Bluetooth Information Exchange Network

Xiaoning (Linda) Liu

A thesis submitted to AUT University
In partial fulfilment of the requirements for the degree of
Master of Engineering (ME)

October 2008

School of Engineering

Primary Supervisor: Prof. Adnan Al-Anbuky

Acknowledgement

First and foremost, I would like to thank my supervisor, Prof. Adnan Al-Anbuky, for his constant guidance, valued advice and continuous support during the time of my thesis study. He has been highly available and highly supportive throughout the whole process of writing this thesis.

I also want to thank my family for the endless love, invaluable support and encouragement through the years.

Many thanks go out to Dr Ching Lin for his helpful advice, understanding and encouragement when he was working as my co-supervisor during the first part of this study.

I am grateful for the research environment and the substantial resources provided by the Engineering school of AUT University that enabled me to gain extensive practical experience through the experimental work performed.

My thanks also go out to Dr Guy Littlefair and Dr. John Collins for their help during the hard time of this study.

Abstract

Bluetooth is a low cost and low power wireless technology for connecting portable and / or fixed Bluetooth enabled devices to form short-range wireless ad hoc personal area networks (PANs). As the Bluetooth specification does not specify a protocol to form ad hoc Bluetooth networks, a method for forming an efficient Bluetooth network under a practical networking scenario is still an open research problem.

This thesis introduces an approach to implement an indoor ad hoc Bluetooth wireless network, Bluetooth information exchange network (BIEN). This network formation is based on Bluetooth and Java technologies. A set of Bluetooth enabled devices configured with the BIEN software application are able to spontaneously establish a dynamic multi-hop wireless network using Bluetooth technology without the need of formal network infrastructure, centralized administration, fixed routers or access points.

In this study, the performance evaluation focuses on the relation between network capacity and topology by testing end-to-end performance in terms of throughput and the latency of communication links with various parameters, including the hop number between nodes and the number of slaves in piconets. The evaluation results show that the throughput reduces with the increased length of a path, and with an increase in the number of slaves in a piconet in the network. The latency also increases with path length, and with the number of slaves in a piconet in the different experimental BIENs, whether if there is traffic or not in the networks. Experimental results have further confirmed the necessity to minimize the number of bridge nodes in the Bluetooth networks due to their traffic bottleneck effect.

This work is an attempt at implementation of a distributed multi-hop scatternet with an integrated routing protocol in the practical environments, while most of the literature focuses on covering the modelling of it. It intends to demonstrate how Bluetooth technology with Java technology can be used to design, develop and deploy ad hoc wireless networks with the commercial Bluetooth devices, and examine how well Bluetooth technology supports ad hoc multi-hop wireless network technology.

Table of contents

List of figures and tables

Statement of originality

‘I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning, except where due acknowledgment is made in the acknowledgments.’

______________________ (signed)

_____________________ (date)

Abbreviation

BIEN	Bluetooth information exchange network
FHSS	Frequency–hopping spread spectrum
GUI	Graphic user interface
HCI	Host Controller Interface
RFCOMM	Radio Frequency Communication
ISM	Industrial-Scientific-Medical
J2ME	Java 2 Platform Micro Edition
J2EE	Java 2 Platform Enterprise Edition
J2SE	Java 2 Platform Standard Edition
JVM	Java virtual machine
L2CAP	Logical Link Control and Adaptation Protocol
PC	Personal computer
PDA	Personal digital assistant
PAN	Personal area network
QoS	Quality of Service
RFCOMM	Radio Frequency Communication
RRT latency	Round trip time latency
SIG	Bluetooth Special Interest Group
SPP	Serial Port Profile
TDD protocol	Time Division Duplex protocol
TDM protocol	Time Division Multiplexing protocol

1 Introduction

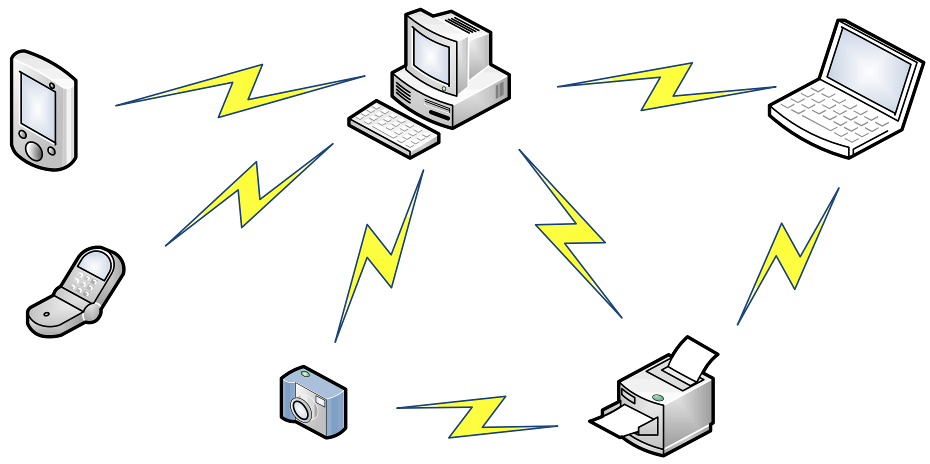
This chapter introduces Bluetooth technology, Bluetooth networking architecture and the associated metrics for evaluating Bluetooth networks. The study objectives of the thesis are then described. Finally, the structure of the thesis is outlined.

1.1 Bluetooth technology

1.1.1 Overview

Bluetooth is a short-range wireless communication technology that allows Bluetooth enabled portable and / or fixed devices to connect and exchange information with each other without any infrastructure, such as cables, connectors, infrared, and other connection media, being required [1]. It uses the slotted Time Division Duplex (TDD) protocol and fast frequency–hopping spread spectrum (FHSS) technology on the unlicensed 2.4GHz Industrial-Scientific-Medical (ISM) band for communication. This facilitates better security in terms of information interception than when a fixed frequency is used.

The major features of Bluetooth technology are its wireless connection, robustness, low power requirements, and low cost. The applications include peripheral connectivity, transferring files, forming ad hoc networks, the transferring of mobile payments, and so on. Since Bluetooth is intended as a replacement for cable, infrared, and other connection media, it provides a way to connect Bluetooth enabled portable and fixed devices; such as personal digital assistants (PDAs), cell-phones, personal computers (PCs), laptops, digital cameras and printers; in order to form wireless networks spontaneously for information exchange between such devices without requiring fixed networking infrastructure (Shown in Figure 1.1).



.

Figure 1.1 A Bluetooth ad hoc network where Bluetooth devices spontaneously connect for communication with each other

This technology was initially conceived by the telecommunications equipment supplier Ericsson in 1994 and was then developed and promoted by the Bluetooth Special Interest Group (SIG). This was established by Ericsson, IBM, Intel, Nokia and Toshiba in 1998, and to date has more than 2000 members, including all telecommunications manufacturers. The objectives of this SIG are to make, evolve, and enhance Bluetooth into a royalty-free, open, and common specification as a global standard for short-range wireless communication, with the objective that any Bluetooth devices conforming to the Bluetooth specification [2] can communicate to each other worldwide.

The Bluetooth specification consists of two sections: The Core Specification (Volume I); and the Profile Definitions (Volume II). Version 2.1 + Enhanced Data Rate (EDR)[3] is the current Core specification, updated as at July, 2007.

According to the Bluetooth specification, Bluetooth technology comprises three parts; being radio technology, the Bluetooth protocol stack, and interoperability profiles. The first two parts are described in the Core specification and the interoperability profiles are described in the Profile Definitions.

1.1.2 Radio technology

Bluetooth radio technology operates on the unlicensed 2.4 GHz industrial-Scientific-Medical (ISM) band, which ranges from 2.402 to 2.480 GHz. This frequency band is divided into 79 channels, with a channel spacing of 1 MHz. The frequency of all the channels can be represented by the following formula:

$$f_k = (2{,}402 + k*1)\ MHz \qquad \textbf{(Equation 1.1)}$$

where k is the channel number within the range of 0 to 78

During a connection, radio transceivers employ the FHSS approach to hop from one channel to another at a rate of 1600 hops/sec. That is the length of the time slot that it stays in every channel is 625µs and each slot matches a hop frequency [4]. This allows for the minimisation of interference from other devices that work in the same frequency band and reduces the possibility that data transmission may be intercepted. In the connection, once a packet is sent on a channel, the two devices retune their frequencies to send the next packet on a different channel (i.e., frequency hop). If the transmission meets an interruption because of interference, the devices simply resend the packet on a different channel.

The Bluetooth specification defines two types of physical links, Synchronous Connection Oriented (SCO) links, and Asynchronous ConnectionLess (ACL) links. The SCO link is used for audio transmission while the ACL link is used for data communication. The SCO link provides a Quality of Service (QoS) guarantee, while the ACL link provides error-free transmission of data through which a lost, or erroneous, packet will be re-transmitted, but without a QoS guarantee.

It supports the full-duplex communication, with slotted Time Division Multiplexing (TDM) protocol for processing of both data and voice transmissions. The signal transmission is Omni-directional, which means that there is no blocking from walls, briefcases, and so on.

The radio wavelength range depends on the device class. The most mobile devices are commonly class 2 devices with a radio range of 10 metres, while the mainly industrial-used devices are class 1 devices with a range of up to 100 metres [5]. Two Bluetooth devices that are within the radio range of each other must tune their transceivers to the same RF frequency at the same time in order to use a shared physical channel to communicate [4].

The transmission data rate varies with the Bluetooth specification version that the device supports. A device that supports Version 1.2 has a data rate of 1 Mbps, with up to 3 Mbps if a device supports Version 2.0 + EDR[5]. Therefore, Bluetooth deals with connecting small devices for communication at slower speeds (currently at a maximum rate of 3Mbps) within a short range (up to a maximum of 100 metres), but is poor for transmitting large files between devices and cannot be used in long-range communication [6].

1.1.3 Bluetooth protocols stack

The core of the Bluetooth specification is the Bluetooth protocol stack that controls a Bluetooth device. This protocol stack provides well-defined layers of functionality so that the Bluetooth specification guarantees interoperability of Bluetooth enabled devices and promotes acceptance of Bluetooth technology [7].

The architecture of the Bluetooth protocol stack is shown in Figure 1.2. This stack consists of two components; the Bluetooth Host, and the Bluetooth Controller. These two components are separated, and connected by the Host Controller Interface (HCI). All protocol layers above the HCI implemented in software belong to the Bluetooth Host, while those protocol layers below the HCI are firmware and hardware called Bluetooth Controller [5].

The functions of each layer are described as follows:

- The HCI layer is the interface between the Bluetooth host and the Bluetooth controller;
- The radio layer is the physical wireless channel for transmitting and receiving packets;
- The Baseband Link Controller (LC) layer controls and sends data packets over the radio link;
- The Link Manager Protocol (LMP) is used to set up connections between devices, examine service quality, and authenticate and secure services;
- The Logical Link Control and Adaptation Protocol (L2CAP) are responsible for collecting application data and changing it to the Bluetooth format;
- RFCOMM is the abbreviation for *Radio Frequency Communication,* also known as the virtual serial port protocol, which allows a Bluetooth device to simulate a serial port;
- OBEX is the Object Exchange Protocol that is used to transfer data as an object;
- SDP is the Service Discovery Protocol used for discovery services on other Bluetooth devices.

