master and a slave with no packet retransmission. The connection is mainly used for transmission of time bound data like voice. Master maintains SCO link by using reserved time slots for transmission at regular interval.

In summary, Bluetooth provides a better solution for cable replacement in terms of short range, low power consumption, low transmission rate, minimal hardware and low cost.

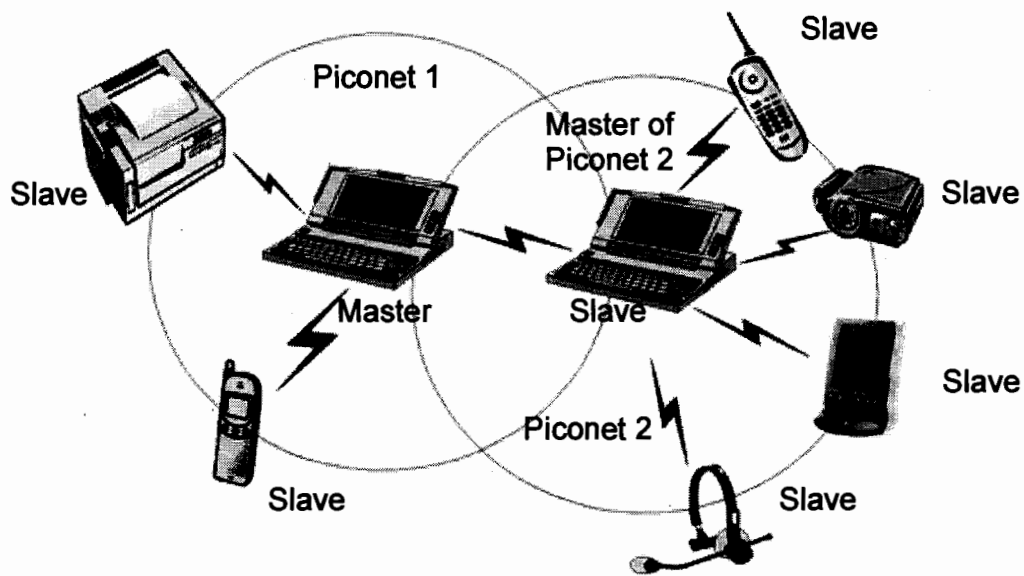


Figure 3.1 Typical Bluetooth Network a Scatter-net

### 3.3 Frequency Hopping and Channels

The radio unit of Bluetooth device operates in 2.4 GHz band. Bluetooth utilizes a frequency-hopping spread-spectrum (FHSS) technique of transmission by using 79 different radio frequency channels and by rapidly switching to another frequency about 1,600 hops per second to limit and minimize the interference problem as multiple devices can transmit and operate at the same range of frequency because of ISM band. Bluetooth uses 625 microsecond single channel transmission by jumping to another new channel after 625 microseconds this process continues by getting the random frequencies.

Frequency	Channels
2.400-2.4835 GHz	$F = 2402 + k, k = 0, \dots, 78$ MHz

Table 3.3 Bluetooth frequency range and channels

Hopping algorithm of Bluetooth uses 79 frequencies as we can see in Table 3.3. Hopping sequence will be known to master and slaves during the establishment of connection.

### 3.4 Bluetooth Data Rates and Data Packets

Theoretically data rate of Bluetooth devices is 1 Mbps but actual data rate it achieves due to overhead is 723.2 Kbps for DH5 packet as shown in Table 3.4. The transmission of packets covers single, three or five time slots. Packet type of DH1, HV1, HV2, HV3, DV AND AUX1 are single slot packets. DM3, DH3 are three slots packets. DM5, DH5 are five slots packets and NULL, POLL, FHS and DM1 are control packets. Data medium (DM), Data high (DH) and high quality voice (HV) are packet types in Bluetooth. Types of DM packets are DM1, DM3 and DM5. Types of DH packets are DH1, DH3 and DH5. Types of HV packets are HV1, HV2, HV3 AND DV. Data high (DH) rate achieves higher data rates by using less error correction in the packets. Data medium (DM) rate achieves a lower bit error rate probability by using more error correction (16 bit CRC) in the packets. HV1 packets carries 80 bits, HV2 carries 160 bits, HV3 carries 240 bits and DV is a combined data (150 bits) and voice (80 bits) packet.

The Bluetooth packet contains a 72-bit access code, a 54-bit header, and a 0 to 2,745-bit payload. The access code is subdivided into a 4-bit preamble, a 64-bit synchronization word, and a 4-bit trailer. Header is subdivided into a 3 bit AM\_ADDR, 4 bit TYPE, 1 bit flow control, 1 bit arqn, 1 bit seqn and 8 bit HEC.

**Table 3.4 Bluetooth Data Rates**

Packet Type	Payload bits	Max Symmetric Data Rate (Kbps)	Forward Asymmetric Data Rate (Kbps)	Reverse Symmetric Data Rate (Kbps)
DM1	17	108.8	108.8	108.8
DH1	27	172.8	172.8	172.8
DM3	121	258.1	387.2	54.4
DH3	183	390.4	585.6	86.4
DM5	224	286.7	477.8	36.3
DH5	339	433.9	723.2	57.6

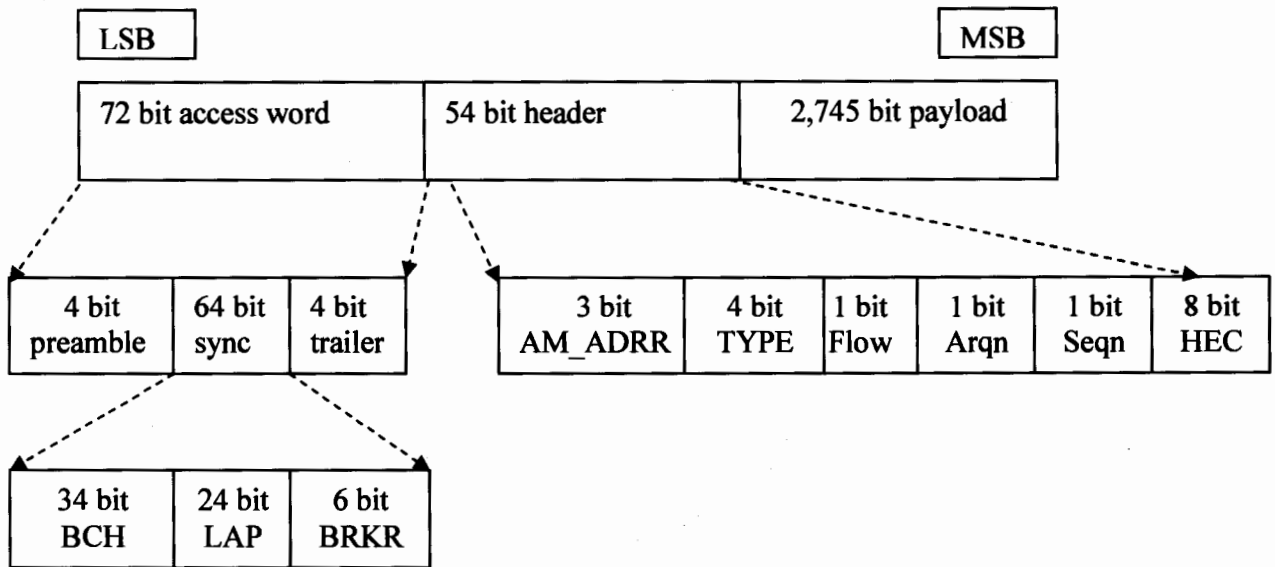


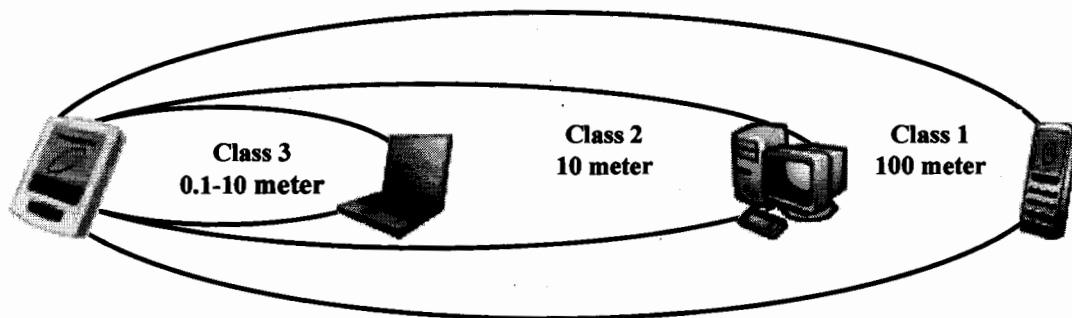
Figure 3.2 Bluetooth Packet

### 3.5 Range / Distance Covered

Bluetooth supports three types of classes class 1, class 2 and class 3 with different capabilities. Class 1 device with operating range of up to 100 meters and power level 100 mW. Class 2 devices with operating range of up to 10 meters and power level 2.5 mW. Class 3 devices with operating range from 0.1 to 10 meter and power level 1mW. The three types of classes are summarized in Table 3.5.

Table 3.5 Device Classes of Power Management

Type	Power	Power Level	Operating Range
Class 1 Devices	High	100 mW (20 dBm)	Up to 100 meters (300 feet)
Class 2 Devices	Medium	2.5 mW (4 dBm)	Up to 10 meters (30 feet)
Class 3 Devices	Low	1 mW (0 dBm)	0.1– 10 meters (less than 30 feet)



**Figure 3.3 Bluetooth Operating Range**

## **3.6 Bluetooth Communication Topology/ Connectivity**

### **3.6.1 Piconet Topology**

In piconet two or more devices connected with each other, and synchronized to master's clock forming a piconet. One device participating in one piconet can be connected with the other piconet forming a scatternet. So a slave in one piconet can be a master or slave in other piconet. Each piconet has different physical channel, piconet clock and hopping sequence.

### **3.6.2 Connecting Procedures and Modes**

The main goal of Bluetooth enabled device is to be connected to other Bluetooth enabled devices (in a piconet) and exchanging data with the other device. Inquiry/discovering and paging/connecting are connecting procedures in a piconet and there are number of modes which a device can be switched during the communication.

### **3.6.3 Inquiry/Discovering Procedure**

It enables a device to discover which devices are in range and determine the address and clocks for the devices. Inquiry procedure involves a source (inquiring device) sending out inquiry packets and then receiving the inquiry reply. The destination (discoverable devices) that receives the inquiry packets will be in the inquiry scan state to receive the inquiry packets. The destination will then enter the inquiry response state and send an inquiry reply to the source. After this procedure of inquiring/discovering has completed a connection can be established

using paging/connecting procedure describes on the next section. The inquiry / discovering procedure is asymmetrical.

#### **3.6.4 Paging/Connecting Procedure**

A unit that establishes a connection will carry out a page procedure and will automatically become master of the connection. A device i.e source pages another device i.e destination. The destination receives the page and sends a reply to the source device. The source sends an FHS packet to the destination. The destination sends its second reply to the source device and then both the devices then switch to the source channel parameters. The connection state starts with a POLL packet sent by the master to verify that slaves has switched to master timing and channel frequency hope. Paging / connecting procedure asymmetrical. One device is in paging state while the other is in connectable state.

#### **3.6.5 Connected Mode**

When the devices are physically connected to each other in a piconet master can change the connected mode from active to sniff, hold or parked mode of any device within that piconet. Different types of modes are described below.

#### **3.6.6 Active Mode**

In this mode Bluetooth unit actively participates on the channel. Master schedules the transmission based on the traffic demands and active slave listen in the master to slave slots for packet transmission.

#### **3.6.7 Hold Mode**

In this mode Master device can put slave device into hold mode where a internal timer is running. Data transfer restarts instantly when device transition out of hold mode . Hold mode is used when connecting several piconets. Hold mode is not a general device mode but applies to unreserved slots on the physical link. In this mode, slots reserved for operation of the synchronous link type SCO becomes active but all asynchronous links are inactive.

#### **3.6.8 Sniff Mode**

Devices synchronized to a piconet can enter power saving modes in which device activity is lowered. In sniff mode a slave listens to the piconet at reduced rate, thus reducing its duty cycle. Sniff mode is not a general device mode but applies to the default ACL logical transports. When

in this mode, the availability of these logical transports is modified by defining a duty cycle consisting of periods of presence and absence. Devices that have their default ACL logical transports in sniff mode may use the absent periods to engage in activity on another physical channel, or to enter reduced power mode.

### 3.6.9 Parked Mode

In this mode device is still synchronized to the piconet but does not participate in the traffic. Parked devices have given up their MAC (AM\_ADDR) address and occasional listen to the traffic of the master to resynchronize and check on broadcast messages. A slave device may remain connected to a piconet but have its physical link in the parked state. In this state the device cannot support any logical links to the master. When the slave is in parked mode the communication between the master and slave becomes inactive in the physical link.

## 3.7 Bluetooth Profiles

A Bluetooth enabled device participating in Bluetooth network has its own profile depending on the application. For example a headset connected with the mobile phone using the headset profile enabling the communication between mobile phone and headset. Profiles are set of communication methods through which one device communicate with the other. The profiles can be Advanced Audio Distribution Profile (A2DP), Basic Imaging Profile (BIP), File Transfer Profile (FTP), Hands-Free Profile (HFP), Synchronization Profile (SYNC), Basic Printing Profile (BPP) etc.

## 3.8 Layers of Bluetooth Architecture

The lowest Bluetooth core protocol layers are

L2CAP layer	Responsible for managing the ordering of submission of PDU fragments to the baseband and scheduling
Link Manager Protocol layer	Responsible for all aspects of a Bluetooth connection, such as power control, roles, encryption etc.
Link Controller layer	Responsible for the encoding and decoding of Bluetooth packets from the data payload and parameters related to the physical channel, logical transport and logical link
Radio layer	Responsible for the actual transmitting and receiving of packets of information on the physical channel

**Table 3.6 Bluetooth Core Protocol Stack**

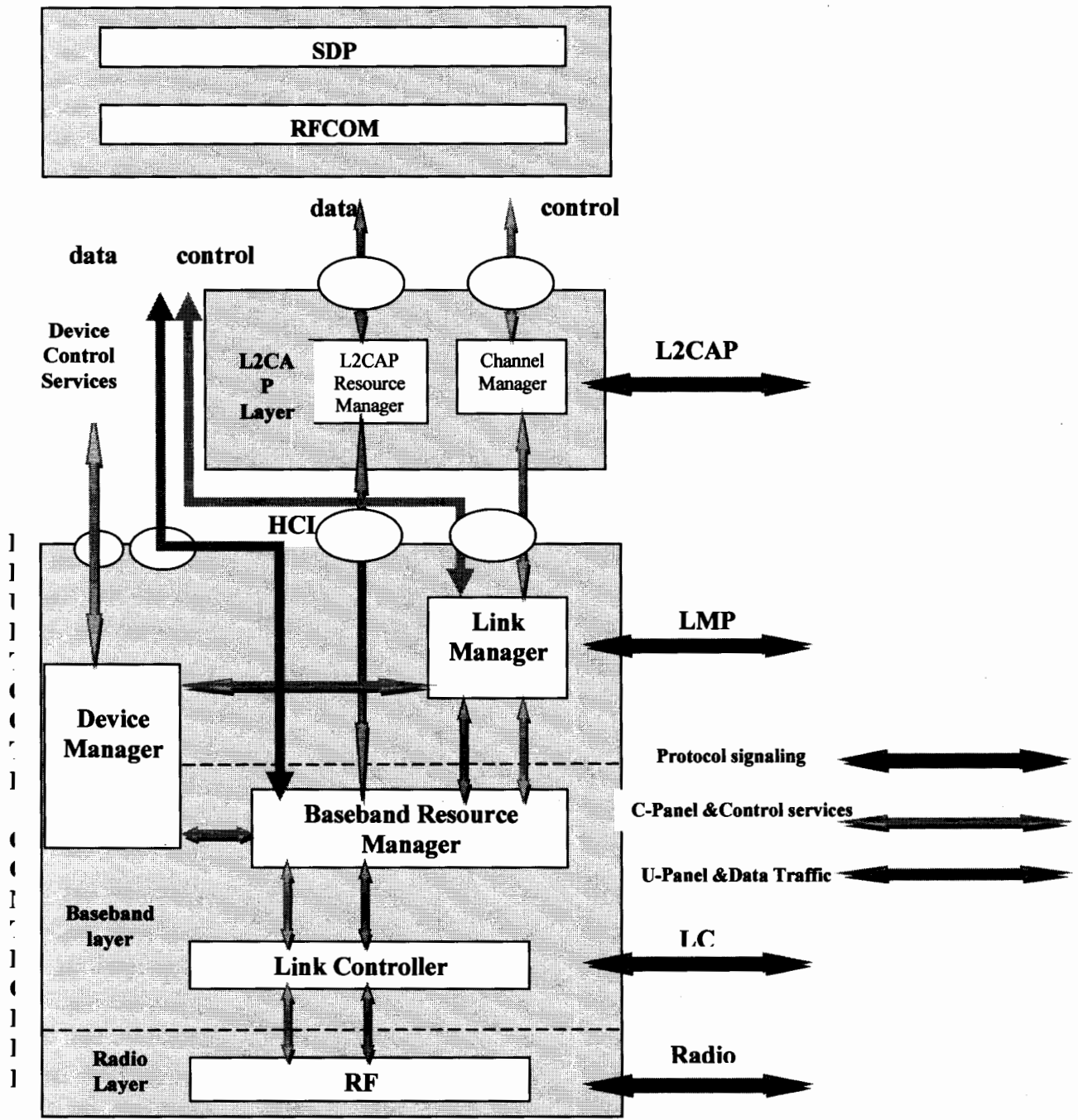


Figure 3.5 Protocol Stack



### **3.8.1 Protocol Layers Stack**

Protocol layers stack of Bluetooth covers the four lowest layers (radio layer, Link controller layer/Baseband layer, Link manager protocol layer, logical link control and adaptation protocol (L2CAP) layer) with host controller interface (HCI) as an intermediate layer and RFCOM, service discovery protocol (SDP) as higher layer protocol.

### **3.8.2 Service Discovery Protocol (SDP)**

It provides application to discover which services are provided by or available through Bluetooth devices. SDP focus on discovering services available from or through Bluetooth devices. It also allows applications to determine the characteristics of those available services. SDP uses a request /response model where each communication consists of one request protocol data unit (PDU) and one response PDU. SDP services stores the information related to service record, service attribute and service class. Service discovery methods are searching for services and browsing for the services.

### **3.8.3 RFCOM Protocol**

RFCOM protocol provides emulation of RS232 serial port over the L2CAP protocol layer. RFCOMM protocol supports up to 60 simultaneous connections between two Bluetooth devices. A two devices type exists in RFCOM Type 1 devices (these are communication end point devices e.g printers, scanners, computers, laptop) and Type 2 devices (these are the devices that are part of communication e.g modem).

### **3.8.4 L2CAP (Logical Link Control and Adaptation Protocol)**

L2CAP supports higher level protocol multiplexing, packet segmentation and reassembly, conveying quality of service information. It provides connection oriented and connectionless data services to upper layers protocols. L2CAP layer works with the Channel manager (The channel manager is responsible for creating, managing, and destroying L2CAP channels for the transport of service protocols and application data streams) and L2CAP resource manager (The L2CAP resource manager block is responsible for managing the ordering of submission of PDU fragments to the baseband ) to perform the different functions.

### **3.8.5 Host to Controller Interface (HCI)**

HCI provides a command interface to the Baseband link controller and Link Manager and access to the hardware status and control registers. An HCI link command provides Host with the ability to control the Link layer connections to other Bluetooth devices. HCI provides commands for accessing Bluetooth hardware capabilities. HCI supports the Link controller, Link policy, Host controller, baseband commands, and status information parameters. HCI commands and events are the generic events, controller information, controller configuration, device setup, device discovery, connection setup, connection state, remote information, piconet structure, QOS, physical links, Link information and testing.

### **3.8.6 Link Manager Protocol Layer (LMP)**

LMP works with Link Manager for link setup and control. Link manager carries out link setup, authentication, link configuration, link release and other protocols services. It discovers other remote Link manager's and communicates them via LMP. LMP consists of a number of PDU which are sent from one device to another determined by AM\_ADDR in the packet header. LMP PDU's are used for general response, authentication, paging, inquiry, change link key, change current link key, encryption, LMP version, switch of master slave role, hold mode, sniff mode, park mode, power control, quality of service, SCO link, pairing connection establishment and release.

### **3.8.7 Baseband layer/ Link Controller**

It describes the specification of link controller (LC) which carries out the baseband protocol and low level link routines. It manages physical channels and link apart from other services like error correction, hop selection and Bluetooth security. Baseband protocol is implemented as a Link Controller which works with Link manager for carrying out link level routines like connection and power control. It also manages asynchronous and synchronous links, handles packets and does paging and inquiry to access and inquire Bluetooth devices in the area. The link controller is responsible for the encoding and decoding of Bluetooth packets from the data payload and parameters related to the physical channel, logical transport and logical link. Link controller works with the Device Manager (which controls the general behavior of the Bluetooth enabled device, managing the device local name, any stored link keys, and other functionality) and Baseband resource manager (which is responsible for all access to the radio medium).

### **3.8.8 Radio Layer (RF)**

The RF block is responsible for transmitting and receiving packets of information on the physical channel. A control path between the baseband and the RF block allows the baseband block to control the timing and frequency carrier of the RF block. The RF block transforms a stream of data to and from the physical channel and the baseband into required formats.

### **3.8.9 Bluetooth Controller**

The lowest three layers are grouped into a subsystem known as the Bluetooth controller. This provides interface between the Bluetooth controller and the other layers including the L2CAP, service layers and higher layers (RFCOMM and SDP) as discussed previously.

## **3.9 Security of Bluetooth and its Security Modes**

In Bluetooth there are three security services which are as follow:

- Services that require authorization and authentication.
- Services that require authentication only.
- Services that are open to all devices.

Bluetooth supports three security modes but one mode can be active for a device participating in the piconet. The three modes are the following:

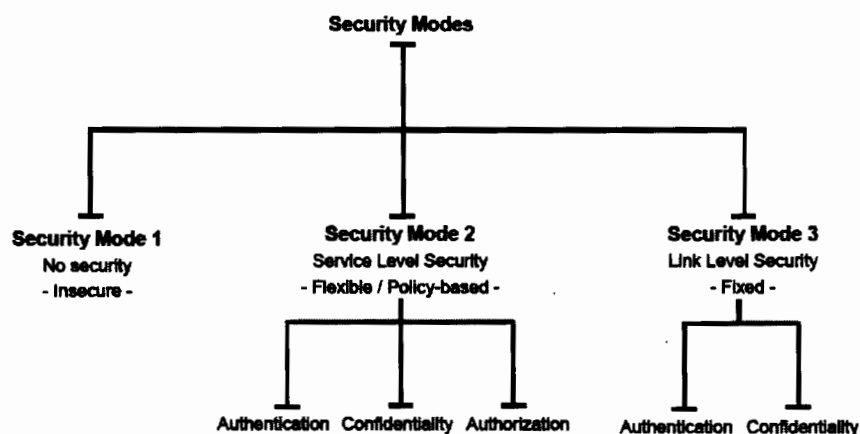
- **Security Mode 1**      Nonsecure mode
- **Security Mode 2**      Service-level enforced security mode
- **Security Mode 3**      Link-level enforced security mode

Security Mode 1 is a non secure mode in which devices will not initiate any security procedures like authentication, encryption, authorization and confidentiality. Security mode 1 is used by those applications in which security is not necessary e.g exchanging things to do, notes and business cards etc.

Security Mode 2 provides the service level security, in which security procedures are initiated after channel establishment. In this mode a centralized security manager is managing and controlling policies for accessing and restricting different services to the devices.

Security Mode 3 provides link level security in this mode security procedures are initiated before the channel establishment. In this mode a secret link key generated through pairing procedure is shared between the two devices.

The Bluetooth modes are depicted in Figure 3.6.



**Figure 3.6 Taxonomy of Bluetooth Security Modes**

## **Chapter 4**

### **Introduction to OMNeT Simulator**

## **4.1 What is OMNeT++**

OMNeT++ is a discrete event simulation environment. Its primary application area is the simulation of communication networks, but because of its generic and flexible architecture, is successfully used in other areas like the simulation of complex IT systems, queueing networks or hardware architectures as well.

OMNeT++ provides component architecture for models. Components (modules) are programmed in C++, then assembled into larger components and models using a high-level language (NED). Reusability of models comes for free. OMNeT++ has extensive GUI support, and due to its modular architecture, the simulation kernel (and models) can be embedded easily into your applications.

Although OMNeT++ is not a network simulator itself, it is currently gaining widespread popularity as a network simulation platform in the scientific community as well as in industrial settings, and building up a large user community.

OMNeT++ has its distant roots in OMNeT, a simulator written in Object Pascal by Dr. György Pongor. The simulator can be used for:

- Traffic modeling of telecommunication networks
- Protocol modeling
- Modeling queueing networks
- Modeling multiprocessors and other distributed hardware systems
- Validating hardware architectures
- Evaluating performance aspects of complex software systems
- Modeling any other system where the discrete event approach is suitable.

### **4.1.1 Components**

- simulation kernel library
- compiler for the NED topology description language (nedc)
- graphical network editor for NED files (GNED)
- GUI for simulation execution, links into simulation executable (Tkenv)

- command-line user interface for simulation execution (Cmdenv)
- graphical output vector plotting tool (Plove)
- graphical output scalars visualization tool (Scalars)
- model documentation tool (opp\_neddoc)
- utilities (random number seed generation tool, makefile creation tool, etc.)
- documentation, sample simulations, etc.

#### 4.1.2 Platforms

OMNeT++ works well on multiple platforms. It was first developed on Linux. Omnet++ runs on most Unix systems and Windows platforms (works best on NT4.0, W2K or XP).

The best platforms used are:

- Solaris, Linux (or other Unix-like systems) with GNU tools
- Win32 and cygwin32 ( Win32 port of gcc)
- Win32 and Microsoft Visual C++

#### 4.1.3 Licensing For OMNet ++

OMNeT++ is free for any non-profit use. The author must be contacted if it is used in a commercial project. The GNU General Public License can be chosen on Omnet++.

## 4.2 Installation of the OMNeT++

### Prerequisites

- **Windows XP, Windows 2000, NT4.0 or other NT-derivative.** OMNeT++ is not supported by WIN95/98/ME, because it uses the Win32 Fiber API which is only available on NT systems. Moreover, the make files and some batch (.cmd) files use syntax which is not supported by the old command.com.
- **Microsoft Visual Studio 6.0,** with at least Service Pack 1 installed. Recommend is Service Pack 5. First install Visual C++ 6.0 and then install OMNeT++.
- **40MB free space on your hard disk.** To compile from source, it requires about 20MB more.