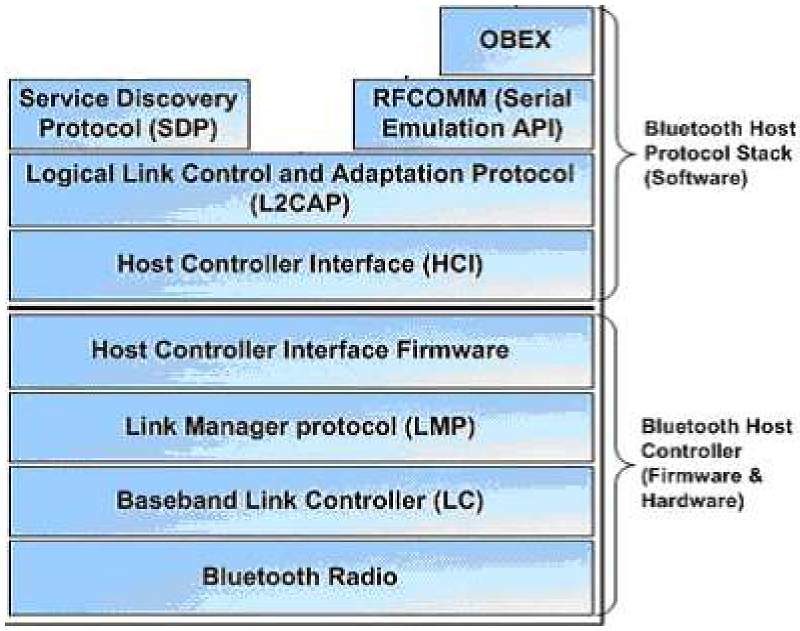

OBEX
Service Discovery Protocol (SDP)
RFCOMM (Serial Emulation API)
Logical Link Control and Adaptation Protocol (L2CAP)
Host Controller Interface (HCI)
Host Controller Interface Firmware
Link Manager protocol (LMP)
Baseband Link Controller (LC)
Bluetooth Radio
Bluetooth Host Protocol Stack (Software)
Bluetooth Host Controller (Firmware & Hardware)

Figure 1.2 Bluetooth Protocol Stack

1.1.4 Bluetooth profiles

Bluetooth profiles depict cross-platform interoperability and consistency requirements necessary for various Bluetooth devices from different manufacturers and vendors to interoperate [8]. They define the roles and capabilities for applications of the Bluetooth devices. The Bluetooth profiles are arranged into a hierarchy of groups. Figure 1.3 shows the overall relation of the Bluetooth profiles.

Bluetooth devices that have a Bluetooth stack, and also conform to a particular profile, can communicate with each other. The four basic profiles are the Generic Access Profile (GAP), the Serial Port Profile (SPP), the Service Discovery Application Profile (SDAP), and the Generic Object Exchange Profile (GOEP). These four profiles interact directly with the Bluetooth protocol stack layers, with the other profiles are built on top of these basic profiles. Currently, all Bluetooth devices in the market must support one, or more of these profiles. More details about the profiles can be found in the Bluetooth Specification [2].

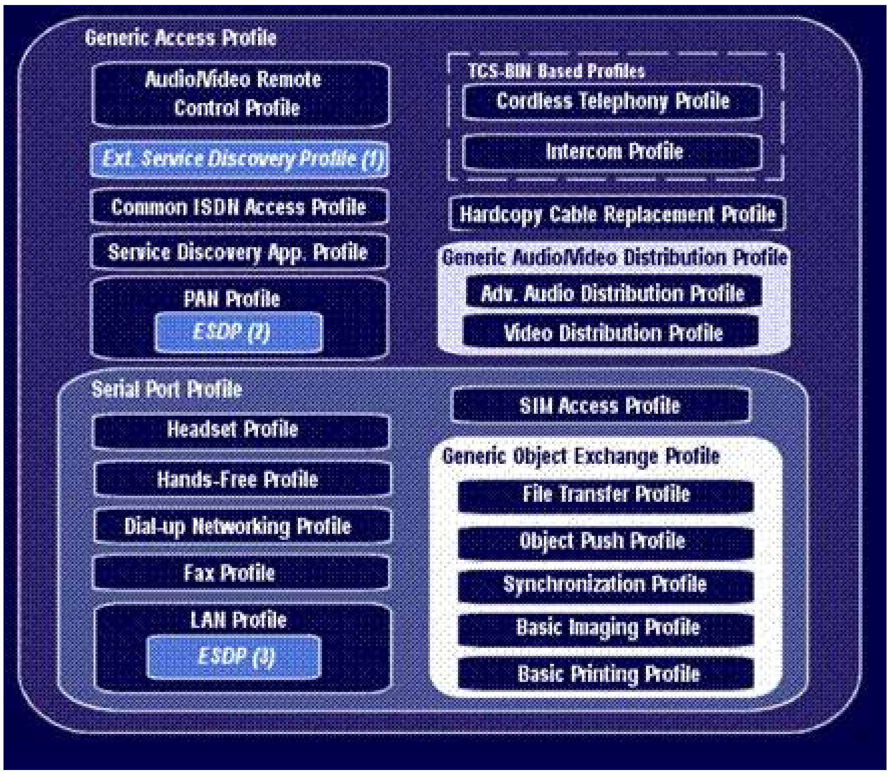

Generic Access Profile
AudioVideo Remote Control Profile
Ext. Service Discovery Profile (1)
Common ISDN Access Profile
Service Discovery App. Profile
PAN Profile
ESDP (2)
TCS-BIN Based Profiles
Cordless Telephony Profile
Intercom Profile
Hardcopy Cable Replacement Profile
Generic AudioVideo Distribution Profile
Adv. Audio Distribution Profile
Video Distribution Profile
Serial Port Profile
Headset Profile
Hands-Free Profile
Dial-up Networking Profile
Fax Profile
LAN Profile
ESDP (3)
SIM Access Profile
Generic Object Exchange Profile
File Transfer Profile
Object Push Profile
Synchronization Profile
Basic Imaging Profile
Basic Printing Profile

Figure 1.3 Bluetooth Profiles

1.2 Bluetooth network technology

An ad hoc Bluetooth network is a personal area network (PAN) that is a dynamic network spontaneously established by a set of Bluetooth devices without the need for a formal infrastructure, or centralised administration. It is limited in temporal and spatial size and is drastically different from the existing wired networks because it is wireless, has rapid deployment, is self-configuring and self-starting, has no administrator, each node does not need to be within the communication range of all the other nodes, and the topology formed is fairly dynamic, as the nodes can join and leave the network in an arbitrary way. These features make it suitable for use when communicating in emergency situations, such as during search and rescue operations in natural disasters, or military conflicts. Appropriate applications also include conferences and meetings in which people need to quickly exchange information, as well as data acquisition operations in harsh environments.

1.2.1 Bluetooth network formation

The Bluetooth specification [3] defines a uniform structure for various Bluetooth enabled devices to connect together in order to form an ad-hoc personal area network (PAN) for the purposes of communication. The network formations are based on the basic building block of the piconet. A group of piconets can connect in formulating a scatternet. This section will shed light on this basic network structure.

1.2.1.1 Connection and piconets

Bluetooth has a client-server architecture, in which the client initiates the connection and the server receives the connection.

When two Bluetooth devices want to connect together, they must be within each other's radio range. In a connection between two devices, one of the devices must play a role of master and the other one performs the role of slave. Any Bluetooth devices can function as a master and/or a slave. Before setting up a connection, a Bluetooth device has to execute an inquiry process to discover other devices within its range by entering an *INQUIRY* state if it wants to connect with other devices. At the same time, the other Bluetooth devices that wait for connections make themselves discoverable by entering an *INQUIRY SCAN* state, in order to listen and respond to the inquiries. If the inquiry process is successful, the device that has performed the *INQUIRY* has obtained a list of nearby compatible devices. The device that is going to be the master in a connection enters into a *PAGE* state to make a connection request to one of the discovered devices. At the same time, one of discovered devices (seeking to be a slave) enters into the *PAGE SCAN* state to wait for, and accept a connection from a master. In this way, a connection can be set up between the devices. Such a simple one-hop network, made up of a master and slave connected together, is named a piconet (See Figure 1.4 (a)).

The piconet is the basic unit of a Bluetooth network, being a one-hop network made up of a master and a number of slaves. There can be, at most, seven active slaves in a piconet. Links only exist between the master and each slave in a piconet, with no direct links between the various slaves as depicted in Figure 1.4(b). The master is the device that starts the communication and the slave is the device that responds to the master. Any Bluetooth device can act as a master, or as a slave. A Bluetooth device can join more than one piconet at the same time, but it can only be the master in one piconet at any one time. A piconet can have more than seven slaves, but only up to seven of them can actively communicate with the master at any one time. The other slaves have to be parked and share the same physical channel, but are unable to transmit data. The master is in charge of parking and activating slaves.

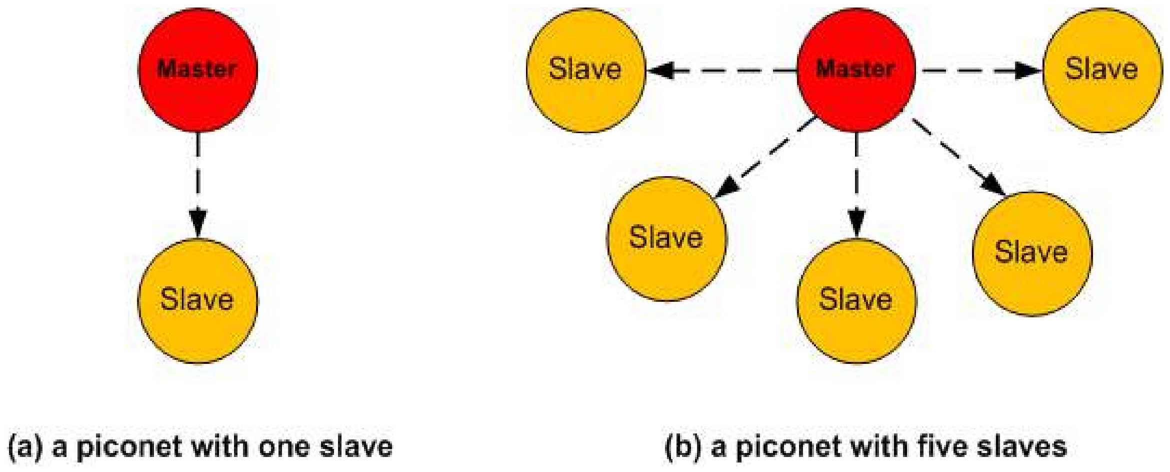


Figure 1.4 Piconet structures

Each Bluetooth device has a unique device address and clock. The frequency hopping sequence of each piconet is distinctive and determined by the master's device address and clock. When a device joins a piconet as a slave, it has to synchronise itself with the master clock and hopping sequence [4]. The TDD schema is used in communication between a master and a slave in a piconet. The communication in a piconet that has one master and one single slave is point-to-point, while point-to-multiple communication occurs if the piconet has more than one slave. There is no direct communication between slaves inside the piconet and all communications from one slave to another slave must go through the master.

All of the active slaves in a piconet share the physical FHSS channel of 1Mbps bandwidth for communication, using the Time Division Multiplexing (TDM) protocol to divide the total bandwidth into time slots between slaves, scheduled by the master of the piconet. The characteristics of this shared physical channel; such as the FHSS frequency, the frequency hopping sequence, and so on, are derived from the master's clock and device address, so the master controls the communication in the piconet completely [3]. It assigns the time slots to slaves for packet transmission in such a way that the master transmits packets to a slave in the even-numbered time slots and transmission in the reverse direction occurs in odd-numbered time slots (Shown in Figure 1.5). The master also determines the polling order of the active slaves, in which a slave can communicate with it at each time slot. The slave that received a packet from the master in the previous time slot can send a packet to the master in the next time slot. The packet transmission is conducted on a different frequency, as the result of the FFSS approach being used, and the slave has to synchronize itself with the frequency hop sequence of the master when it exchanges data with the master, as only two devices with the same frequency hop sequence can communicate to each other.

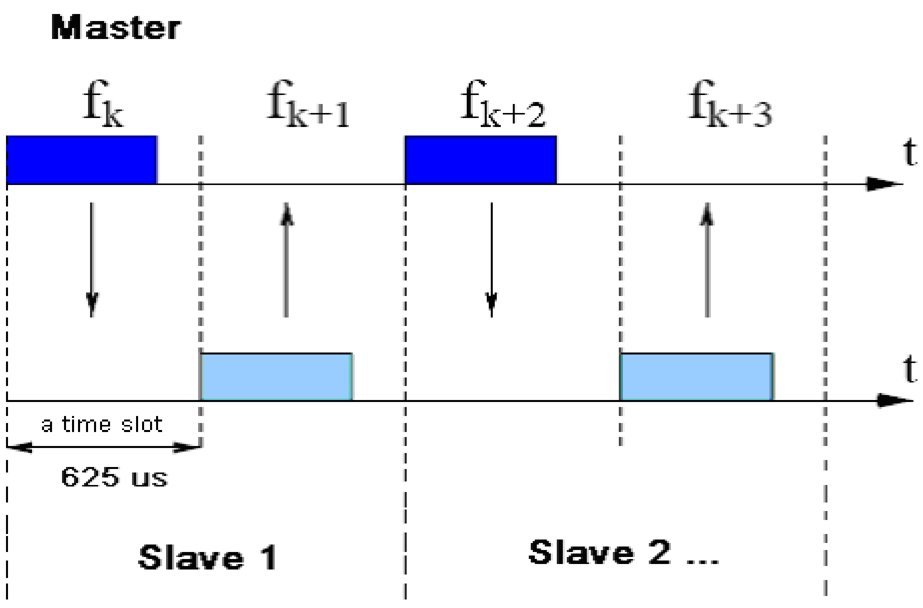


Figure 1.5 FHSS/TDM communication between master and slaves on corresponding slots in a piconet

1.2.1.2 Bridge and scatternet

A bridge node is a Bluetooth device that joins two or more piconets at the same time. By means of a bridge, two or more piconets can be connected and can transfer data between them. A bridge node that is a slave in all of the piconets it belongs to is called a S/S Bridge (See Figure 1.6 (a)) When a bridge node is a master in one piconet and a slave in all other piconets it participates in, is called a M/S Bridge (See Figure 1.6 (b)).

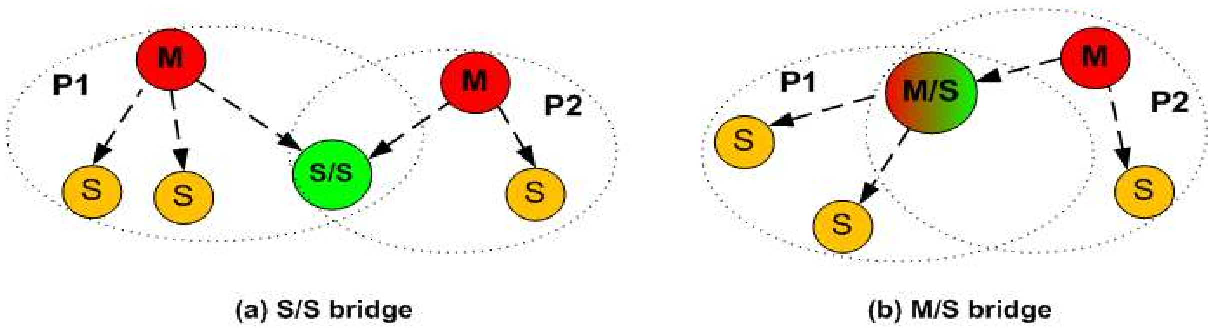


Figure 1.6 Bridge structures

Two or more piconets can connect together via bridges to form a large ad hoc network called a scatternet. Figure 1.7 shows an example of a scatternet, in which three piconets are connected. *Piconet 1* and *Piconet 2* are connected by a *M/S* bridge, while a *S/S* bridge is used to connect *Piconet 1* and *Piconet 3*. By means of this approach, many piconets can join together to form larger ad hoc wireless networks beyond the limited Bluetooth radio range, which allows them with higher capacity and more flexibility.

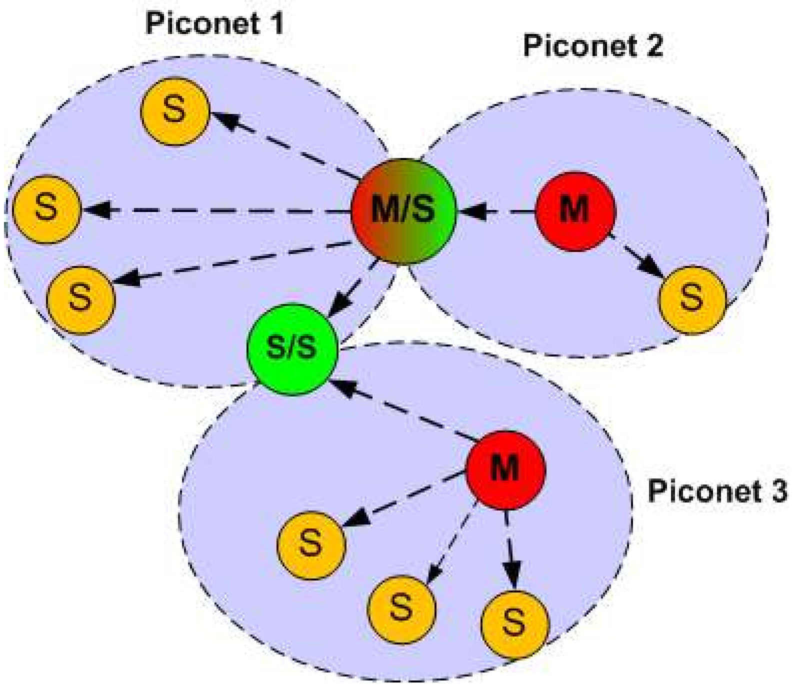


Figure 1.7 A scatternet with three piconets

The bridges act as gateways between the piconets and can forward data from one piconet to another. A bridge node can, however, only communicate with one piconet at a time by synchronizing its clock with the clock of the piconet and then waiting for the polling from the master of that piconet in order to exchange data with other nodes within the piconet. The bridge node communicates with all the piconets that it connects together based on the TDM protocol, through which the bridge divides its time among these piconets and communicates with them one by one. In order to communicate with each connected piconet, the bridge node has to switch from one clock to another, which causes some delay. As a result of this switching delay, the bridge node is likely to become a bottleneck to any traffic.

1.2.2 Metrics for assessing Bluetooth scatternets

Although the Bluetooth specification defines the standard way for Bluetooth devices to connect together to form scatternets, it does not specify a scatternet formation protocol. Various researchers [9-12] have suggested metrics for evaluating whether a scatternet is of a high-quality. This is specified by whether the scatternet:

- Totally relies on local information, and selects masters and bridges based on the device resources;

- Minimizes the time required to generate a fully distributed and connected scatternet with the maximum possible data throughput;
- Minimizes the number of piconets to give quicker routing and reduce the number of packet collisions due to interference from multiple piconets existing in the same area;
- Minimizes both the number of roles a device is assigned and the number of bridge nodes in a scatternet, in order to avoid communication bottlenecks;
- Minimizes the number of M/S bridges to avoid communication blackout in a piconet when the master is active as a slave in other piconets, since a bridge node can only be active in one piconet at a time;
- Be able to handle the addition and subtraction of nodes anywhere on the network;
- Be robust and self-healing by keeping multiple routes between nodes;
- Implements an efficient network routing mechanism to reduce the path length between devices, the amount of control data transmitted for forming and maintaining the network, and the response time for efficient communication;
- Minimizes the power consumption if the scatternet is formed by mobile and portable Bluetooth devices.

1.3 Study objectives

The Bluetooth specification does not fully describe how to form a scatternet and imposes additional restrictions; such as that a piconet can only have up to seven active slaves, need network resilience, low cost device, and low power consumption. This has made modelling a scatternet, in general, complex and difficult. This has generated a significant challenge and has become an open topic of discussion among researchers in relation to the scatternet formation algorithm, protocol, and scheduling scheme, among other considerations. The target of this study is to develop a schema for forming an indoor ad hoc Bluetooth scatternet for information exchange, to implement it on commercial Bluetooth devices in real scenarios, and to evaluate its performance. The intended scatternet has a mesh-like topology, should be reliable and efficient for communication, and must be easily implemented on commercial Bluetooth devices, such as PCs and laptops with Bluetooth USB dongles. The scatternet also needs to target a dynamic environment, where devices can join or leave the network freely without interrupting the whole network.

The main study objectives of this thesis are as follow:

1. Investigate Bluetooth technology and ad hoc Bluetooth network technology, including the Bluetooth Specification and Bluetooth protocol stack;
2. Study existing solutions for Bluetooth network formation;
3. Investigate ad hoc wireless network routing protocols;
4. Investigate available Bluetooth application development platforms, Bluetooth Application Program Interfaces (APIs) and Bluetooth hardware, for the purpose of implementing a Bluetooth scatternet;
5. Design a schema for Bluetooth information exchange network (BIEN) formation;
6. Select a routing protocol and design a routing mechanism for the intended network;
7. Develop a Java based software application to configure the Bluetooth devices to form a BIEN with an efficient routing mechanism for facilitating information exchange over the network; and
8. Design technical measurement methods for testing and evaluating the performance of the resultant scatternet.

1.4 Structure of the thesis

The rest of the thesis is organised as follows. Chapter 2 presents an investigation of the previously proposed major solutions to the problem of Bluetooth scatternet formation, the ad hoc wireless routing protocols, and the wireless mesh network. Chapter 3 describes the hardware and software tools and the platforms used in the development of this work. The following chapters describe the work developed as part of this research related to design, development, implementation, and testing of the Bluetooth information network. It includes Chapter 4, which describes the system architecture of the BIEN and the schema to form an efficient BIEN. Chapter 5 introduces the Java based software application developed in this work for implementing the BIEN network system, a routing mechanism, and an information exchange application. Chapter 6 presents the experiment design and the deployment for testing the implemented BIEN, as well as the evaluation results of the BIEN's performance. The last chapter draws conclusions regarding this work and explores the avenues for future work in this topic area.

2 Literature review

This chapter first reviews and discusses the existing solutions to Bluetooth scatternet formation, and then carries out a brief investigation of the ad hoc wireless routing protocols and the wireless mesh network. Finally, a conclusion is drawn based on the topics discussed.

2.1 Previous solutions to Bluetooth scatternet formation

As the Bluetooth specification does not formally define a scatternet formation protocol, the method for generating an efficient scatternet is still an open topic. As a result, plentiful and diverse solutions related to Bluetooth scatternet formation algorithms, protocols, and schemas have been proposed in the literature. The most promising solutions proposed so far include tree, star, mesh, and ring algorithms.

The main tree algorithms for Bluetooth scatternet formation are proposed in [13], [14] and [15].