### 4.3 OMNeT++ Module Structure

- Overall structure of modules with connections plus simulation kernel

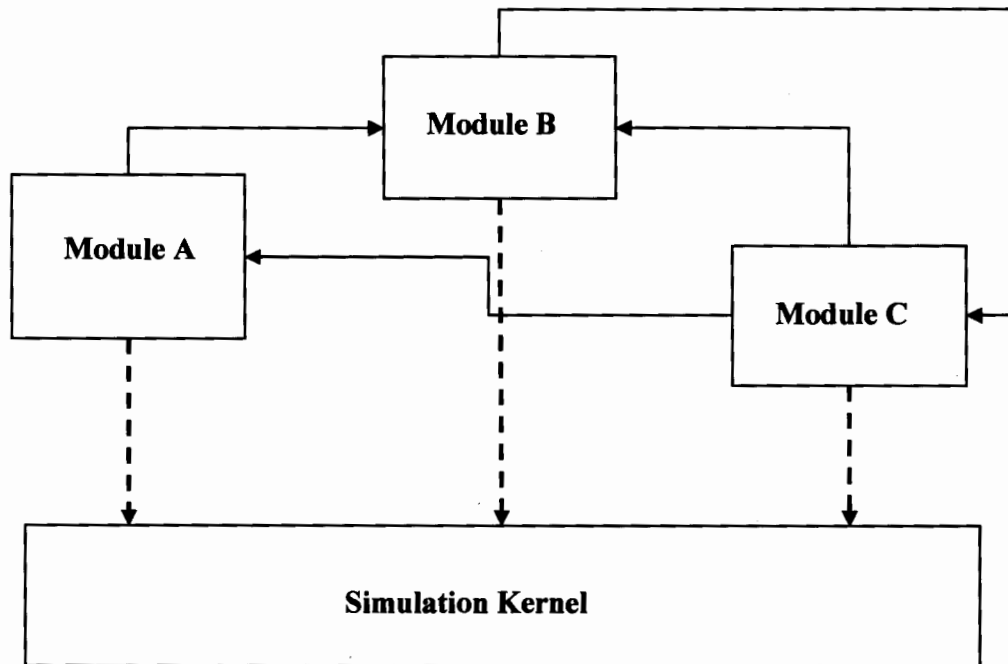


Figure 4.1 Module structure

#### Modules

- Modules exchange messages like arrival of a message at a module is an event.
- As OMNeT++ is object-oriented, all modules are instances of certain classes, representing “module types”.
- These classes must be derived from a specific class, `cSimpleModule`, an abstract class which provides basic functionality for a module.

#### Module example

```
// ===== in mymodule.h:  
#include "omnetpp.h"  
class MyModule: public cSimpleModule  
{  
// a macro that calls / creates constructors
```



```

// and sets up inheritance relationships:
Module_Class_Members (MyModule, cSimpleModule, 0)
// user-specified functionality follows:
.....
};
// ===== in mymodule.cc:
// announce this class as a module to OMNeT:
Define_Module (MyModule);

```

## 4.4 Simulation Modeling In OMNet ++

The following are types of modeling that can be used:

- communication protocols
- computer networks and traffic modeling
- multi-processor and distributed systems
- administrative systems
- And any other system where the discrete events approach is suitable.

### 4.4.1 Library Modules

Object libraries can be made using simple modules. The best simple modules to be used for library modules are the ones that implement:

- Physical/Data-link protocols: Ethernet, Token Ring, FDDI, LAPB etc.
- Higher layer protocols: IP, TCP, X.25 L2/L3, etc.
- Network application types: E-mail, NFS, X, audio etc.
- Basic elements: message generator, sink, concentrator/simple hub, queue etc.
- Modules that implement routing algorithms in a multiprocessor or network

### 4.4.2 Network Moduling

A model network consists of “nodes” connected by “links. The nodes representing blocks, entities, modules, etc, while the link representing channels, connections, etc. The structure of how fixed elements (i.e nodes) in a network are interconnected together is called topology.

Omnet++ uses NED language, thus allowing for a more user friendly and accessible environment for creation and editing. It can be created with any text-processing tool (perl, awk, etc).

### 4.4.3 Network Description (NED)

NED language describes the modular description of networks. The network description consists of a number of component descriptions such as channels, simple and compound module types, parameters and gates etc and can be used in various network descriptions. Thus, it is possible for the user to customize his or her personal library of network descriptions.

A NED description can contain the following components, in arbitrary number or order:

- import statements
- channel definitions
- simple and compound module declarations
- system module declarations

## 4.5 USER INTERFACES

The OMNet++ provides two types of user interfaces which are:

- **Tkenv**: Tk-based graphical, windowing user interface (X-Window, Win95, WinNT etc...)
- **Cmdenv**: command-line user interface for batch execution

Simulation is tested and debugged under Tkenv, while the Cmdenv is used for actual simulation experiments since it supports batch execution.

### 4.5.1 Tkenv

Tkenv is a portable graphical user interface. It has support of tracing, debugging, and simulation execution and has the ability to provide simulation information at any point during the execution. This feature makes Tkenv a good GUI interface for the simulations. A snapshot of a Tkenv interface is shown in figure 4.2.

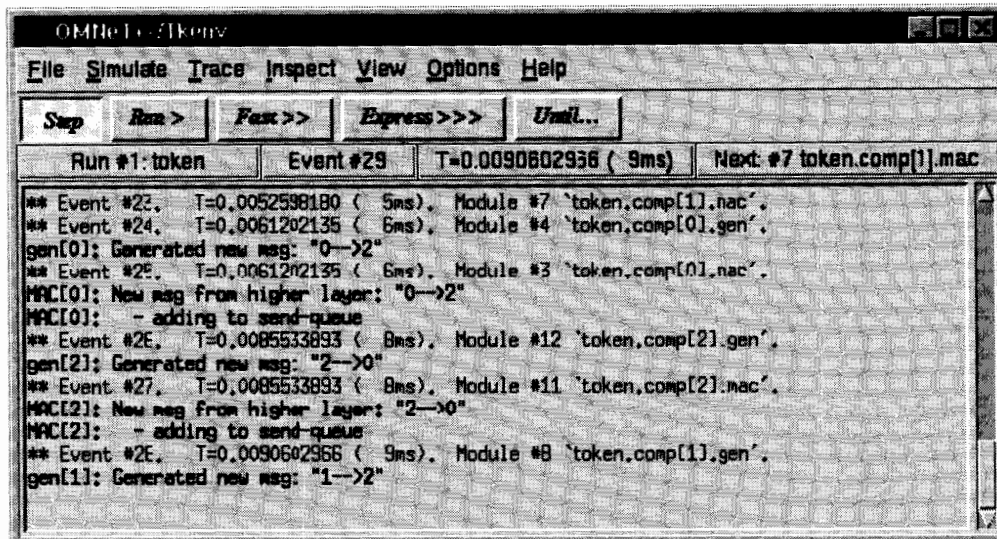


Figure 4.2 Example of a Tkenv User Interface in Omnet++.

Important features in Tkenv:

- Separate window for each module's text output
- Scheduled messages can be watched in a window as simulation progresses
- Event-by-event execution
- Execution animation
- Labeled breakpoints
- Inspector windows to examine and alter objects and variables in the model
- Graphical display of simulation results during execution. Results can be displayed as histograms or time-series diagrams.
- Simulation can be restarted
- Snapshots (detailed report about the model: objects, variables etc.)

It is recommended for testing and debugging when used with gdb or xxgdb. Tkenv provides a good environment for experimenting with the model during executions and verification of the correct operation during the simulation program and also able to display simulation results during execution.

### 4.5.2 Cmdenv

Cmdenv is designed primarily for batch execution. It is a portable and small command line interface that provides fast compilation and runs on all platforms. Cmdenv simply executes all simulation runs that are described in the configuration file.

## 4.6 NED Overview

The topology of a model is specified using the NED language. The NED language supports the modular description of a network that may consist of a number of component descriptions (channels, simple/compound module types). The channels, simple modules and compound modules of one network description can be reused in another network description.

Files containing network descriptions have a .ned suffix. NED files can be loaded dynamically into simulation programs, or translated into C++ by the NED compiler and linked into the simulation executable.

### 4.6.1 Reserved words

The reserved words of the NED language are:

Import	channel	endchannel	simple	endsimple
Module	endmodule	error	delay	datarate
const	parameters	gates	submodules	connections
gatesizes	if	for	do	endfor
network	endnetwork	nocheck	ref	ancestor
true	false	like	input	numeric
string	bool	char	xml	xml doc

### 4.6.2 Identifiers

Identifiers are the names of modules, channels, networks, submodules, parameters, gates, channel attributes and functions. Identifiers must be composed of letters of the English alphabet (a-z, A-Z), numbers (0-9) and the underscore “\_”. Identifiers may only begin with a letter or the underscore. If you want to begin an identifier with a digit, prefix the name you’d like to have with an underscore, e.g. \_3Com.

### 4.6.3 Case sensitivity

In OMNet all identifiers and the network description are case sensitive. For example, TCP and Tcp are two different names.

### 4.6.4 Comments

Comments in NED file is just like C++ comments that begin with a double slash '//', and last until the end of the line. Comments are ignored by the NED compiler.

### 4.6.5 The import directive

The import directive is used to import declarations from another network description file. When a file is imported, only the declaration information is used. Also, importing a .ned file does not cause that file to be compiled with the NED compiler when the parent file is NED compiled, i.e., one must compile and link all network description files not only the top-level ones.

74 52/3

Example:

```
import "ethernet"; // imports ethernet.ned
```

## 4.7 Simple module definitions

Simple modules are the basic building blocks for other (compound) modules. Simple module types are identified by names. By convention, module names begin with upper-case letters. A simple module is defined by declaring its parameters and gates. Simple modules are declared with the following syntax:

```
simple SimpleModuleName
    parameters:
    //...
    gates:
    //...
endsimple
```

### 4.7.1 Compound Module Definitions

Compound modules are modules composed of one or more submodules. Any module type (simple or compound module) can be used as a submodule. Like simple modules, compound modules can also have gates and parameters, and they can be used wherever simple modules can be used.

A compound module definition looks similar to a simple module definition: it has gates and parameters sections. There are two additional sections, submodules and connections.

The syntax for compound modules is the following:

```
module CompoundModule
    parameters:
    //...
    gates:
    //...
    submodules:
    //...
    connections:
    //...
endmodule
```

### 4.7.2 Network definitions

Module types can be defined as module declarations (compound and simple module declarations) for simulation model a network definition is necessary that declares a simulation model as an instance of module type. There can be several network definitions in your NED files. Simulation will be able to run any of those NED files.

The syntax of a network definition is similar to that of a submodule declaration:

```
network wirelessLAN: WirelessLAN
```

```
parameters:
```

```
    numUsers=10,
    httpTraffic=true,
```

```
    ftpTraffic=true,  
    distanceFromHub=truncnormal(100,60);  
endnetwork
```

In large networks, you have two possibilities for generating NED files:

1. Generating NED files from data files
2. Building the network from C++ code

The two solutions have different advantages and disadvantages. The first is more useful in the model development phase, while the second one is better for writing larger scale, more productized simulation programs.

#### **4.7.3 Generating NED files**

Awk or Perl are good text processing tools that can read textual data file and then generate NED files and it can also be easily modified. The NED files can either be translated by ned tool in to C++ and compiled in, or loaded dynamically. It also provides database support.

#### **4.7.4 Building the network from C++ code**

Another alternative is to write C++ code which becomes part of the simulation executable. The code would read the topology data from data files or a database, and build the network directly, using dynamic module creation. The code which you need to write would be similar to the \*\_n.cc files output by nedtool.

Since writing such code is more complex than letting perl generate NED files, this method is recommended when the simulation program has to be somewhat more productized, for example when OMNeT++ and the simulation model is embedded into a larger program, e.g. a network design tool.

# **Chapter 5**

## **Experimental Analysis And Simulation Results**



Experiments are performed in OMNeT simulator 3.3. OMNeT simulator covers a large number of application, protocols, network elements and with different traffic model. Its primary application area is the simulation of communication networks, but because of its generic and flexible architecture, is successfully used in other areas like the simulation of complex IT systems, queueing networks or hardware architectures as well. OMNeT++ provides component architecture for models. Components (modules) are programmed in C++, then assembled into larger components and models using a high-level language (NED). Reusability of models comes for free. OMNeT++ has extensive GUI support, and due to its modular architecture, the simulation kernel (and models) can be embedded easily into your applications.

OMNeT++ uses NED and CPP source files for compilation of the simulation

## **5.1 Experimental Setup**

### **Network Scenarios**

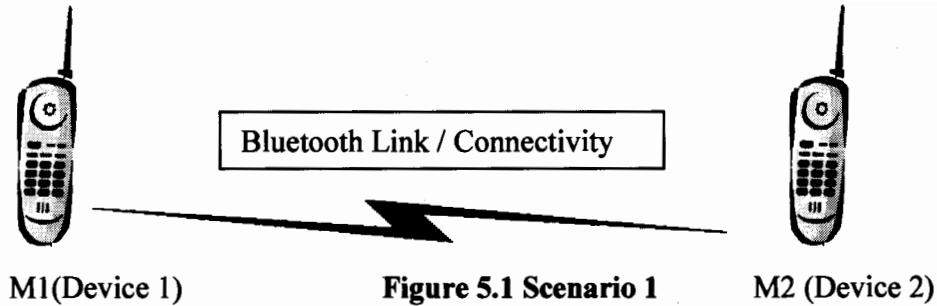
In exploring the performance of voice packets on Bluetooth SCO link, the behaviors in the two scenarios are studied.

- Device 1( master) to Device 2 (Slave) [piconet without delay]
- Device 1( master) to Device 2 (Slave) [piconet in presence of delay]

In our simulation model two mobiles are connected with each other in piconet (one is master and the other is slave) of link bandwidth 1Mbps with packet type HV3, link type SCO, protocol L2CAP, LC, LM, RF and security mode 1 with profile general audio / video. Where device1 is acting as master and device2 is acting as slave in the piconet. Device2 is identified by AM\_ADDR which is 3 bit address. Both the devices are using 0111 type code for packet in SCO link.