Zaruba et al. [13] have proposed two algorithms to form a tree-like scatternet called Bluetree. The algorithms presume that each node knows information regarding the root and the information regarding its one-hop neighbours when booting up. The first algorithm, called a blueroot grown bluetree, initiates by assigning a single node as the "*root*" of the tree, termed the blueroot, which is the root of the bluetree. It is assigned the role of master, and then obtains its direct, one-hop neighbours as slaves. Each slave of the root is assigned an additional role of master and pages its unconnected one-hop neighbours to try to get them as slaves. The tree expands by repeating this process recursively at each subsequent level of the tree until the entire tree has been built. Finally, the bluetree is reconfigured to limit the number of slaves of each master. If a master node has more than five slaves, it selects two slaves and instructs one to be the master of the other one. The new master disconnects from the piconet to form a new piconet, so that each master node has no more than five slaves (See Figure 2.1). In the topology of the bluetree, each node is master of its children nodes and slave of its parent node, except in the case of the root node and the leaf nodes.

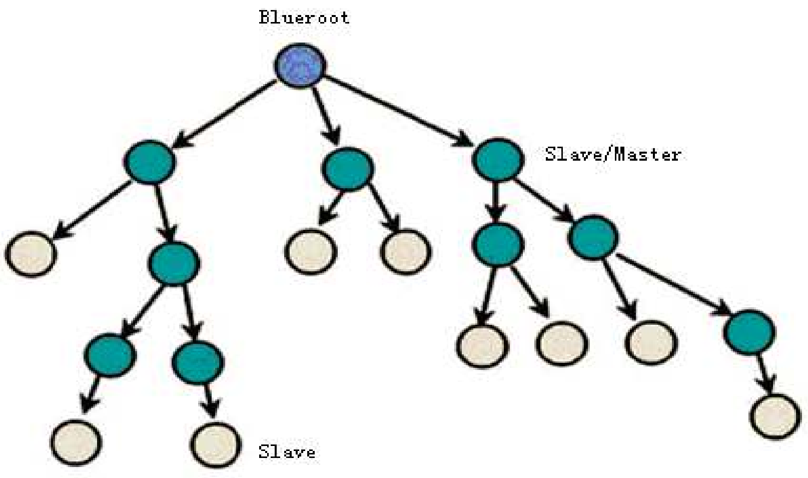


Figure 2.1 A **blueroot grown bluetree**

The second bluetree algorithm, which is called a distributed bluetree, accelerates the process of scatternet formation by selecting some initial roots (sub-roots) to start forming sub-trees, then merges these sub-trees to one root node to build a bluetree. The difference between the two algorithms is that, in the first algorithm, only one node, i.e. the blueroot, starts the formation while, in the second algorithm, several sub-roots form several sub-trees, which are then interconnected to one root node to form a scatternet. In the tree formation, the scatternet can be extended easily and the number of the links between any pair of nodes is kept at one so that the tree topology can simplify message routing due to only one path between any two nodes throughout the network. The algorithms do not, however, prove whether this formation process will actually terminate, or how nodes discover each other and build links between them. Moreover, this bluetree topology will result in a bottleneck situation when the number of nodes in the scatternet increases. In addition, the tree formation has a limitation in terms of fault tolerance, because there is only one route between two nodes. This means that if some links fail or nodes leave the network, the process of recovery and reformation of the tree becomes complicated. Some adaptations to the Bluetree algorithm are introduced by [16] to address the reformation process. Each adaptation tries to decrease the number of slaves of any piconet that has more than five slaves by using different neighbour information. In [10], Basagni et al. provides adaptations to the Bluetrees algorithm by putting '*weights*' into each node to help select the masters.

In [14], a protocol for forming a scatternet with the topology of a tree, called a Tree Scatternet Formation (TSF), has been proposed. It designates master and slave roles when the nodes connect to tree structures. A TSF contains one, or more, rooted spanning trees, which continually try to merge to a topology with fewer trees. During the process of the tree formation, a free, or unconnected, node can only connect to another free node, or to a non-root node, of the tree. A root node with more than one child can only merge with another root node. If a slave (child) node leaves the tree, its master (parent) must determine whether it is a master (parent) with slaves (children), or if it has now become a free node. If it is a master node with slaves as leaves, those slaves that have become free nodes will try to connect to the tree again. If those slaves were in the middle of the tree with child nodes below them, then they become roots after their master leaves, and will try to merge with other roots of the tree. The reasoning behind this is that the TSF protocol is designed to be a self-healing protocol. To avoid bridge nodes becoming bottlenecks, TSF ensures that any bridge node only connects with exactly two piconets. As this algorithm is self-healing and decentralized, it may reduce interruptions to the communication when nodes join, or disjoin, the tree. The TSF tree topology will, however, cause bottlenecks at the root for high bandwidth applications. Simulation results [14] show that a relatively short scatternet formation time is needed to form a TSF, but also show that TSF does not reduce the number of piconets.

A fully decentralized algorithm [15], developed by Law et al., forms a connected scatternet with the objective of reducing the number of piconets and bounded degrees. The resulting topology is tree-like and, therefore, suffers from the same drawbacks as the other tree type solutions, i.e. there are limits to the efficiency in terms of communication and robustness for reconstruction.

Based on such investigations, solutions for multi-hop Bluetooth scatternet formation, that are different from the tree protocols, are proposed in [17], [18], [19], [20] and [21].

In [17] and [18], a three-phase multi-hop scatternet formation algorithm called Bluestar has been presented. The first phase is the device discovery, through which neighbour nodes obtain "*symmetric*" knowledge of each other. The second phase is piconet formation. During this phase, the nodes with the largest weight among their neighbours are selected as masters and create their piconets by attempting to obtain their neighbours with smaller weights as slaves, so that all of the nodes are divided into separate piconets (Shown in Figure 2.2).

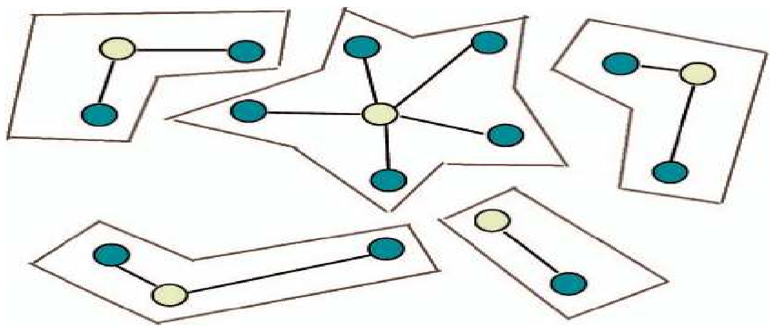


Figure 2.2 BlueStars - Phase 2: Piconet formation

The third phase is a scatternet formation. The masters gather information from their slaves about other masters and select nodes as bridges through which they connect with other piconets to form a scatternet. The resulting scatternet has a mesh-like topology with multiple routes between any pair of nodes (Shown in Figure 2.3).

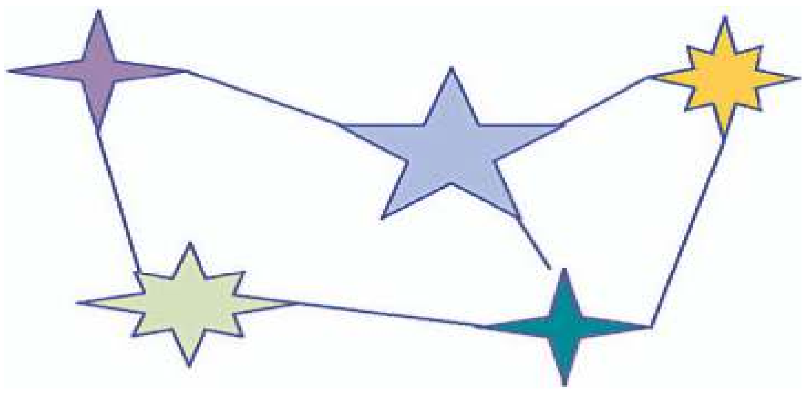


Figure 2.3 BlueStars – Phase 3: Piconets' interconnection to a scatternet

The Bluenet [19] is also a three-phase Bluetooth scatternet formation schema, which is similar to a BlueStars formation with the exception that the nodes are not required to have "*symmetric*" knowledge of each other during the discovery phase. It generates a mesh-like scatternet. Simulation results from this formation show that a mesh-like scatternet can achieve higher information-carrying capacity than a tree-shaped one, however, the Bluenet schema cannot always ensure the building of a connected network.

Stojmenovic developed a dominating set-based three-phase Bluetooth scatternet formation protocol [20]. The first phase is to discover all neighbours within radio range and then build sparse connected graphs based on the unit graph. In the second phase, a degree reduction technology is used in the graph to reduce the number of links of each node. This limits the number of slaves in each piconet to, at most, seven. In the last phase, a scatternet formation protocol is used in the topology to build a scatternet. This technology requires, however, each node to aware of its current geographic location, which can be collected through applications using some extra hardware, such as a GPS.

Petrioli et al. [21] proposed a mesh scatternet formation protocol called BlueMesh, which creates a connected scatternet with piconets having, at most, seven slaves. The process of the scatternet formation is carried out in consecutive iterations, through which each master selects seven slaves from its neighbours to form piconets. Gateways are then chosen to connect other piconets, until a connected scatternet is achieved. The resulting scatternet has a mesh-like structure.

In addition to the solutions of tree, star, and mesh, structure scatternets, [22] and [23] respectively introduced a ring scatternet formation protocol, where the resultant scatternets have a ring structure. This scatternet is simple and easily created, but the efficiency of the communication may be lower compared with other scatternets, due to the large number of hops between any two nodes and the large number of piconets in the scatternet.

Another topology formation protocol has been introduced in [24], and relies on a leader election to collect information about the network and assign roles to each of the nodes. The leader controls the scatternet formation process, so it is a centralized solution. As a result of this the scatternet is not scalable and has a maximum of 36 nodes.

An on-demand formation of the scatternet route algorithm was proposed in [25]. It combines scatternet formation with on-demand routing, which minimizes unnecessary links and route maintenance. In the scatternet formation process, a route request is released, either only after a piconet is fully generated, or it is sent to each neighbour immediately after paging it, without waiting for the piconet to be fully formed.

There are also studies that focus on special aspects of scatternet formation. Two of these studies [26] and [27] analysed the performance of two bridge structures; the M/S bridge, and the S/S bridge, confirming, on the basis of simulations, that the S/S bridge structure provides better performance than the M/S bridge structure, with [27] suggesting the minimization of M/S nodes in scatternets leads to better performance. Another study [28] introduced a method that reduces end-to-end delays in a generated scatternet.

2.2 Discussion of the previous solutions

Although there have already been a number of Bluetooth scatternet formation algorithms already proposed in literature, each of them has its advantages and disadvantages.

Compared with other algorithms, the Bluetree topologies [13] use the smallest possible number of links to establish the connected scatternets and costs the least of the available network resources to maintain the scatternets, however, the resulting network has serious drawbacks because of its hierarchical topology. First, the structure is not reliable. Once one parent node disconnects from the tree, all its offspring nodes will also be split from the tree, meaning that they will not be able to communicate with the rest of the tree. Hence, the scatternet has to be reconfigured in order to maintain a connected network, but the algorithms only describe the initial scatternet formation and do not mention the requirement of rebuilding the scatternet when nodes connect and disconnect from the tree dynamically. This omission makes the Bluetree scatternet very vulnerable, as the breaking of routes will occur frequently under an environment of mobile networking. Furthermore, the Bluetree scatternet may have inefficient communication routing, because a message must go across the tree in upward and downward directions to get to its destination [19]. In addition, the root of the tree and the parent nodes can easily become bottlenecks for communication which will result in problems for the network traffic, since a great amount of piconet traffic must pass across them. Although the TSF protocol [14] states that the TSF tree scatternet is self-healing when a change occurs on the network, there is actually no way to overcome the problem of a bottleneck at the root due to the hierarchical tree structure. As for the Bluestar algorithm [17, 18], the scatternet formation process is simple compared with a tree scatternet when there are only a few piconets in the scatternet and each of the piconets has a few nodes. As there is no limit to the size of the piconets in the Bluestar topology scatternets, however, there will be a potentially large number of slaves in a piconet, which results in a large overhead for parking and reactivating slaves where the number of slaves in the piconet is greater than seven. This makes network communication scheduling complicated when there are many piconets and each piconet has many nodes. A vital problem of the Bluering scatternet [23] is the low efficiency of communication, since the communication between nodes in different piconets has to go through the whole ring network.

Moreover, most of the proposed algorithms are designed to be conducted in a static environment, with no node entering or leaving the scatternet. They focus on studying the initial formation of the scatternets and do not deal with maintenance of the network, especially in situations where devices are added or subtracted, which may often happen under a mobile environment. Therefore, the resulting networks cannot be considered as robust and self-healing. One study [27] holds that most of the proposed solutions do not satisfy the required list of desirable properties, and that actual implementations and comparisons are relatively few.

Another study [29] made a comparison between Bluetree[13], Bluestar[17, 18], Bluenet[19] scatternet formation protocols and another proposed protocol [30]. Simulation results showed that the device discovery stage is the most time-consuming operation. It was concluded that scatternet formation is still a formidable task, due to the limitations of device discovery and the additional complexity imposed by Bluetooth technology on the implementation of distributed algorithms.

Furthermore, even though many papers discuss the issue of Bluetooth scatternet formation, most of these studies have not directly addressed the implementation in real scenarios. They are based on simulation results. In addition, most solutions only discuss the initial scatternet formation, without considering routing issues in real traffic conditions. Some papers [31-37] have discussed the implementation of Bluetooth networks, but they have not considered multi-hop scatternets with integrated routing, but either only localized in piconet implementation without interconnections between piconets or do not consider the routing issue for communication between piconets. Paper [38] discussed a implementation of a table driven routing protocol for Bluetooth ad hoc network based on Java. This however does not state how to perform multiple connections for each node working as different roles in scatternet formation. The paper also does not discussed how to process the routing traffic and communication data in each node. This is vital for efficient routing updates and network convergence in real scatternet operation. Performance of the implementation and experimental results to evaluate the implementation were not discussed.

It could be expected that, in the near future, some adaption could be made to Bluetooth specifications to achieve solutions that comply with a number of desirable properties and make these solutions suitable for being implemented in commercial scenarios[27]. Therefore, based on current Bluetooth technology, developing an approach to form a Bluetooth scatternet that can be easily implemented on commercial Bluetooth devices under a practical networking environment for exchanging information has emerged as the target of this thesis.

2.3 Ad hoc wireless routing protocols

An ad hoc routing protocol is a standard, or protocol, that is used to discover appropriate routes between nodes for data transmission in an ad hoc wireless network. The main goal of these protocols is to build a correct and efficient route between any pair of nodes so that messages may be transmitted between them opportunely. Such a routing protocol also controls how nodes share information with each other and report changes, which allows the network to adjust dynamically to differing situations.

Ad hoc routing protocols may be generally classified into two categories, namely table driven (proactive) routing protocols and source initiated on-demand (reactive) routing protocols [39, 40], which are described in the following sections.

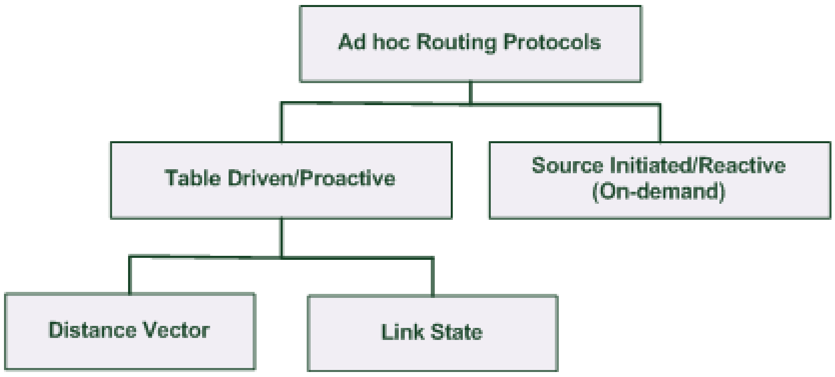


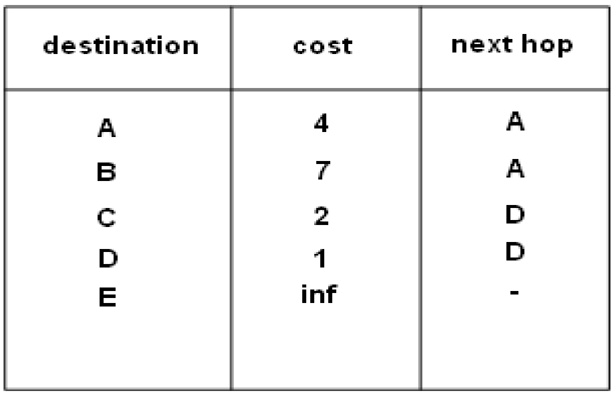
Figure 2.4 Categorization of ad hoc wireless routing protocols

- <u>Table driven (proactive) protocols</u>

These protocols maintain consistent and fresh information on the routes from each node to all of its destination nodes in the network, which requires each node to; use routing tables to store the lists of destinations and the routes to every other node, update these routing tables once there is a change in the network topology, and distribute them throughout the network regularly. As the routing information is continuously updated and propagated all over the network, the shortest paths to all other nodes are always available, whether they are needed or not. This is the significant advantage of these protocols, because it results in fast routing decisions and minimum delays in message sending. As a result, it improves the communication efficiency of the network. Maintaining up to date routing information and distributing it regularly throughout the network will, however, incur a great amount of signalling traffic, and the consumption of power and bandwidth. Moreover, if the size of the network is large and has a large number of nodes, each node needs a large memory for holding the routing table, which is also a limitation for resource-limited mobile devices. It also has a slow reaction to network reconstruction and failure. Hence, it is suited for use in small networks.

These table driven protocols include two types of protocols; the Distance Vector routing protocols, and the Link State routing protocols.

The Distance Vector routing protocol (DV) [41] uses the *Bellman-Ford routing algorithm* [42] to calculate the best route, including the direction and the distance, to every destination in the network. It assigns a number, called the metric cost, to each link between the nodes in the network. The total cost of getting to a destination is achieved by means of mathematical calculations. This cost is the distance to a destination, such as the metrics of the route, or the hop count to the destination.



destination	cost	next hop
A	4	A
B	7	A
C	2	D
D	1	D
E	inf	-

Figure 2.5 An example of a routing table in the DV protocol

This DV routing protocol requires that each node stores a direction and cost pair for each destination in a routing table (See Figure 2.5) and simply informs its neighbours of its routing table regularly, or when a change in the topology of the system is detected. Once having received the routing tables from its neighbours, the receiving node checks for each network path on the received routing table to determine whether there is a better path than the current one in its own routing table. If there is, it picks the lowest cost from the neighbour routing tables received. It then adds this entry into its routing table for re-propagation to its neighbour nodes. Eventually, all the nodes throughout the network will discover the best next hop as being the direction and the route with the smallest total cost; that is, the shortest path for all destination nodes.

This operation of the DV routing protocol is very simple and efficient in small networks. If there are no changes in the topology of the network, the DV algorithm converges to its optimum condition. If the costs of some paths have changed, or a new node joins the network, this algorithm can re-converge to the new optimum condition. If nodes or links have been broken, however, the problems respectively called routing loops and counting to infinity can occur, as the Bellman-Ford routing algorithm does not stop routing loops and undergoes the counting to infinity problem.

The Destination Sequenced Distance Vector routing protocol (DSDV) [43] is a typical example of an ad hoc wireless DV routing protocol.

DSDV improves upon the conventional *Bellman-Ford routing algorithm* to avoid the problem of routing loops, but keeps its simplicity. It requires each node to maintain a routing table, which holds a list of route entries for all other nodes in the network. Each entry holds the routing information for a destination node, marked with a sequence number assigned by the destination node. Each node increases its sequence number when a change happens in the neighbourhood, which allows the nodes to distinguish the new routes from out of date ones, in order to prevent the formation of routing loops. To keep the routing table consistent, the table updates are propagated all over the network periodically, rather than just to the neighbouring nodes no matter what the number of changes in the topology of the network. This increases the computation complexity and makes it inefficient in the area of route updating. As a result, the size of the network able to be supported by the DSDV protocol is limited.

Another well-known dynamic DV routing protocol that has been widely used in spontaneous networking systems is the Routing Information Protocol (RIP) [44]. It uses the hop count as the routing metric to calculate the distance to any nodes in the network. RIP employs a split horizon with poison reverse technique to avoid routing loops. The split horizon is a routing broadcasting rule which forbids a router from sending route information back to the sender. A variation of the split horizon is that with poison reverse. A router here advertises updated routings with unreachable destinations back to the sender for every received update to stop routing loops. RIP also uses a maximum number of hops, which is 15, for each path from the source to a destination, in order to deal with the count to infinity problem. The limit on the hop number, however, also limits the size of the network that RIP can support.

The Link State routing protocol (LS) [41] is another type of the table driven routing protocols, which uses a different method to produce the routing table from that of the DV routing protocol. It requires that each node flood information about other nodes it can connect to throughout the network, and then each node separately creates a map based on this information. With this map, each node can then independently use a shortest path algorithm, such as *Dijkstra's algorithm* [45], to calculate the best path from itself to every possible destination. The routing table is made up of the collection of the best paths. Once a change happens on the network, a notification is flooded through the whole network. Consequently, all of the nodes will recalculate the best routes. This approach is more reliable and uses less bandwidth consumption than the DV routing approach. Compared with the DV protocol, however, it has much more complex procedure and more computations. It also requires more memory for the link state database and the routing table, while DV protocols also have slower convergence time than it does.

- Source initiated on-demand (Reactive) protocols

This type of routing protocol creates routes only when required by a source node. When a node requires a route to a destination node, it starts a route discovery procedure by flooding the entire network with a routing request packet, which travels from one node to another until the destination node, or an intermediate node that has a route to the destination node, is reached. This procedure is terminated when a route is found, or all possible paths have been checked. The route stays valid until the destination has become unreachable to the source, or it is no longer required.

The on-demand protocols are very efficient, as they reduce the control overhead and power consumption through creating routes only when needed. The source node is, however, unable to start sending messages before the route to the destination has been found. The latency time in route discovery is so high that it might not be acceptable for real time communication. In addition, the extreme packet flooding for routing discovery may cause traffic congestion in the network.

The ad hoc On Demand Distance Vector protocol (AODV) [46] and the Dynamic Source Routing protocol (DSR) [47] are two types of on-demand protocols used in wireless networks.