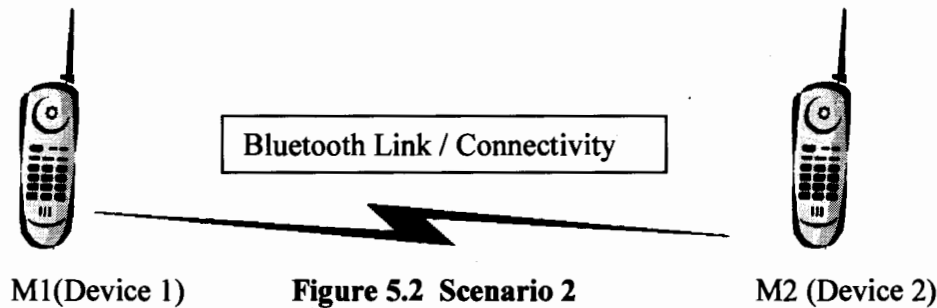
#### **(a) Scenario 1:**

In the first scenario M1 mobile is acting as sender (master) and M2 mobile as receiver (slave) in environment without delay i.e 0ms delay with 100 bits BER and data rate 1500 bps.



**(b) Scenario 2:**

In the second scenario M1 mobile is acting as sender (master) and M2 mobile as receiver (slave) in the presence of delay i.e 50, 100, 150, 250 ms delay with 100 bits BER and data rate 1500 bps.



## 5.2 Compiling the Simulation

First open the command prompt then go to the OMNeT++ directory. Now open the project directory. Then type the following command for compiling the simulation.

```
> opp_nmake make -f
```

It will use .NED nd .CPP source files after that type the command

```
> nmake -f makefile.vc
```

Then during compilation this command will create some necessary files for the simulation. It will use the following files `envir.lib` , `tkenv.lib` , `tcl84.lib`, `tk84.lib`, `sim_std.lib`, `nedxml.lib`, `libxml2.lib`, `iconv.lib`, `wsock32.lib`.



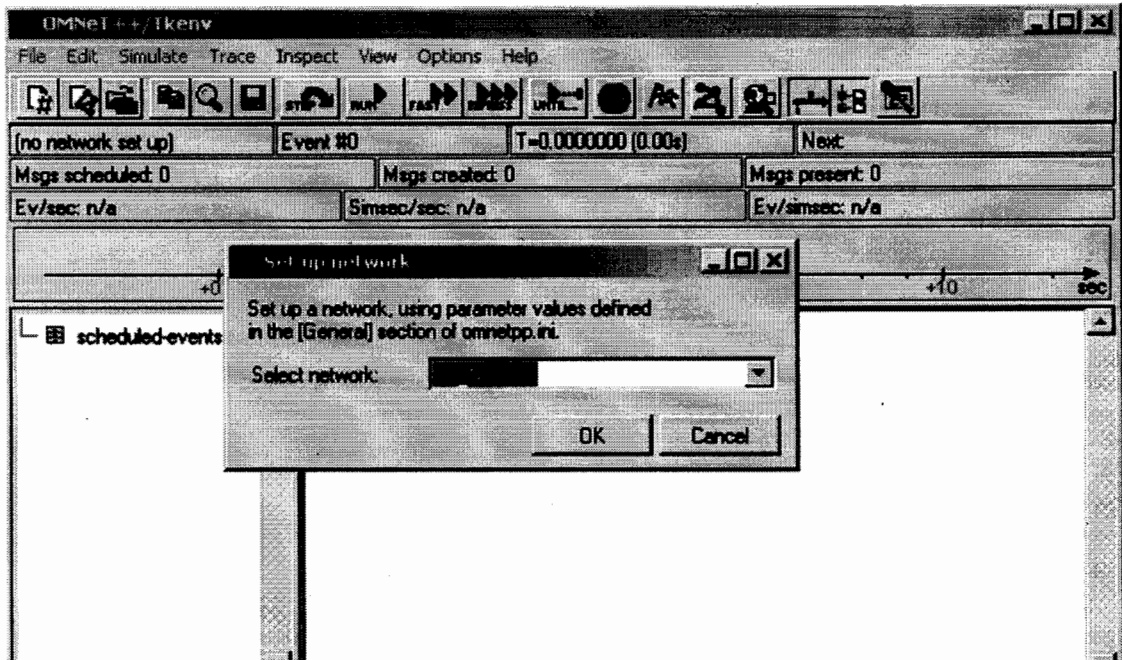


Figure 5.4 Running the simulation

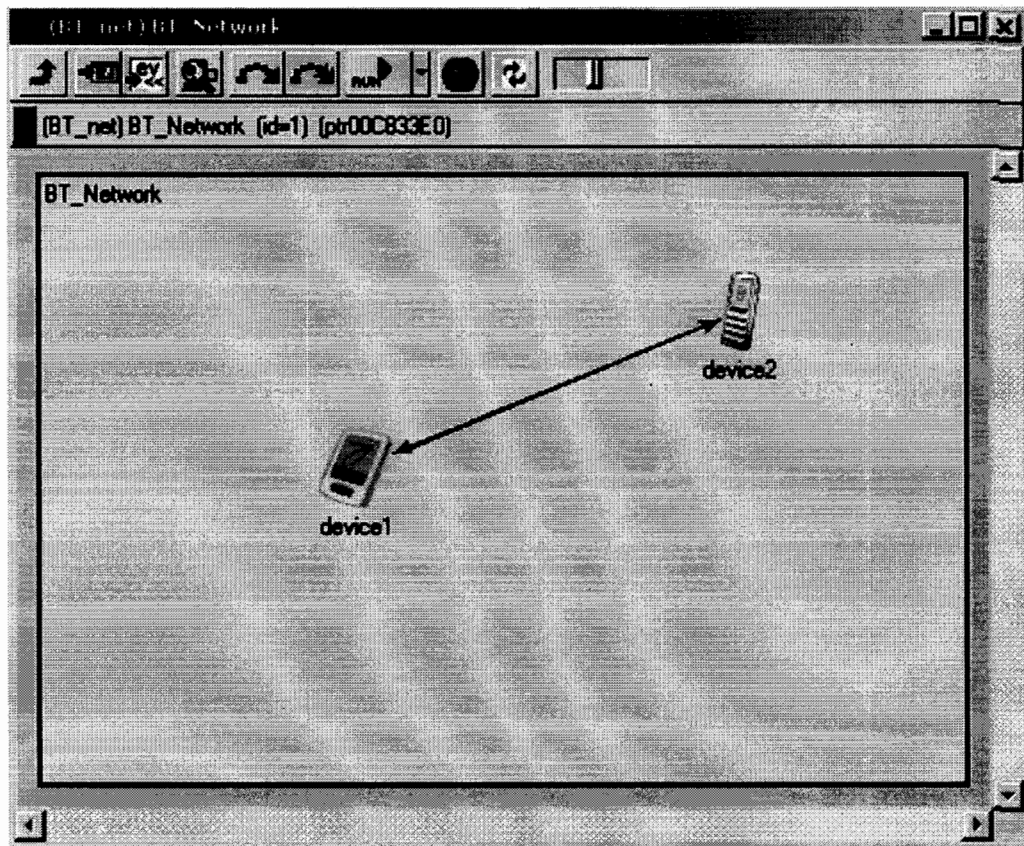
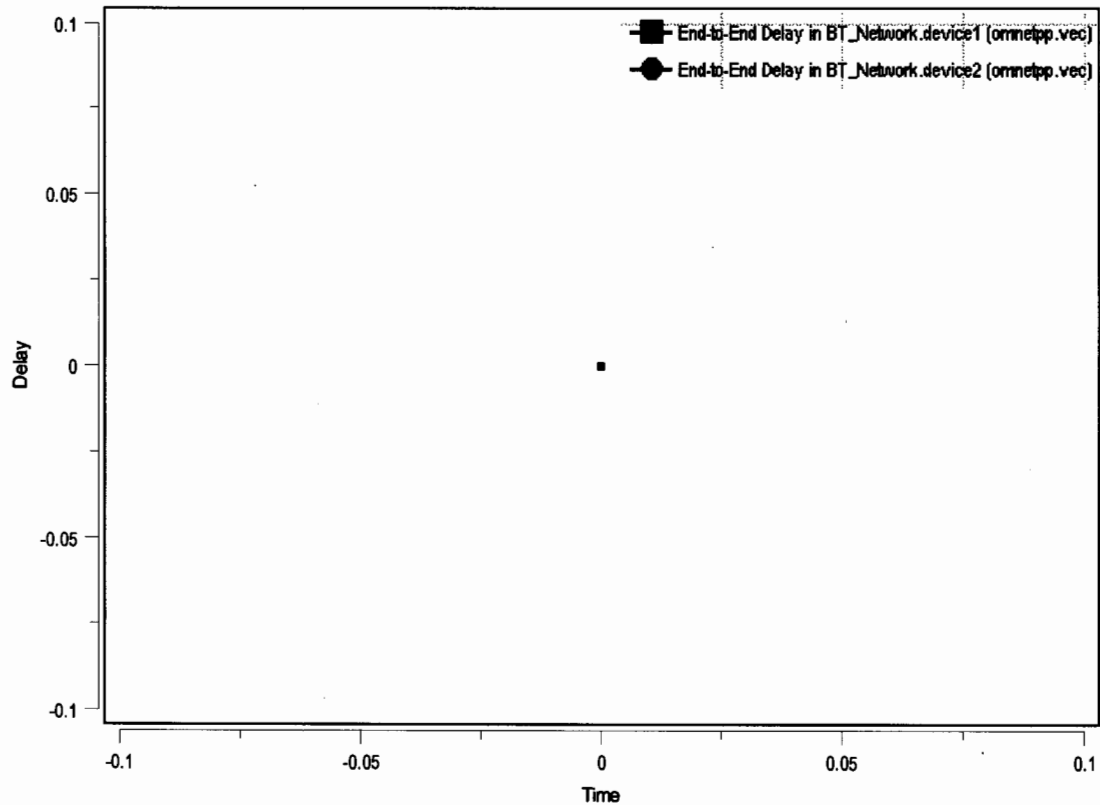


Figure 5.5 Running the simulation interface

## 5.4 Scenario 1

### Vector Graph device 1 and device 2 (with 0ms delay)



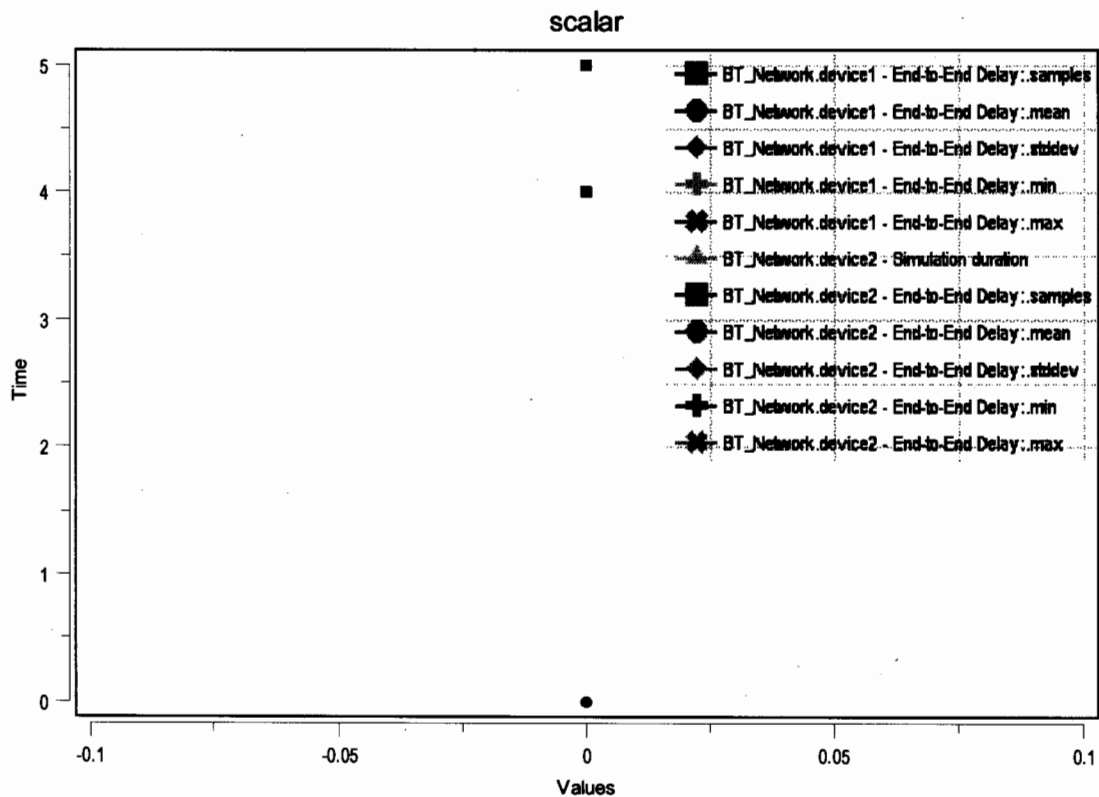
**Figure 5.6 End to End delay (device1 & 2 with 0ms delay)**

The above figure shows the end to end delay of device 1 and 2 in the form of vector graph.

The end to end delay of both the devices is zero against the different time span it is not possible in the real world but for the comparison with other scenario the test has been performed. Time is taken on x- axis and the delay is on y- axis .In this graph the values of delay is zero for both the devices (device 1 and device 2). The values for Bit Error

Rate (BER) are taken 100 bits and the data rate is 1500 bps for both the devices. Now see the performance that is maximum here in this case. The performance depends on the delay if the delay is minimum then maximum performance will be achieved. Here in this case the performances of both the devices are equal because both the devices have the same mean, standard deviation and variance.