The AODV protocol is based on the DSDV protocol previously described, but has improved the DSDV protocol by reducing the number of route broadcasts through creating routes on-demand. As the nodes that are not located on the on-demand route do not keep the routing information, or do routing table exchanges in the AODV protocol and, therefore, it does not produce extra traffic for the communication. Hence, it is classified as a pure on-demand route acquisition system. Nonetheless, it needs more time to build a connection and the route discovery to initiate communication is more difficult than for other protocols.

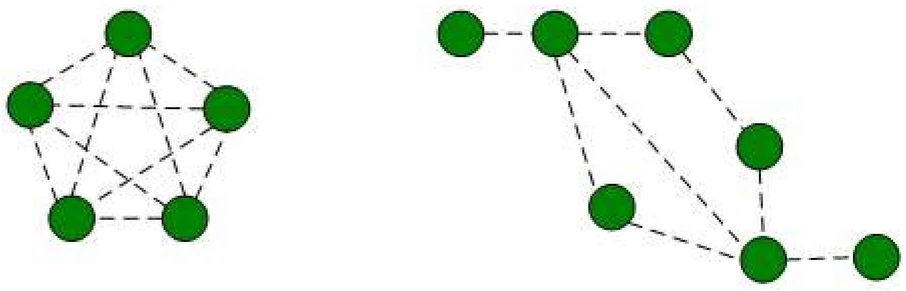
DSR protocol is a loop-free on-demand routing protocol based on the idea of source routing, where a data packet brings the full path to the transmission. It requires each node to keep a route cache holding the source routes known by this node. The routes in this cache are updated when new routes are learned. Although it does not need to broadcast the table update messages periodically throughout the network, in comparison with the table driven protocols, the delay to connection creation is high. It also has significant routing overhead because of the source routing process used. Thus, it is suitable for static and low mobility networks, and its performance will quickly worsen with growth in mobility of the network nodes.

Compared with AODV, DSR uses source routing in which a data packet takes the entire path to be transmitted, while in AODV, the source nodes and the intermediate nodes keep the next hop routing related to each flow for data transmission.

Performance comparisons between the table driven routing protocols and the on-demand routing protocols have been made [48], which show that the table driven protocols perform better on throughput and in terms of the end to end delay than do on-demand protocols. On the other hand, on-demand protocols work better in the area of routing loads than do table driven protocols.

2.4 Wireless mesh network

A wireless mesh network [49-51] is an ad hoc multi-hop network with a mesh topology. It is formed by wireless nodes, in which all nodes are connected to each other directly, or through multiple hops, with there being more than one path for communication between any pair of nodes. It includes two connection types, namely full mesh topology (See Figure 2.6 (a)) and partial mesh topology (See Figure 2.6 (b)). In the full mesh network, every node directly connects to all of the other nodes while, in the partial mesh network, some nodes only connect to those with which they exchange the most information. This mesh network topology was initially developed at MIT for industrial control.



(a) Full mesh topology (b) Partial mesh topology

Figure 2.6 Two wireless mesh network layouts

In a mesh network, each node acts as a router with, usually more than one path to the other nodes, using special routing protocols to forward packets to these nodes via its neighbours, so that all nodes can communicate with each other directly, or via one or more intermediate nodes, without manual configuration. Hence, it is considered to be a self-configuring and self-organizing network.

As each node is connected to multiple nodes, the mesh network is a fully connected network and there are multiple communication paths between nodes throughout the network. Hence, it is reliable. When a node disconnects from the network, or a connection breaks down, the messages can be rerouted automatically via other paths by '*jumping*' from node to node around the broken paths until they reach the destination node, meaning that the rest of the network can still communicate with each other. Also, due to the multiple links between nodes in the mesh network, the distance between them is shortened, which increases the link quality without increasing the transmitter power in single nodes. Hence, the links will be more reliable. As a result, the mesh topology improves the whole reliability of the network so that the mesh network is known as a reliable and self-healing network, as the failure of one or more nodes does not essentially affect the operation of the network. These characteristics are especially suitable for operation in the Bluetooth environment, where nodes join and leave the network frequently at arbitrary times.

In addition, since the operation of the wireless mesh network does not need a system administrator, or a central control point, it is convenient to connect more nodes to enlarge the range and the capacity of the network. Therefore, it is scalable and is capable of dealing with a large network containing hundreds, or even thousands, of nodes.

These characteristics of self-configuration, dynamic self-organization, and self-healing make the wireless mesh network able to be rapidly implemented and easily maintained in highly changing environments. As a result, the wireless mesh networks are more robust than other networks and are generally thought of as a mode of ad hoc networking [52]. Currently this type of network is filling a more and more important role in the wireless mobile networks. It is well suited for the scenarios of office and campus environments, residential spots, military applications, public access spots; such as airports, shopping centres, and sport situations; emergency situations; such as disaster rescue and recovery; and so on [53]. The following are real-life examples of wireless mesh network applications:

On 3 June 2006, Strawberry Fair in Cambridge, Britain, used a mesh network to provide mobile live television, radio, and internet services to approximately 80,000 people [54].

The Meraki Mesh [55] is an intelligent wireless mesh router developed by the Meraki Company in early 2007. It is intended for large-scale, high-capacity network formation for a distance communication with coverage of 250 metres. The speed of the wireless mesh network formed is claimed to be up to 50 Mbps.

Smesh [56] is a wireless mesh network that offers peer-to–peer connectivity, internet connectivity, and fast handoff to mobile users throughout the network. It is mainly designed for carrying out research on real-time wireless network implementations.

In addition, in view of the characteristics of the wireless mesh network mentioned previously, many researchers on Bluetooth scatternet formation have developed different algorithms, protocols, and schemas for Bluetooth mesh network formation, such as Bluestar [17], Bluenet [19], BlueMesh [21], and MTSF [57] as well as the dynamic and distributed mesh-like network protocol proposed in reference [58]. These scatternet formation protocols provide the resulting networks with characteristics of robustness and self-healing.

2.5 Conclusions

Although there are plenty of existing solutions for Bluetooth scatternet formation in the literature, most of the evaluation studies are based on simulation results only, which rely on the Bluetooth environment provided by the simulators. The software-based Bluetooth network simulators cannot, however, currently provide a complete Bluetooth scatternets environment, as they are unable to accurately capture many real-world radio signal propagation effects, such as non-uniform path loss, interference, and distance fading. Thus, evaluation results based on simulations cannot faithfully reflect the performance of Bluetooth networks and indicate how well the Bluetooth technology supports ad hoc wireless networks. This limitation can be overcome by an implementation of physical Bluetooth networks in a realistic experimental environment. Moreover, most of the proposed algorithms on scatternet formation are too large and complex to be implemented in real commercial Bluetooth devices, and few real implementations under realistic scenarios are done based on those proposals. Therefore, developing an algorithm for forming an efficient Bluetooth scatternet that can be implemented in a practical networking environment is still an open issue. This is the target of this project, i.e., to conduct research on real-time ad hoc Bluetooth network implementation, by developing a schema and application for setting up a Bluetooth scatternet with mesh-like topology and an efficient routing mechanism in a test bed, for exchanging data between any participated Bluetooth devices in an indoor interactive scenario; such as conferences, or meetings in offices or homes; in which the network nodes can exchange information instantly, evaluating the performance of the network developed, and investigating how well Bluetooth technology supports ad hoc multi-hop wireless networks.

3 Development platforms

This chapter presents the hardware and software platforms used for developing the BIEN application and implementing the BIEN networks during the course of this work.

3.1 Bluetooth device

The Bluetooth device used in this work consists of a personal computer (PC) with a Bluetooth USB dongle, in which a Bluetooth protocol stack is implemented. This protocol stack is implemented with a standard two–processor architecture (Shown in Figure 3.1).

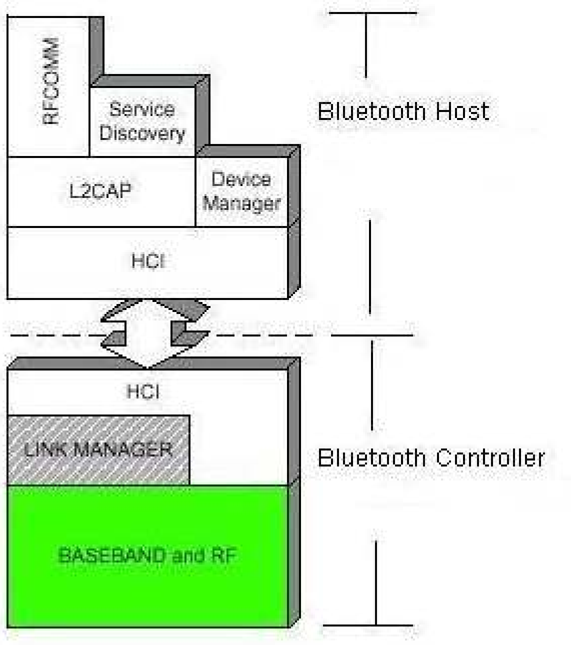


Figure 3.1 Two-Processor Architecture of Bluetooth Protocol Stack

In this implementation, the lowest three layers of the Bluetooth stack known as the Bluetooth controller are embedded in the Bluetooth USB dongle, while the higher layers above the HCI of the stack known as Bluetooth host are built in the operating system on the PC. These two parts compose the full Bluetooth protocol stack.

In this work two models of Bluetooth dongles are adopted, namely the XH8880 DSE Bluetooth USB dongle (See Figure 3.2) and the EA0021 DSE Bluetooth dongle (See Figure 3.3). These two dongles have similar functions, but are sourced from different manufacturers. Both of them are compliant with Bluetooth specification V2.0 + EDR, with a data transmission rate of up to 3Mbps. They are class-1 Bluetooth devices, with a radio range of up to 100 metres and use the fast frequency hopping approach to operate in the 2.4GHz – 2.4835GHz ISM band. The Bit Error Rate (BER) for these devices is -80dbm at 0.1%. With a full speed USB V2.0 interface, they can be plugged in a PC via a USB port and work with the operating systems of Windows 2000/XP, MacOS 10.2.8 or above, and Linux 2.4.6 or above.

Figure 3.2 XH8880 DSE Bluetooth USB Dongle

Figure 3.3 EA0021 Bluetooth USB Dongle

As both types of dongle support multiple connections, they are selected as the Bluetooth devices for PCs or laptops, and take on different roles in the BIEN scatternets of this project (See the Test bed in Section 6.1.2).

The Microsoft Bluetooth stack is adopted as the Bluetooth host. It is built in Service Pack 2 for Windows XP, as part of the operating system for supporting Bluetooth wireless technology natively in Windows XP. It can interface with most Bluetooth adapters and dongles on the market to form a complete Bluetooth protocol stack. All of the PCs that form the Bluetooth devices in this work are Intel Pentium IV 3GHz standard PCs, with 1.00 GB of RAM based on Windows XP SP2.

The experimental network environment is described in Section 6.1.2.

3.2 Software development tools

This section introduces the Java-based BIEN software development tool and environment used in this work.

3.2.1 Java technology

Java [59] is a high-level, object-oriented programming language with high performance. It is developed by Sun Microsystems in 1995, but it is available through the open source community and has become more and more independent of Sun. Although the Java language was influenced in diverse ways by C and C++, it is easy to learn and use, unlike C and C++. As Java code is portable, object orientated, intended for running in a distributed environment, and supports a combination of large numbers of object classes it is suitable for developing large applications for big projects. Java program platforms are independent and can be run on any computer systems with the Java virtual machine (JVM) installed; including Microsoft Windows OS, Solaris OS, Linux and Mac OS; regardless of the computer architecture on which the program was initially developed.