### Scalar Graph (with 0ms delay)



**Figure 5.8 Simulation duration (with 0ms delay)**

The above figure shows the simulation duration of device 1 and device 2 in the form of scalar graph. In this graph the values (mean and standard deviation) are taken at X-axis and the time is taken on Y-axis. In this graph value of mean and variance is zero at every simulation time. This shows that at the minimum delay the values of the mean and

standard deviation is zero or minimum. The delay can be minimized by the minimum interference of other devices.

The above graph shows the standard deviation but rest of the values i.e mean variance, minimum, maximum and samples are not shown because all have the same values. In real world 0ms delay is not possible but for the comparison and evaluation purpose this test has been performed to compare the results with the other scenario.

#### 5.4.1 Delay (0, 0)

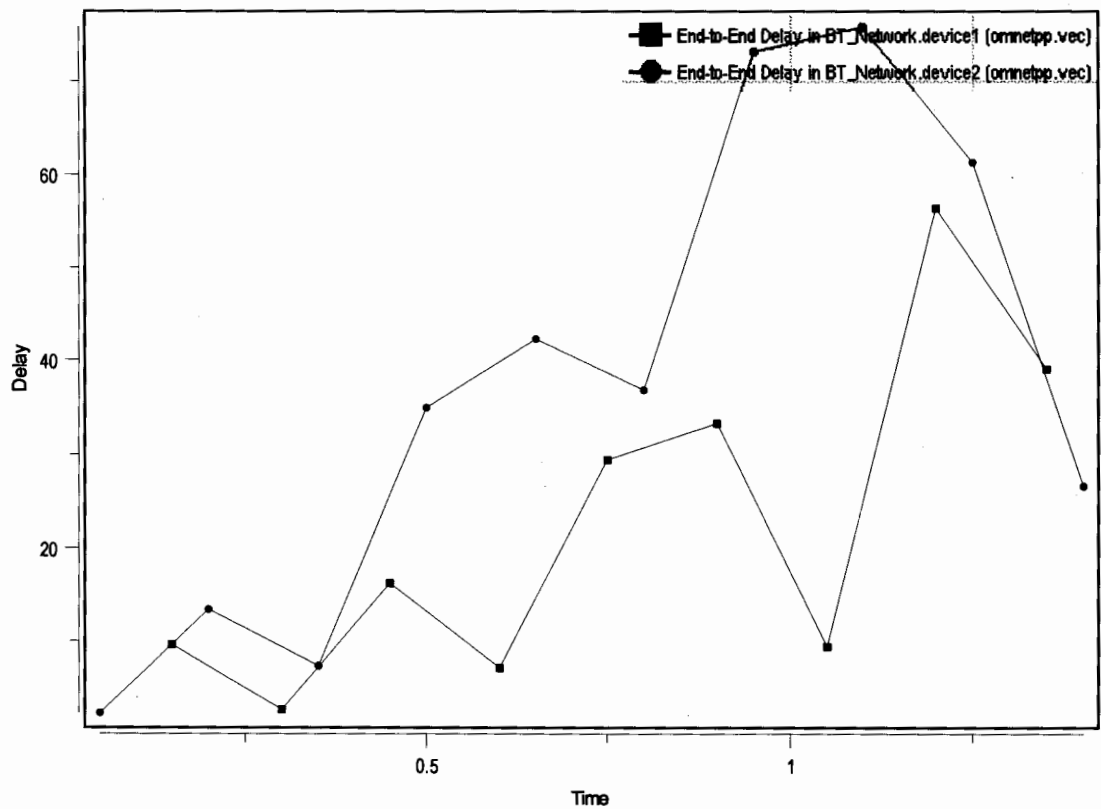
Table 5.1 shows that In the first scenario Device 1 have 0ms delay with BER 100 bits and data rate of 1500 bps the mean , standard deviation and variance is 0. Device 2 has also 0ms delay with BER 100 bits and data rate of 1500 bps the mean, standard deviation and variance is 0. For both the devices the mean, standard deviation and variance is same i.e equal to zero, that shows the equal and best performance.

Device	Delay (ms)	BER (bits)	Data rate (bps)	Mean	Std Dev	Variance
Device 1	0	100	1500	0	0	0
Device 2	0	100	1500	0	0	0

**Table 5.1 Device 1 and Device 2 delay 0ms**

## 5.5 Scenario 2

### Vector Graph device 1 and 2 (with delay 50ms, 100ms)



**Figure 5.9 End to End delay (device1 & 2 with 50,100ms delays)**

The above figure shows the end to end delay of device 1 and device 2 in the form of vector graph. In this graph the delay of device 1 is 50ms and the delay of device 2 is 100ms. The Bit Error Rate (BER) of both the devices is 100 bits and the Data Rate is also same of the both the devices that is 1500 bps. The result obtained from this data differs from one another due to change in delay of both the devices.



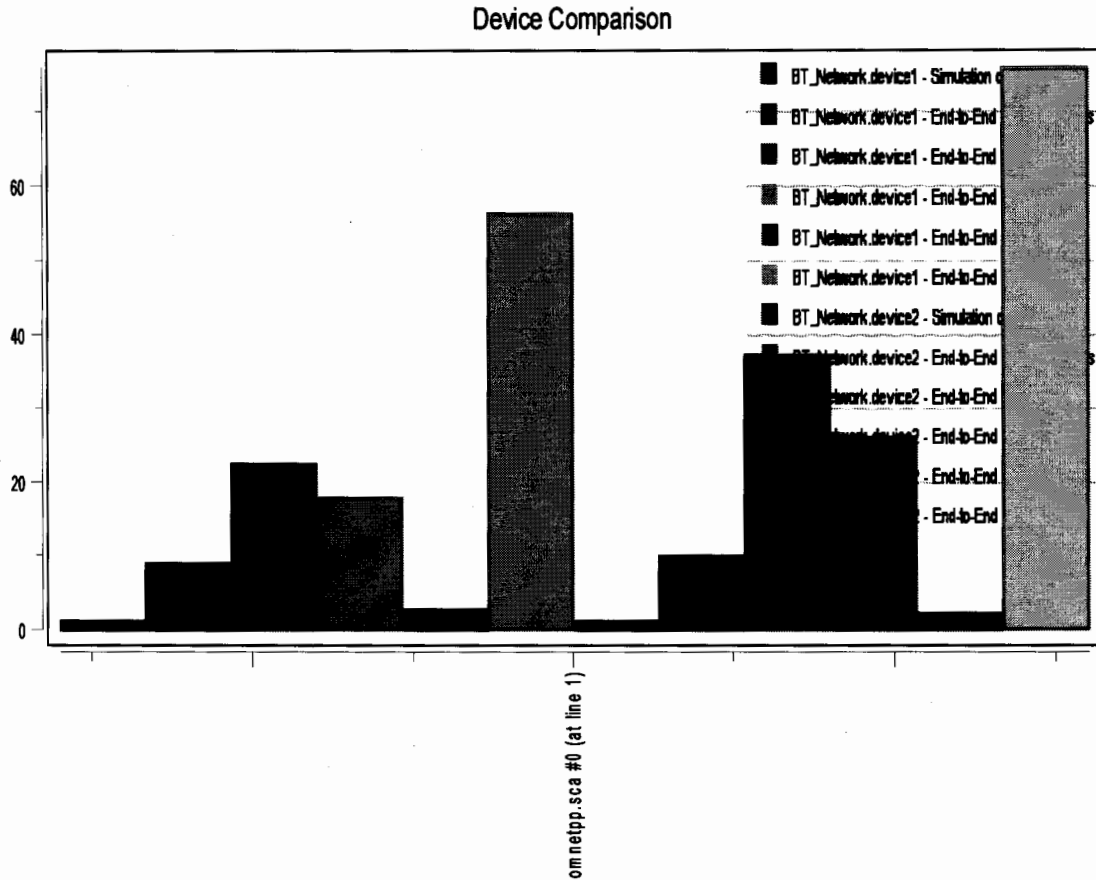
The graph shows the delay versus time. The following results are obtained from the above graph.

Device 1		Device 2	
Time	Delay (ms)	Time	Delay (ms)
0.15	9.06	0.05	2.35
0.3	2.7	0.20	13.40
0.45	16.2	0.35	7.35
0.60	7.2	0.5	35.0
0.75	29.25	0.65	42.25
0.90	33.3	0.30	36.80
1.05	9.45	0.95	73.15
1.20	56.4	1.1	75.9
1.35	39.15	1.25	11.25

**Table 5.2 Delay vs Time**

The above Table shows the different values of delay with respect to time. In this table different delays are observed at time span 0.15 the delay is 9.6ms and at 0.3 time span the delay is 2.7ms in this way all the delays are shown in the above table with respect to time. Here the time span is 0.15 it means that the delay is recorded after at every 0.15ms. In this table we see a great variation in delay. The variation in delay and zig zag curve as shown in the graph will be due to the change in frequency channels.

**Scalar Graph (with delay 50ms,100ms)**



**Figure 5.10 Simulation duration (with 50 & 100ms delay)**

The above figure shows the simulation duration of device 1 and device 2 in the form of scalar graph. This graph shows the comparison of device 1 and device 2. In this comparison mean, standard deviation and variance of device 1 is compared with the device 2 by applying the delay. The delay of device 1 is 50ms and delay of device 2 is 100ms. The Bit error rate and data rates are constant in this comparison. The mean and standard deviation of device 1 are 22.5833 and 17.9968 respectively. Similarly the delay of device 2 is 100ms, bit error rate and data rate are constant. The mean and standard deviation of device 2 are 37.405 and 26.2105 respectively. While comparing the mean and standard deviation of device 1 and device 2 we see that the mean and standard deviation depends upon the value of delay the less value of delay causes decrease in

mean and standard deviation. In the above graph the mean and standard deviation of device 1 is less than the value of mean and standard deviation of device 2.

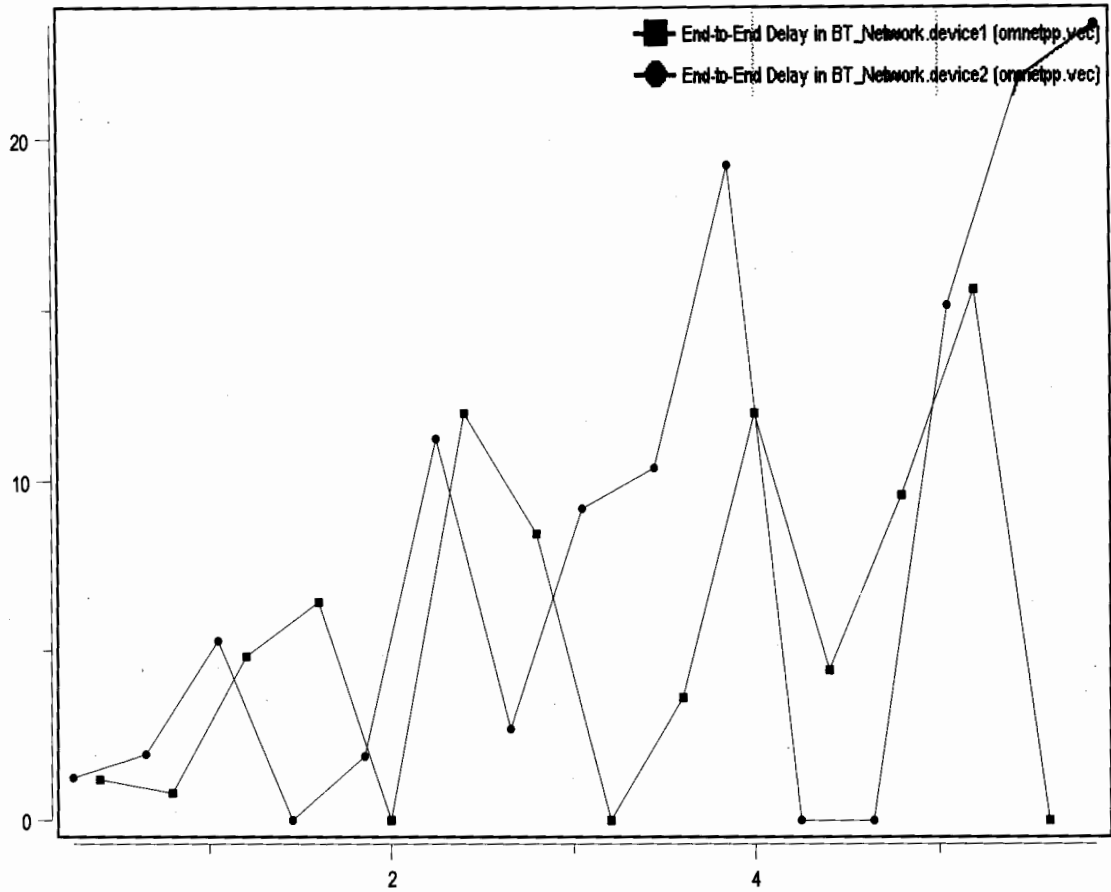
### 5.5.1 Delay (50,100)

Table 5.3 shows that the Device 1 has 50ms delay with BER 100 bits and data rate of 1500 bps the mean, standard deviation and variance is 22.5833, 17.9968 and 8.9984 respectively. .Device 2 have 100ms delay with BER 100 bits and data rate of 1500 bits the mean, standard deviation and variance is 37.405, 26.2105 and 13.10525 respectively .As device 1 have mean, standard deviation and variance is less than the device 2 so the device 1 is showing the better performance as compared to device 2 in the presence of delay introduced in both the devices.