The Java platform (Shown in Figure 3.4) that sits on the operating systems is a collection of related software programs with two components, there being the JVM and the Java application programming interface (API) [60]. The API includes libraries of associated classes and interfaces that contain a number of off the peg software programs and give many functional capabilities.

The Java platform has three different editions, namely Java 2 Platform Standard Edition (J2SE), Enterprise Edition (J2EE), and Micro Edition (J2ME). Each of them is a combination of a language version, a Java Compiler, a set of APIs, and a JVM (See Figure 3.4).

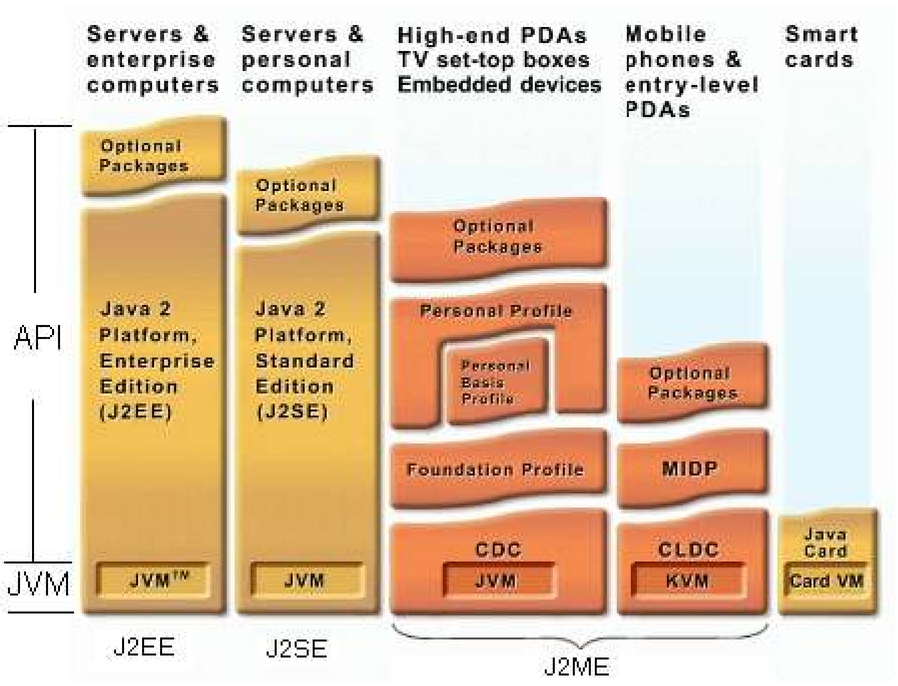

Servers & enterprise computers
Servers & personal computers
High-end PDAs TV set-top boxes Embedded devices
Mobile phones & entry-level PDAs
Smart cards
API
JVM
Optional Packages
Java 2 Platform, Enterprise Edition (J2EE)
JVM™
Optional Packages
Java 2 Platform, Standard Edition (J2SE)
JVM
Optional Packages
Personal Profile
Personal Basis Profile
Foundation Profile
CDC
JVM
Optional Packages
MIDP
CLDC
KVM
Java Card
Card VM
J2EE
J2SE
J2ME

Figure 3.4 The Family of the Java Platform

J2SE is the core of Java technology and is also the base for J2EE. It is used in all classes of Java programming, from desktop applications to J2EE applications. J2EE is J2SE, along with diverse functional APIs for enterprise applications, such as multi-tier, client-server, Web services, and so on. It can be considered to be a superset of J2SE, because any J2EE application uses the J2SE libraries, with J2EE and J2SE using an identical language. J2ME provides a group of APIs for developing mobile applications targeted at small embedded devices with limited resources, such as mobile phones, pagers, PDAs, and automobile navigation systems.

JSR-82 [61] is the office standard Java API for Bluetooth wireless technology that enables Java applications to communicate with each other in Bluetooth environments. Hence, Java is a suitable programming language for developing the BIEN application for this project.

3.2.2 Bluetooth API

The Bluetooth API used for the software application to communicate with the Bluetooth protocol stack is the free third party software product known as BlueCove [62]. This is an implementation of the JSR-82 for J2SE. It provides a Java library for Java applications to interface with the Bluetooth protocol stack underneath, including Service Discovery Application Profile (SDAP), Serial Cable Emulation Protocol (RFCOMM), Logical Link Control and Adaptation Protocol (L2CAP), and Generic Object Exchange Profile (OBEX). As it supports J2SE based applications communicating with Microsoft

Bluetooth stack built in the Windows XP SP2 and is open source software, it is selected as the Bluetooth API for supporting the BIEN application to communicate the underlying Bluetooth protocol stack.

3.2.3 Eclipse

Eclipse [63] is an open source integrated development environment (IDE) with the aim of building an open development platform consisting of extensible software frameworks, tools and runtimes for creating, deploying and maintaining software during its lifecycle. It is mainly developed in Java and can be extended by plug-ins built for this framework by any user to enhance its capabilities.

Eclipse provides essential tools for Java programmers; including a Java IDE, Java Development Tools (JDT), and the built-in Eclipse Compiler for Java (ECJ); to develop Java applications, including J2SE and J2EE applications. Integrated with the EclipseMe plug-in [64], it enables J2ME programmers to develop applications for mobile devices. As the Eclipse IDE provides Java developers with everything needed for creating Java applications, it is believed to be the best Java development tool available. Therefore, Eclipse is selected as the BIEN application development platform for this project.

3.3 Summary

This chapter described both the hardware and software platforms for developing the BIEN application and deploying the BIEN networks in this work.

4 Bluetooth information exchange network

This chapter describes the system architecture of a Bluetooth information exchange network (BIEN) developed by this research. This includes a schema for the BIEN topology design and network formation protocol.

4.1 System Architecture

4.1.1 Bluetooth information exchange network (BIEN)

This thesis presents an indoor ad hoc Bluetooth wireless scatternet, namely a Bluetooth information exchange network (BIEN). This is an ad hoc personal area network (PAN), based on Bluetooth and Java technologies. It is developed using a set of Bluetooth enabled devices. These devices can spontaneously establish a dynamic multi-hop wireless network using Bluetooth technology without the need for a formal network infrastructure, centralized administration, fixed routers, or access points. Each Bluetooth device, termed a BlueNode, has an identical configuration and acts as a router with, usually, more than one path to the other nodes. An implementation of an efficient dynamic routing protocol forwards data to all other nodes in the network via its immediate neighbours, so that all BlueNodes can communicate with each other directly or via one or more intermediate nodes without the need for manual configuration, and regardless of whether they are in radio range of each other. Hence, BIEN is considered to be a distributed multi-hop, self-configuring, and self-organizing network.

The key function of the BIEN is information exchange between BlueNodes in the network by means of file or image transfer or chatting by text with each other.

Although the experimental BIEN created in our study has a special topology (See Figure 4.1), a number of different topologies and the sizes of BIENs that BlueNodes can form can be assumed. As an efficient table-driven routing protocol implemented in each BlueNode maintains network-wide route information for fast route setup, BIEN can be implemented in dynamic environments where devices can freely join or leave the network at any time without interrupting the communication of the whole network.

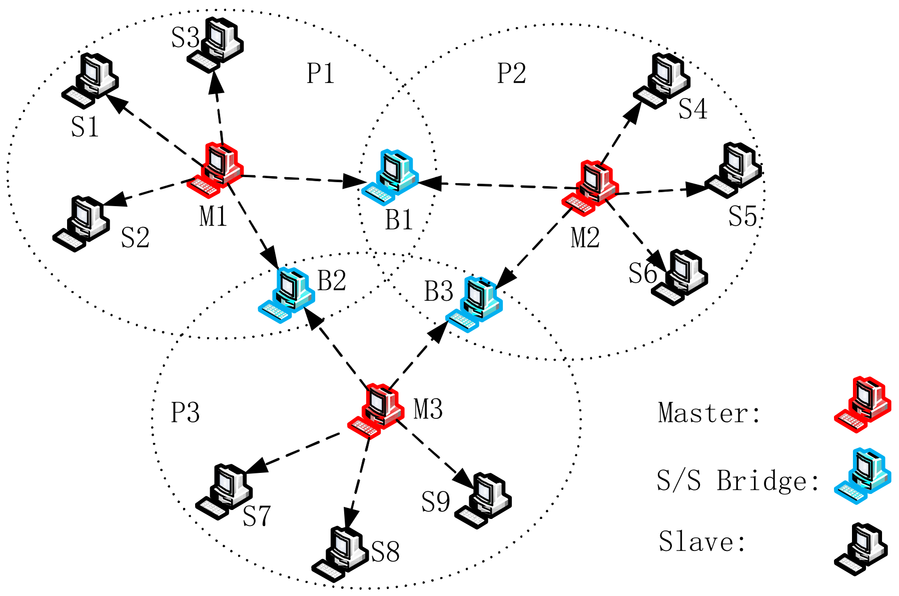


Figure 4.1 Architecture of an experimental BIEN

As every BlueNode in BIEN can freely exchange information with any other BlueNodes in the network, BIEN can be used to quickly exchange information and instantly acquire data between any Bluetooth devices configured as BlueNodes for indoor interactive scenarios, such as conferences, or meetings in offices or homes. It can also be easily adapted to a mobile version (J2ME based), which will be able to be used in emergency conditions; such as search and rescue operations in natural disasters, or military conflicts; where data acquisition operates in harsh environments.

4.1.2 BlueNode

The design structure of its configuration consists of four parts (Shown in Figure 4.2), which are the Bluetooth controller, the Bluetooth host, the Bluetooth application programming interface (API), and the Java based software application for forming and using scatternets. The first two parts comprise the Bluetooth Protocol Stack. The Bluetooth Controller, which includes the lowest layers of the stack, comes from the Bluetooth dongle, while the Bluetooth host, including the higher layers of the stack, is built-in to the operating system on a PC or laptop; such as Windows XP SP2, MacOS, or Linux. This is a standard two–processor structure used to implement a Bluetooth protocol stack for a BlueNode. The Bluetooth APIs used for the BIEN software application's communication with the Bluetooth protocol stack is free third party software, BlueCove. It is an implementation of the standard Java APIs for Bluetooth wireless technology (JABWT), i.e., JSR-82, for J2SE. The Java-based software application is an implementation of the Bluetooth network formation, a routing protocol, and BIEN networking applications. The network formation is responsible for connecting Bluetooth devices to form a network. The routing protocol provides an effective routing mechanism for each BlueNode to determine and maintain the best routes to any other nodes in the BIEN. The networking applications offer user information exchange utilities and a graphical user interface (GUI). Both the software application and the Bluetooth API sit on top of the RFCOMM layer of the stack. The data is transferred between the applications and the firmware, working its way down the stack.