Device	Delay (ms)	BER (bits)	Data rate (bits)	Mean	Std Dev	Variance
Device 1	50	100	1500	22.5833	17.9968	8.9984
Device 2	100	100	1500	37.405	26.2105	13.10525

**Table 5.3 Device 1 and Device 2 delay 50ms,100ms respectively**

**Vector Graph device 1 and 2 (with delay 250ms, 150ms)**



**Figure 5.11 End to End delay (device1 & 2 with 250,150 delay)**

The above figure shows the end to end delay of device 1 and device 2 in the form of vector graph.

The graph shows the delay verses time. The following results are obtained from the above graph.

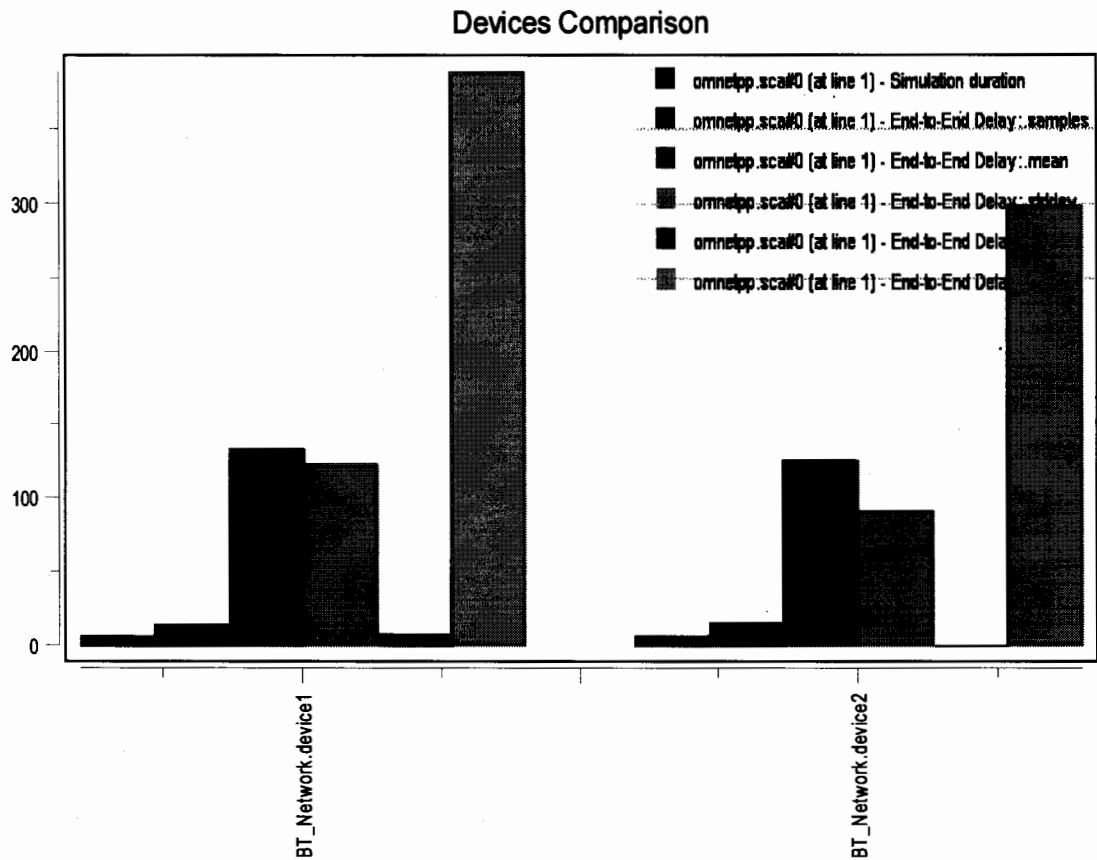
Device 1		Device 2	
Time	Delay (ms)	Time	Delay (ms)
0.4	25.6	0.25	11.75
0.8	7.8	0.65	43.55
1.2	43.2	1.05	22.05
1.6	19.2	1.45	101.5
2.0	78	1.85	120.25

Device 1		Device 2	
Time	Delay (ms)	Time	Delay (ms)
2.4	88.8	2.25	1035
2.8	25.2	2.65	204.05
3.2	150.4	3.05	210.45
3.6	104.4	6.45	169.25
4.0	156	3.85	73.15
4.4	250.8	4.25	276.25
4.8	355.2	4.65	148.8
5.2	390	5.05	116.15
5.6	190.4	5.45	299.15

**Table 5.4 Delay vs Time**

The above Table shows the different values of delay with respect to time. In this table different delays are observed at time span 0.4 the delay is 25.6ms and at 0.8 time span the delay is 7.8ms in this way all the delays are shown in the above table with respect to time. Here the time span is 0.4ms it means that the delay is recorded after at every 0.4ms. In this table we see a great variation in delay. The variation in delay is due to change in frequency channel as shown in graph.

### Scalar Graph (with delay 250ms, 150ms)



**Figure 5.12 Simulation duration (with 250 & 150ms delay)**

The above figure shows the simulation duration of device 1 and device 2 in the form of scalar graph

This graph shows the comparison of device 1 and device 2 in this comparison mean and standard deviation of device 1 is compared with the device 2 by applying the delay. The delay of device 1 is 250ms and delay of device 2 is 150ms. The Bit error rate and data rates are constant in this comparison. The mean and standard deviation of device 1 are 134.6 and 123.7807 respectively. Similarly the delay of device 2 is 150ms with bit error rate and data rate are constant. The mean and standard deviation of device 2 are

126.6833 and 92.1629 respectively. While comparing the mean and standard deviation of device 1 and device 2 we see that the mean and standard deviation depends upon the value of delay, the less value of delay causes decrease in mean and standard deviation. In this case the mean and standard deviation of device 2 is less than the value of mean and standard deviation of device 1 showing the better performance of device 2 as compared with device 1.

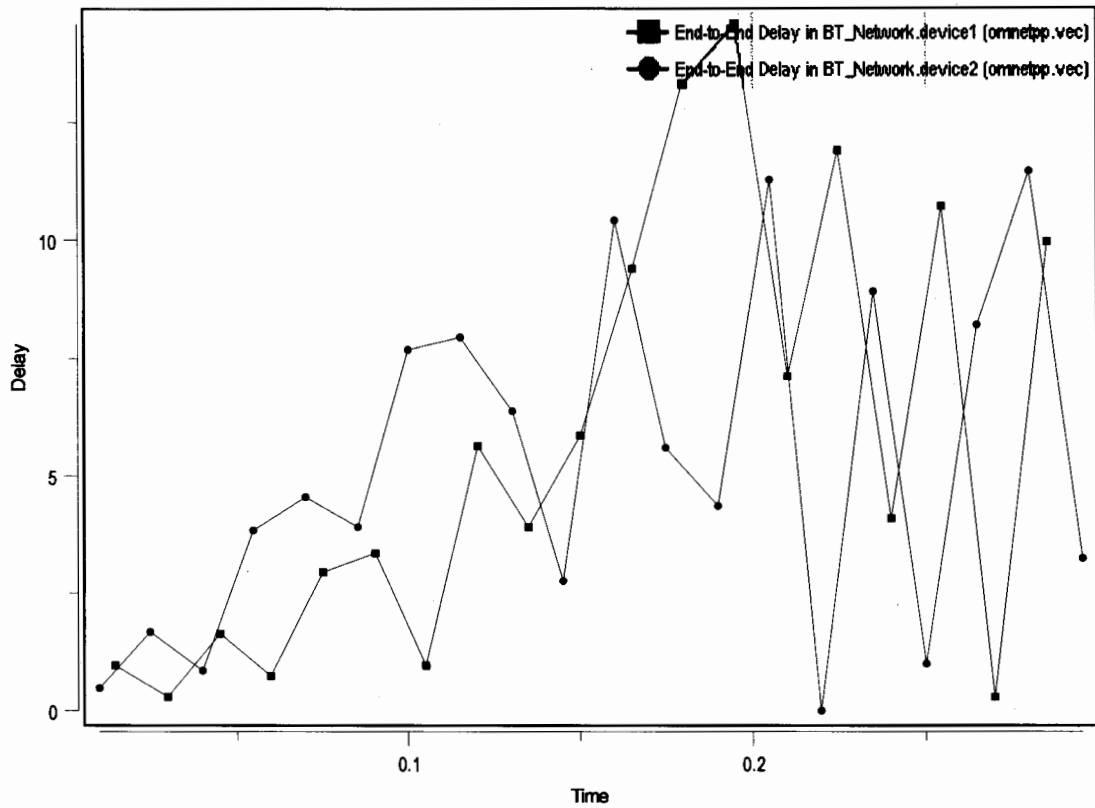
### 5.5.2 Delay (250,150)

Table 5.5 shows that the Device 1 has 250ms delay with BER 100 bits and data rate of 1500 bps the mean, standard deviation and variance is 134.6, 123.7807 and 61.89035 respectively. Device 2 have 150ms delay with BER 100 bits and data rate of 1500 bits the mean, standard deviation and variance is 126.6833, 92.1629 and 46.08145 respectively. In this case we have reduced the delay of device 2 as it has the mean, standard deviation and variance is less than the device 1 so the device 2 is showing the better performance as compared to device 1 in the presence of delay introduced in both the devices and as the variance of device 2 is better than the device 1 showing the better performance between the two.

Device	Delay (ms)	BER (bits)	Data rate (bits)	Mean	Std Dev	Variance
Device 1	250	100	1500	134.6	123.7807	61.89035
Device 2	150	100	1500	126.6833	92.1629	46.08145

**Table 5.5 Device 1 and Device 2 delay 250ms,150ms respectively**

### Vector Graph with different data rate



**Figure 5.13 End to End delay (device1 & 2 with 1500,200 data rate)**

The above figure shows the end to end delay of device 1 and device 2 in the form of vector graph.

The graph shows the delay verses time. The following results are obtained from the above graph.

Device 1		Device 2	
Time	Delay (ms)	Time	Delay (ms)
0.015	0.96	0.01	0.47
0.03	0.27	0.025	1.675
0.045	1.62	0.04	0.84
0.06	0.72	0.055	3.85
0.075	2.925	0.07	4.55

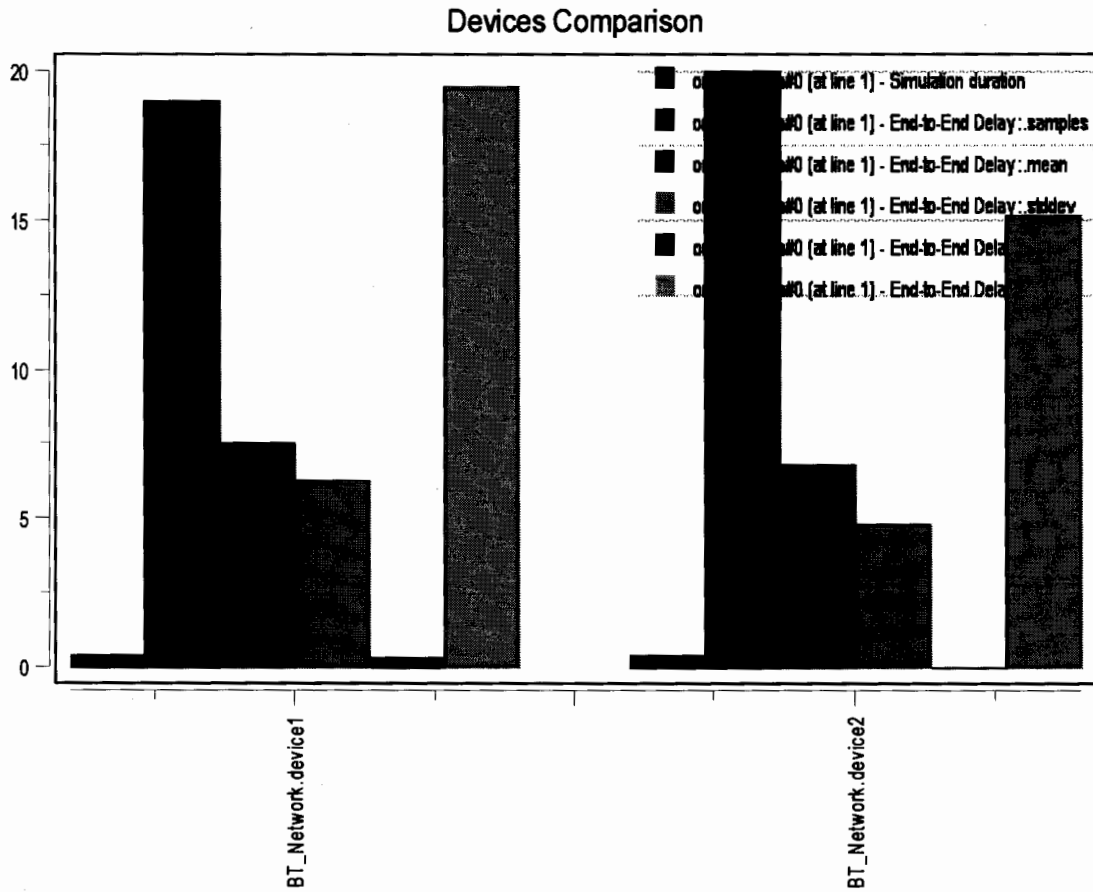


Device 1		Device 2	
Time	Delay (ms)	Time	Delay (ms)
0.09	3.3	0.085	3.91
0.105	0.945	0.1	7.7
0.120	5.64	0.115	7.935
0.135	3.915	0.3	6.37
0.15	5.85	0.145	2.755
0.16	9.405	0.15	10.4
0.18	13.32	0.16	5.6
0.195	14.625	0.175	4.37
0.21	7.14	0.19	11.275
0.225	11.925	0.25	0
0.240	4.08	0.22	8.93
0.255	10.71	0.235	1
0.270	0.27	0.25	8.215
0.285	9.975	0.265	11.48

**Table 5.6 Delay vs Time**

The above Table shows the different values of delay with respect to time. In this table different delays are observed at time span 0.015 the delay is 0.96ms and at 0.03 time span the delay is 0.27ms in this way all the delays are shown in the above table with respect to time. Here the delay is recorded after at different time span. In this table we see a great variation in delay. From the above table it shows that if we change the data rate as device 1 have 1500 bits and device 2 has 200 bits data rate we can see the performance that device 2 has better performance as compared to device 1 if we have less data rate then the performance will be better as losses are less. So data rate also effects the performance.

**Scalar Graph with different data rate**



**Figure 5.14 Simulation duration (with 1500 & 200 data rate)**

The above figure shows the simulation duration of device 1 and device 2 in the form of scalar graph.

This graph shows the comparison of device 1 and device 2 in this comparison mean and standard deviation of device 1 is compared with the device 2 by applying the different data rate. The Bit error rate is constant in this comparison. The mean and standard deviation of device 1 are 7.5526 and 6.2875 respectively. Similarly the device 2 with bit error rate is constant but data rate is 200 bits. The mean and standard deviation of device 2 are 6.833 and 4.8449 respectively. While comparing the mean and standard deviation of device 1 and device 2 we see that in this case the mean and standard deviation

depends upon the data rate, the less value of data rate causes decrease in mean and standard deviation. In this case the mean and standard deviation of device 2 is less than the value of mean and standard deviation of device 1 showing the better performance of device 2 as compared with device 1 with the different data rates.

### 5.5.3 Data rate (1500, 200)

Table 5.7 shows that the Device 1 has 50ms delay with BER 100 bits and data rate of 1500 bps the mean, standard deviation and variance is 7.5526, 6.2875 and 3.1437 respectively. Device 2 have 50ms delay with BER 100 bits and data rate of 200 bits the mean, standard deviation and variance is 6.833, 4.8449 and 2.4224 respectively. In this case both the devices have the same delay but device 2 as it has mean and standard deviation is less than the device 1 so the device 2 is showing the better performance as compared to device 1 in the presence of same delay and the same BER but with different data rates. Hence data rate will affect the performance.

Device	Delay (ms)	BER (bits)	Data rate (bits)	Mean	Std Dev	Variance
Device 1	50	100	1500	7.5526	6.2875	3.1437
Device 2	50	100	200	6.833	4.8449	2.4224

**Table 5.7 Device 1 and Device 2 data rate 1500,200 respectively**

## 5.6 Formula Used

We have used the built in statistics function of OMNeT++ to calculate the mean and standard deviation and variance.

### End to End delay

The formula used to calculate end to end delay is

eed = simulation Time() – message creation Time();

## **Chapter 6**

### **CONCLUSION**

## CONCLUSION

After the experiments of both the scenarios we have come to the conclusion by evaluating the performance of the voice packets on the SCO link that in first scenario in the absence of delay and interference the voice packets and performance will not be effected as with different time interval between the sender and receiver but in the second scenario as in the presence of delay the voice packets will be effected when delay is added in the environment, the difference between the mean , standard deviation and variance is recorded between the sender and the receiver.

In the second scenario we have less performance due to delay and overcrowded environment between the sender and receiver, as the delay is less the better the performance will be keeping the BER and data rate same. But the performance with different data rates with the same delay and BER will also reduce the performance

Overall performance in the presence of delay and interference like (in the presence of wireless LAN setup , Microwave oven and other devices operating at the same frequencies) may drop the Voice packets and reduced the performance.

As we see in the results that data rate also effects the performance in the presence of delay if lesser is the data rate the better the performance will be in the presence of delay.

As interference causing the delay effects the radio layer which is responsible for the transmitting and receiving of packets on the physical channel. Logical link layer including (logical link, transport layer) controls the packets i.e encoding and decoding

and remove the errors in the packet. To minimize interference from other devices Frequency-hopping spread spectrum (FHSS), short data packets as we have tested in our simulation and adaptive power control is used by the bluetooth device.

The delay can be caused due to the devices that operate in the same band, due to WLAN, thunderstorm, cordless phone and microwave ovens. Physical test has been performed to check what effect delay can cause during the communication. Two Bluetooth enabled mobile devices are used with java support in which super Bluetooth hack software is installed to test the communication between the devices. In the presence of microwave oven voice call has been established between the two mobile phones. Voice is interrupted due to delay caused by the frequency of microwave oven but as the device is moved away from the microwave oven the communication becomes stable.

So at the end we conclude on basis of our results that delay can effect the performance.

## BIBLIOGRAPHY

- [1] Bluetooth: A Technical Overview, ACM Students Magazine, <http://portal.acm.org>.”
- [2] Yun Wu, “SCO link sharing in Bluetooth voice access networks”, in journal of Parallel and distributing computing.
- [3] Johansson, P., Kapoor, R., Kazantzidiz, M. and Gerla, M. “*Personal Area Networks: Bluetooth or IEEE 802.11*” International Journal of Wireless Information Networks, Vol. 9.
- [4] R. Kapoor, C. Jyh-Ling, Y.Z. Lee and M. Gerla, “Bluetooth: carrying voice over ACL links,” in the proceedings of 4th International Workshop on Mobile and Wireless Communications Network .
- [5] N. GOLMIE\*, R.E. VAN DYCK, “Interference Evaluation of Bluetooth and IEEE 802.11b systems” ,in IEEE 802.11, Wireless Networks.
- [6] Leong Wai Yie, J.Homer, “Determining soft blocking in Bluetooth network,” IEEE International Conference on Industrial Technology, EEE ICIT '02.
- [7] J. Bray and C. F. Sturman,” Bluetooth - Connect Without Cables”, Prentice Hall.
- [8] J. C. Haartsen, "The Bluetooth Radio System," IEEE Personal Communications.
- [9] R. Schneiderman, "Bluetooth's Slow Down," IEEE Spectrum.
- [10] Syngress - Bluetooth Application Developers Guide.
- [11] [http://bluetooth.com/Bluetooth/Learn/Works/Core\\_System\\_Architecture.htm](http://bluetooth.com/Bluetooth/Learn/Works/Core_System_Architecture.htm)  
"Architecture of Bluetooth”.
- [12] András Varga "OMNET++ User Manual" <http://whale.hit.bme.hu/omnetpp/>

- [13] Holger Karl "OMNET++ A Tool For Simulation Programming" [http://www.tkn.tu-berlin.de/curricula/ss07/praxis\\_sim/SimPraxis-Omnet.pdf](http://www.tkn.tu-berlin.de/curricula/ss07/praxis_sim/SimPraxis-Omnet.pdf)



# **APPENDICES**

## APPENDIX A

### Source Code

#### BT\_Device.cpp source code

```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>

/**
 * Derive the BT_Device class from cSimpleModule. In the Bluetooth network,
 * both the `device1' and `device2' modules are BT_Device objects, created by
 * OMNeT++
 * at the beginning of the simulation.
 */
class BT_Device : public cSimpleModule
{
private:
    int counter;
    cOutVector endToEndDelayVec;
    cStdDev eedStats;

protected:
    // The following redefined virtual function holds the algorithm.
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    virtual void finish();
};

// The module class needs to be registered with OMNeT++
Define_Module(BT_Device);

void BT_Device::initialize()
{
    endToEndDelayVec.setName("End-to-End Delay");
    eedStats.setName("End-to-End Delay:");

    counter = 1;
    WATCH(counter);

    if (strcmp("device1", name()) == 0)
    {
        // create and send first message on gate "out". "tictocMsg" is an
        // arbitrary string which will be the name of the message object.
        cMessage *msg = new cMessage("Voice Packet");
```

```

        send(msg, "out");
    }
}

void BT_Device::handleMessage(cMessage *msg)
{
    simtime_t eed = simTime() - msg->creationTime();

    eed=eed* inrand(6);
    endToEndDelayVec.record(eed);
    eedStats.collect(eed);

    counter ++;
    if (counter==20)
    {
        // If counter is Twenty, delete messages. If you run the model,
        // find that the simulation will stop at this point with the message
        // "no more events".
        ev << name() << "'s counter reached at Maximum Limit so now
        Deleting message\n";
        delete msg;
    }
    else
    {
        if (strcmp("device1", name()) == 0)
            bubble("Going to Device 2!");
        else
            bubble("Going to Device 1!");

        ev << name() << "'s counter is " << counter << ", sending back
        message\n";
        send(msg, "out");
    }
}

void BT_Device::finish()
{
    recordScalar("Simulation duration", simTime());

    eedStats.recordScalar();
}

```

## BT\_Network NED file Source Code

```
module BT_net
  submodules:
    device1: BT_Device;
    display: "i=device/palm2,#808000;p=173,160";
    device2: BT_Device;
    display: "i=device/cellphone,#80ff80;p=371,80";
  connections:
    device1.out --> delay 10ms --> device2.in;
    device1.in <-- delay 10ms error 0 datarate 1024 <-- device2.out;
    display: "b=478,312";
endmodule
```

## BT\_Network\_n.cpp Source Code

```
#include <math.h>
#include "omnetpp.h"

// NEDC version check
#define NEDC_VERSION 0x0303
#if (NEDC_VERSION!=OMNETPP_VERSION)
# error Version mismatch! Probably this file was generated by an earlier version of
nedc: 'make clean' should help.
#endif

// Disable warnings about unused variables. For MSVC and BC only:
// GCC has no way to turn on its -Wunused option in a source file :(
#ifdef _MSC_VER
# pragma warning(disable:4101)
#endif
#ifdef __BORLANDC__
# pragma warn -waus
# pragma warn -wuse
#endif

static cModuleType * _getModuleType(const char *modname)
{
  cModuleType *modtype = findModuleType(modname);
  if (!modtype)
    throw new cRuntimeError("Module type definition %s not found (Define_Module)
missing from C++ code?", modname);
  return modtype;
}

static void _checkModuleVectorSize(int vectorsize, const char *mod)
```

```

{
    if (vectorsize<0)
        throw new cRuntimeError("Negative module vector size %s[%d]", mod,
vectorsize);
}

static void _readModuleParameters(cModule *mod)
{
    int n = mod->params();
    for (int k=0; k<n; k++)
        if (mod->par(k).isInput())
            mod->par(k).read();
}

static int _checkModuleIndex(int index, int vectorsize, const char *modname)
{
    if (index<0 || index>=vectorsize)
        throw new cRuntimeError("Submodule index %s[%d] out of range, sizeof(%s) is
%d", modname, index, modname, vectorsize);
    return index;
}

static cGate *_checkGate(cModule *mod, const char *gatename)
{
    cGate *g = mod->gate(gatename);
    if (!g)
        throw new cRuntimeError("%s has no gate named %s",mod->fullPath().c_str(),
gatename);
    return g;
}

static cGate *_checkGate(cModule *mod, const char *gatename, int gateindex)
{
    cGate *g = mod->gate(gatename, gateindex);
    if (!g)
        throw new cRuntimeError("%s has no gate %s[%d]",mod->fullPath().c_str(),
gatename, gateindex);
    return g;
}

static cGate *_getFirstUnusedParentModGate(cModule *mod, const char *gatename)
{
    int baseId = mod->findGate(gatename);
    if (baseId<0)
        throw new cRuntimeError("%s has no %s[] gate",mod->fullPath().c_str(),
gatename);
    int n = mod->gate(baseId)->size();
}

```

```

    for (int i=0; i<n; i++)
        if (!mod->gate(baseId+i)->isConnectedInside())
            return mod->gate(baseId+i);
    throw new cRuntimeError("%s[] gates are all connected, no gate left for '+'
operator",mod->fullPath().c_str(), gatename);
}

static cGate *_getFirstUnusedSubmodGate(cModule *mod, const char *gatename)
{
    int baseId = mod->findGate(gatename);
    if (baseId<0)
        throw new cRuntimeError("%s has no %s[] gate",mod->fullPath().c_str(),
gatename);
    int n = mod->gate(baseId)->size();
    for (int i=0; i<n; i++)
        if (!mod->gate(baseId+i)->isConnectedOutside())
            return mod->gate(baseId+i);
    int newBaseId = mod->setGateSize(gatename,n+1);
    return mod->gate(newBaseId+n);
}

static cFunctionType *_getFunction(const char *funcname, int argcount)
{
    cFunctionType *functype = findFunction(funcname,argcount);
    if (!functype)
        throw new cRuntimeError("Function %s with %d args not found", funcname,
argcount);
    return functype;
}

static cChannel *_createChannel(const char *channeltypename)
{
    cChannelType *channeltype = findChannelType(channeltypename);
    if (!channeltype)
        throw new cRuntimeError("Channel type %s not found", channeltypename);
    cChannel *channel = channeltype->create("channel");
    return channel;
}

static cChannel *_createNonTypedBasicChannel(double delay, double error, double
datarate)
{
    cBasicChannel *channel = new cBasicChannel("channel");
    if (delay!=0) channel->setDelay(delay);
    if (error!=0) channel->setError(error);
    if (datarate!=0) channel->setDatarate(datarate);
    return channel;
}

```

```

}

static cXMLElement *_getXMLDocument(const char *fname, const char
*pathexpr=NULL)
{
    cXMLElement *node = ev.getXMLDocument(fname, pathexpr);
    if (!node)
        throw new cRuntimeError(!pathexpr ? "xmldoc(\"%s\"): element not found" :
"xmldoc(\"%s\", \"%s\"): element not found",fname,pathexpr);
    return node;
}

```

```

ModuleInterface(BT_Device)
    // gates:
    Gate(in, GateDir_Input)
    Gate(out, GateDir_Output)
EndInterface

```

```

Register_ModuleInterface(BT_Device)

```

```

///  

///  

// Sample code:  

// class BT_Device : public cSimpleModule  

// {  

//     Module_Class_Members(BT_Device,cSimpleModule,16384)  

//     virtual void activity();  

//     // Add you own member functions here!  

// };  

//  

// Define_Module(BT_Device);  

//  

// void BT_Device::activity()  

// {  

//     // Put code for simple module activity here!  

// }  

//

```

```

ModuleInterface(BT_net)
EndInterface

```

```

Register_ModuleInterface(BT_net);

```

```

class BT_net : public cCompoundModule
{
public:
    BT_net() : cCompoundModule() {}
protected:
    virtual void doBuildInside();
}

```

```

};

Define_Module(BT_net);

void BT_net::doBuildInside()
{
    cModule *mod = this;

    // temporary variables:
    cPar tmpval;
    const char *modtypename;

    mod->setBackgroundDisplayString("b=478,312");

    // submodules:
    cModuleType *modtype = NULL;
    int submodindex;

    //
    // submodule 'device1':
    //
    int device1_size = 1;
    modtype = _getModuleType("BT_Device");
    cModule *device1_p = modtype->create("device1", mod);
    {
        cContextSwitcher __ctx(device1_p); // do the rest in this module's context

        _readModuleParameters(device1_p);
        device1_p->setDisplayString("i=device/palm2,#808000;p=173,160");
    }

    //
    // submodule 'device2':
    //
    int device2_size = 1;
    modtype = _getModuleType("BT_Device");
    cModule *device2_p = modtype->create("device2", mod);
    {
        cContextSwitcher __ctx(device2_p); // do the rest in this module's context

        _readModuleParameters(device2_p);
        device2_p->setDisplayString("i=device/cellphone,#80ff80;p=371,80");
    }

    // connections:
    //
    cGate *srcgate, *destgate;

```



```

cChannel *channel;
cPar *par;
// connection
srcgate = _checkGate(device1_p, "out");
destgate = _checkGate(device2_p, "in");
channel = _createNonTypedBasicChannel(0.05, 100, 500000);
srcgate->connectTo(destgate,channel);

// connection
srcgate = _checkGate(device2_p, "out");
destgate = _checkGate(device1_p, "in");
channel = _createNonTypedBasicChannel(0.05, 100, 200000);
srcgate->connectTo(destgate,channel);

// check all gates are connected:
mod->checkInternalConnections();

//
// this level is done -- recursively build submodules too
//
device1_p->buildInside();
device2_p->buildInside();
}

class BT_Network : public cNetworkType
{
public:
    BT_Network(const char *name) : cNetworkType(name) {}
    BT_Network(const BT_Network& n) : cNetworkType(n.name()) {operator=(n);}
    virtual void setupNetwork();
};

Define_Network(BT_Network);

void BT_Network::setupNetwork()
{
    // temporary variables:
    cPar tmpval;
    const char *modtypename;

    cModuleType *modtype;
    BT_Network_p->buildInside();}

```

## APPENDIX B

### Vector Data Generated

Device 1		Device 2	
Time	Delay (ms)	Time	Delay (ms)
0.15	9.06	0.05	2.35
0.3	2.7	0.20	13.40
0.45	16.2	0.35	7.35
0.60	7.2	0.5	35.0
0.75	29.25	0.65	42.25
0.90	33.3	0.30	36.80
1.05	9.45	0.95	73.15
1.20	56.4	1.1	75.9
1.35	39.15	1.25	11.25

Device 1		Device 2	
Time	Delay (ms)	Time	Delay (ms)
0.4	25.6	0.25	11.75
0.8	7.8	0.65	43.55
1.2	43.2	1.05	22.05
1.6	19.2	1.45	101.5
2.0	78	1.85	120.25
2.4	88.8	2.25	1035
2.8	25.2	2.65	204.05
3.2	150.4	3.05	210.45
3.6	104.4	6.45	169.25
4.0	156	3.85	73.15
4.4	250.8	4.25	276.25
4.8	355.2	4.65	148.8
5.2	390	5.05	116.15
5.6	190.4	5.45	299.15

Device 1		Device 2	
Time	Delay (ms)	Time	Delay (ms)
0.015	0.96	0.01	0.47
0.03	0.27	0.025	1.675
0.045	1.62	0.04	0.84
0.06	0.72	0.055	3.85
0.075	2.925	0.07	4.55
0.09	3.3	0.085	3.91
0.105	0.945	0.1	7.7
0.120	5.64	0.115	7.935
0.135	3.915	0.3	6.37

<b>Device 1</b>		<b>Device 2</b>	
<b>Time</b>	<b>Delay (ms)</b>	<b>Time</b>	<b>Delay (ms)</b>
0.15	5.85	0.145	2.755
0.16	9.405	0.15	10.4
0.18	13.32	0.16	5.6
0.195	14.625	0.175	4.37
0.21	7.14	0.19	11.275
0.225	11.925	0.25	0
0.240	4.08	0.22	8.93
0.255	10.71	0.235	1
0.270	0.27	0.25	8.215
0.285	9.975	0.265	11.48

## APPENDIX C

### Scalar Data Generated

Directory	File	Run#	Module	Name	Value			
C:/OMNeT++/BlueTooth/results/r-no-delay/	omnetpp.sca	0				(at	line	1)
	BT_Network.device1		Simulation duration	0.0				
C:/OMNeT++/BlueTooth/results/r-no-delay/	omnetpp.sca	0				(at	line	1)
	BT_Network.device1		End-to-End Delay:.samples	4.0				
C:/OMNeT++/BlueTooth/results/r-no-delay/	omnetpp.sca	0				(at	line	1)
	BT_Network.device1		End-to-End Delay:.mean	0.0				
C:/OMNeT++/BlueTooth/results/r-no-delay/	omnetpp.sca	0				(at	line	1)
	BT_Network.device1		End-to-End Delay:.stddev	0.0				
C:/OMNeT++/BlueTooth/results/r-no-delay/	omnetpp.sca	0				(at	line	1)
	BT_Network.device1		End-to-End Delay:.min	0.0				
C:/OMNeT++/BlueTooth/results/r-no-delay/	omnetpp.sca	0				(at	line	1)
	BT_Network.device1		End-to-End Delay:.max	0.0				
C:/OMNeT++/BlueTooth/results/r-no-delay/	omnetpp.sca	0				(at	line	1)
	BT_Network.device2		Simulation duration	0.0				
C:/OMNeT++/BlueTooth/results/r-no-delay/	omnetpp.sca	0				(at	line	1)
	BT_Network.device2		End-to-End Delay:.samples	5.0				
C:/OMNeT++/BlueTooth/results/r-no-delay/	omnetpp.sca	0				(at	line	1)
	BT_Network.device2		End-to-End Delay:.mean	0.0				
C:/OMNeT++/BlueTooth/results/r-no-delay/	omnetpp.sca	0				(at	line	1)
	BT_Network.device2		End-to-End Delay:.stddev	0.0				
C:/OMNeT++/BlueTooth/results/r-no-delay/	omnetpp.sca	0				(at	line	1)
	BT_Network.device2		End-to-End Delay:.min	0.0				
C:/OMNeT++/BlueTooth/results/r-no-delay/	omnetpp.sca	0				(at	line	1)
	BT_Network.device2		End-to-End Delay:.max	0.0				

Directory	File	Run#	Module	Name	Value			
C:/OMNeT++/BlueTooth/results/r-d-50-100/	omnetpp.sca	0				(at	line	1)
	BT_Network.device1		Simulation duration	1.4				
C:/OMNeT++/BlueTooth/results/r-d-50-100/	omnetpp.sca	0				(at	line	1)
	BT_Network.device1		End-to-End Delay:.samples	10.0				
C:/OMNeT++/BlueTooth/results/r-d-50-100/	omnetpp.sca	0				(at	line	1)
	BT_Network.device1		End-to-End Delay:.mean	22.5833333333				
C:/OMNeT++/BlueTooth/results/r-d-50-100/	omnetpp.sca	0				(at	line	1)
	BT_Network.device1		End-to-End Delay:.stddev	17.9968920928				
C:/OMNeT++/BlueTooth/results/r-d-50-100/	omnetpp.sca	0				(at	line	1)
	BT_Network.device1		End-to-End Delay:.min	2.7				
C:/OMNeT++/BlueTooth/results/r-d-50-100/	omnetpp.sca	0				(at	line	1)
	BT_Network.device1		End-to-End Delay:.max	56.4				

```

C:/OMNeT++/BlueTooth/results/r-d-50-100/      omnetpp.sca  0  (at  line  1)
  BT_Network.device2 Simulation duration  1.4
C:/OMNeT++/BlueTooth/results/r-d-50-100/      omnetpp.sca  0  (at  line  1)
  BT_Network.device2 End-to-End Delay:.samples  10.0
C:/OMNeT++/BlueTooth/results/r-d-50-100/      omnetpp.sca  0  (at  line  1)
  BT_Network.device2 End-to-End Delay:.mean   37.405
C:/OMNeT++/BlueTooth/results/r-d-50-100/      omnetpp.sca  0  (at  line  1)
  BT_Network.device2 End-to-End Delay:.stddev  26.2105200466
C:/OMNeT++/BlueTooth/results/r-d-50-100/      omnetpp.sca  0  (at  line  1)
  BT_Network.device2 End-to-End Delay:.min    2.35
C:/OMNeT++/BlueTooth/results/r-d-50-100/      omnetpp.sca  0  (at  line  1)
  BT_Network.device2 End-to-End Delay:.max    75.9

```

Directory	File	Run#	Module	Name	Value
C:/OMNeT++/BlueTooth/results/r-d-250-150/	omnetpp.sca	0	(at line 1)	BT_Network.device1 Simulation duration	5.85
C:/OMNeT++/BlueTooth/results/r-d-250-150/	omnetpp.sca	0	(at line 1)	BT_Network.device1 End-to-End Delay:.samples	15.0
C:/OMNeT++/BlueTooth/results/r-d-250-150/	omnetpp.sca	0	(at line 1)	BT_Network.device1 End-to-End Delay:.mean	134.6
C:/OMNeT++/BlueTooth/results/r-d-250-150/	omnetpp.sca	0	(at line 1)	BT_Network.device1 End-to-End Delay:.stddev	123.780724236
C:/OMNeT++/BlueTooth/results/r-d-250-150/	omnetpp.sca	0	(at line 1)	BT_Network.device1 End-to-End Delay:.min	7.2
C:/OMNeT++/BlueTooth/results/r-d-250-150/	omnetpp.sca	0	(at line 1)	BT_Network.device1 End-to-End Delay:.max	390.0
C:/OMNeT++/BlueTooth/results/r-d-250-150/	omnetpp.sca	0	(at line 1)	BT_Network.device2 Simulation duration	5.85
C:/OMNeT++/BlueTooth/results/r-d-250-150/	omnetpp.sca	0	(at line 1)	BT_Network.device2 End-to-End Delay:.samples	15.0
C:/OMNeT++/BlueTooth/results/r-d-250-150/	omnetpp.sca	0	(at line 1)	BT_Network.device2 End-to-End Delay:.mean	126.683333333
C:/OMNeT++/BlueTooth/results/r-d-250-150/	omnetpp.sca	0	(at line 1)	BT_Network.device2 End-to-End Delay:.stddev	92.1629935546
C:/OMNeT++/BlueTooth/results/r-d-250-150/	omnetpp.sca	0	(at line 1)	BT_Network.device2 End-to-End Delay:.min	0.0
C:/OMNeT++/BlueTooth/results/r-d-250-150/	omnetpp.sca	0	(at line 1)	BT_Network.device2 End-to-End Delay:.max	299.75

Directory	File	Run#	Module	Name	Value
C:/OMNeT++/BlueTooth/	omnetpp.sca	0	(at line 1)	BT_Network.device1 Simulation duration	0.39
C:/OMNeT++/BlueTooth/	omnetpp.sca	0	(at line 1)	BT_Network.device1 End-to-End Delay:.samples	20.0
C:/OMNeT++/BlueTooth/	omnetpp.sca	0	(at line 1)	BT_Network.device1 End-to-End Delay:.mean	7.55263157895

C:/OMNeT++/BlueTooth/ omnetpp.sca 0 (at line 1) BT\_Network.device1 End-  
 to-End Delay:.stddev 6.28752631283  
 C:/OMNeT++/BlueTooth/ omnetpp.sca 0 (at line 1) BT\_Network.device1 End-  
 to-End Delay:.min 0.36  
 C:/OMNeT++/BlueTooth/ omnetpp.sca 0 (at line 1) BT\_Network.device1 End-  
 to-End Delay:.max 19.5  
 C:/OMNeT++/BlueTooth/ omnetpp.sca 0 (at line 1) BT\_Network.device2  
 Simulation duration 0.39  
 C:/OMNeT++/BlueTooth/ omnetpp.sca 0 (at line 1) BT\_Network.device2 End-  
 to-End Delay:.samples 20.0  
 C:/OMNeT++/BlueTooth/ omnetpp.sca 0 (at line 1) BT\_Network.device2 End-  
 to-End Delay:.mean 6.833  
 C:/OMNeT++/BlueTooth/ omnetpp.sca 0 (at line 1) BT\_Network.device2 End-  
 to-End Delay:.stddev 4.84490410529  
 C:/OMNeT++/BlueTooth/ omnetpp.sca 0 (at line 1) BT\_Network.device2 End-  
 to-End Delay:.min 0.0  
 C:/OMNeT++/BlueTooth/ omnetpp.sca 0 (at line 1) BT\_Network.device2 End-  
 to-End Delay:.max 15.17