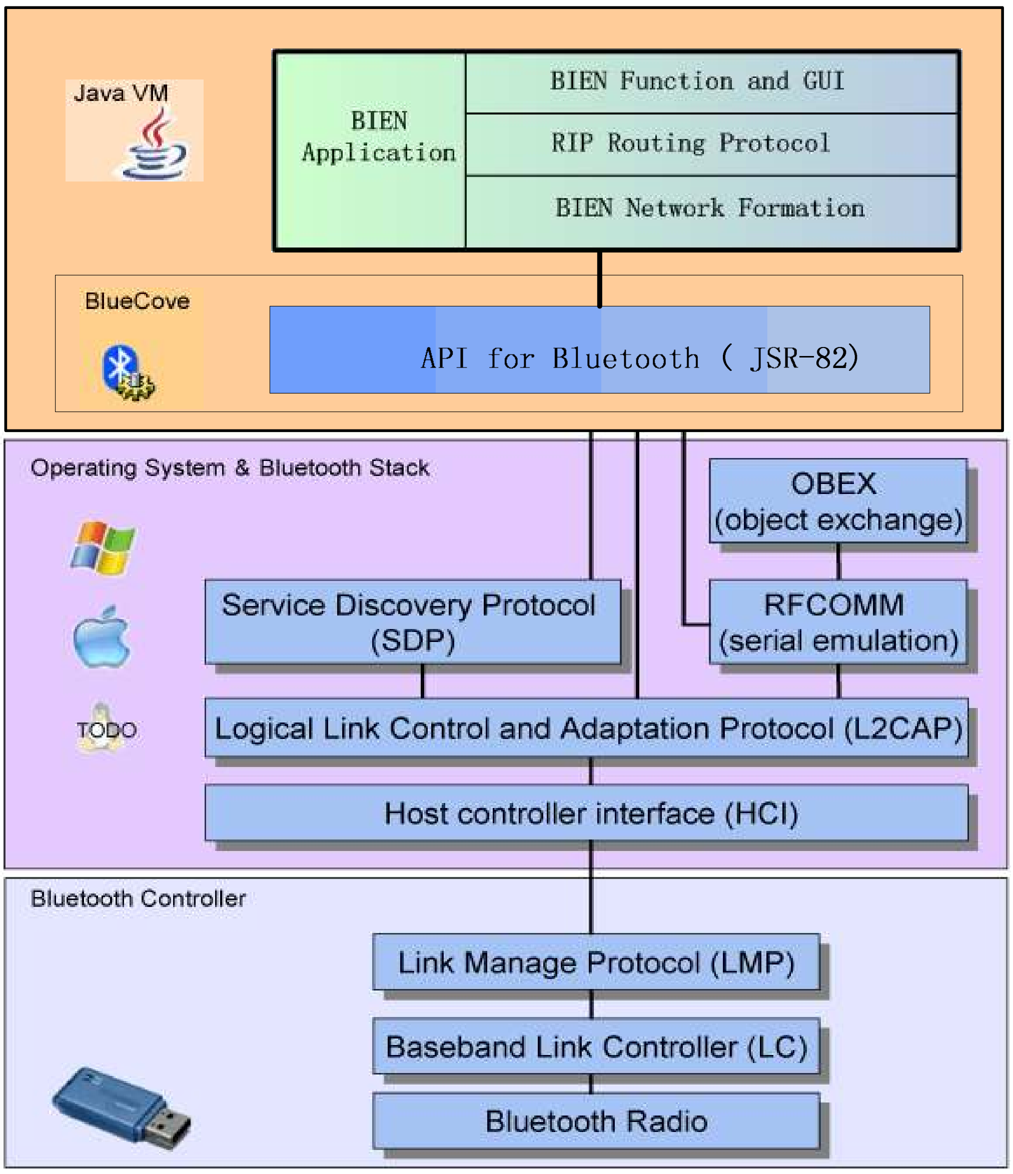

Java VM
BIEN Application
BIEN Function and GUI
RIP Routing Protocol
BIEN Network Formation
BlueCove
API for Bluetooth (JSR-82)
Operating System & Bluetooth Stack
OBEX (object exchange)
Service Discovery Protocol (SDP)
RFCOMM (serial emulation)
TODO
Logical Link Control and Adaptation Protocol (L2CAP)
Host controller interface (HCI)
Bluetooth Controller
Link Manage Protocol (LMP)
Baseband Link Controller (LC)
Bluetooth Radio

Figure 4.2 Configuration of a BlueNode

Bluetooth devices configured as BlueNodes, as shown in Figure 4.2, have an identical ability to connect to each other to form a BIEN and perform the information exchange function within the network.

4.1.3 Routing mechanism

The task of the routing mechanism is to discover the best path, with the minimum length and termed hop count, from a source node to any other nodes in a BIEN. This routing mechanism is deployed in each BlueNode, so that each node can act as a router to determine and maintain routes to all possible destinations within the network with the purpose of providing efficient communication. The routing protocol used in BIEN is the best known and most widely used *Distance Vector routing protocol (DV*), referred to as the *Routing Information Protocol (RIP)* (Referred to in Section 2.3). RIP is based on the *Bellman-Ford algorithm* with the purpose of computing the best routes to all destinations in networks. Each node maintains a routing table that holds the routes to all other nodes in the network. It uses a hop count, which is the number of hops that a packet has to travel through to the destination from a source in a network, as a routing metric to measure network path length. This hop count will increase by one every time the packet crosses a node. The maximum hops in a path from the source to the destination are limited by 15. This is to prevent the routing hops from continuing indefinitely. It sends its full routing table out to all its immediate neighbours, which are the directly connected nodes, every 30 seconds and when there are changes in the network for routing updates. Based on the information in the tables received from its neighbours, each node uses the Bellman-Ford algorithm to calculate the distance; i.e., hop count; and determine the direction; i.e., next hop; for the shortest path to each destination node in the network. Although limiting the maximum number of hops to 15 can prevent routing loops from continuing indefinitely, this also reduces the size of networks it can support.

The reason for choosing RIP for routing in BIEN is that it is one of the most durable of all routing protocols and is the most commonly used protocol for managing routing information within an internal network or autonomous system. Also, because it works well with small networks with a maximum of 15 hops allowed in a path from a source to a destination, it is an effective solution for routing in ad hoc Bluetooth wireless scatternets. As the Bluetooth characteristics preclude it from forming very big networks, the maximum 15 hops in a path from a source to a destination can cover around a thousand destinations for a network with a hierarchical topology, so it is sizable enough to build a large ad hoc Bluetooth wireless network. Moreover, the advantages of the availability of the routes to all destinations with minimum latency time compared with other routing protocols; such as source initiated on-demand routing protocols; the simplicity of the computation and operation compared with DSDV, easy configuration and administration, and the ability to dynamically adapt to network changes, mean that it meets all of the requirements for routing in BIEN. In addition, as the size of Bluetooth networks is small compared with other wireless networks, the routing table held by each node is small, so they do not occupy large amounts of memory and the traffic caused by routing maintenance is not significant. In BIEN, each BlueNode maintains a routing table containing the routes with the shortest distance; i.e., smallest hop count; and direction; i.e., next hop; to all other BlueNodes, based on their unique Bluetooth addresses in the network. Each BlueNode sends the entire routing table to its immediate neighbors every 30 seconds as well as when there are changes in the network topology. BlueNodes calculate the best routes that have smallest hop count value throughout the network, based on the routing tables received, and update their own routing tables to reflect the new routes so that data packets can be directly forwarded to their destinations with minimum latency of routes.

In BIEN, RIP is implemented on the top of network formation applications and runs at the network layer of the *Internet Protocol Suite*. In order to improve its performance, the split horizon and triggered update mechanisms are used to prevent incorrect routing information from being propagated and to speed up the network convergence when there are changes in a network's topology. Although RIP by default limits the maximum number of hops in a path from the source to a destination to 15 to prevent routing loops from continuing indefinitely, it is still too big for an ad hoc Bluetooth network. Therefore, considering the Bluetooth environment, reducing the maximum number of hops in a path is reasonable and practical, and it can provide better stability features for networks. In this study, 10 is the maximum number of hops in a path. As the topology of the BIEN in our experiments is mesh-like, this is high enough to create a large Bluetooth network.

4.2 BIEN formation schema

Although there are no restrictions regarding the topology and size of a BIEN created by BlueNodes, in order to optimize network performance, a schema for constructing efficient BIENs with a mesh-like topology is proposed. This schema includes rules for topology design and a protocol for network formation.

4.2.1 BIEN topology design

Based on the metric for assessing Bluetooth scatternet discussed in Section 1.2.2, rules for designing BIEN topology are described as follows:

1) There are at most seven slaves in a piconet.

This limit is imposed to avoid the overhead of parking and un-parking slaves for masters in the piconets, in order to save intra piconet resources for communication, as each piconet only allows seven active slaves to communicate with their master and any Bluetooth activities consume bandwidth.

2) Only S/S bridges are used for piconet interconnection.

The reason why M/S bridges are not included in the BIEN is to avoid communication blackouts in a piconet when its master is active as a slave in other piconets because a bridge node can only be active in one piconet at a time and has to switch between different piconets it is participating in, using a time multiplexing protocol to communicate with them one by one. Therefore, the communication in a piconet will suffer a blackout if the master is active as a slave in other piconets, since all communication between slaves in a piconet has to go through the master. In addition, based on some studies in [27] and [26], it is preferable to avoid using the master node as a slave node in other piconets to avoid communication bottlenecks.

3) Each bridge node only interfaces two piconets.

As discussed above, bridge nodes have to exchange data with all the interfaced piconets one-by-one, based on time division. They present a bottleneck to the traffic through the piconets in which they participate. Minimizing the number of piconets that a bridge interfaces with can reduce the bridge switch overhead time, reduce resource consumption caused by switching overheads, and improve the data rate a bridge can perform between the piconets, so that the bottleneck effect to traffic can be decreased. This is helpful to improve network performance in areas such as throughput and latency.

4) There is only one bridge between any two piconets.

This is adopted to reduce redundant links between the same pair of piconets, in order to keep network resources as much as possible only for communication. It is also helpful to reduce the bridge overhead, in order to improve communication efficiency.

5) Each piconet may connect to more than one piconet.

This is adopted to improve the reliability of a BIEN, as setting up links between a piconet and other piconets can build a network with a mesh-like topology. This provides multiple paths between nodes in the network, which improves the level of network connectivity and increases the fault tolerance capacity, so that it is resilient to changes in the network topology.

Maintaining good connectivity through multiple links does, however, unavoidably cost some network resources. This has an impact on the communication efficiency of the network. Thus, a compromise will be made on the number of piconets that each piconet may connect to when forming a stable and efficient BIEN with a reasonable level of connectivity and enough network resources for communication.

4.2.2 BIEN formation protocol

Based on the rules of scatternet topology design, outlined above, a simple and fast protocol to form BIEN is proposed, which is carried out in three steps, namely piconet formation, piconet interconnection and network expansion.

1) Piconet Formation

First, all BlueNodes begin as free nodes, without any knowledge of each other. There are no masters or slaves. Each node determines its own role. To initialize a piconet, the most important step is role selection. Normally, in real life indoor scenarios; such as in conference or meeting rooms; it is easy to collect information regarding all of the currently available nodes in order to estimate the number of masters required to initiate a BIEN. Therefore, the control of role selection for each node is given to the user of the node. The master nodes may be determined based on their available resources, as the master role consumes the most resources.

In order to reduce interference from inquiry train collisions, which can result in an increase in the time required for device discovery, only master nodes perform device discovery to pinpoint nearby available BlueNodes. This will improve the efficiency of device discovery. At the same time, the remaining nodes that assigned themselves the slave role enter the inquiry scan and page scan modes to await their discovery and connection by a master. Once a master has acquired a list of nearby nodes by their Bluetooth device addresses, friendly name, and device class, it will begin to connect them one by one as slaves to form a piconet. Each initial piconet is limited to having at most five slaves to reserve at least two connections for the master to connect bridge nodes for piconet interconnection. Once a node has become a slave of a master, it instantly stops inquiry scanning and page scanning to make it unavailable to the other masters. At the end of this step, several independent piconets have been formed with the currently existing nodes (See Figure 4.3 below).

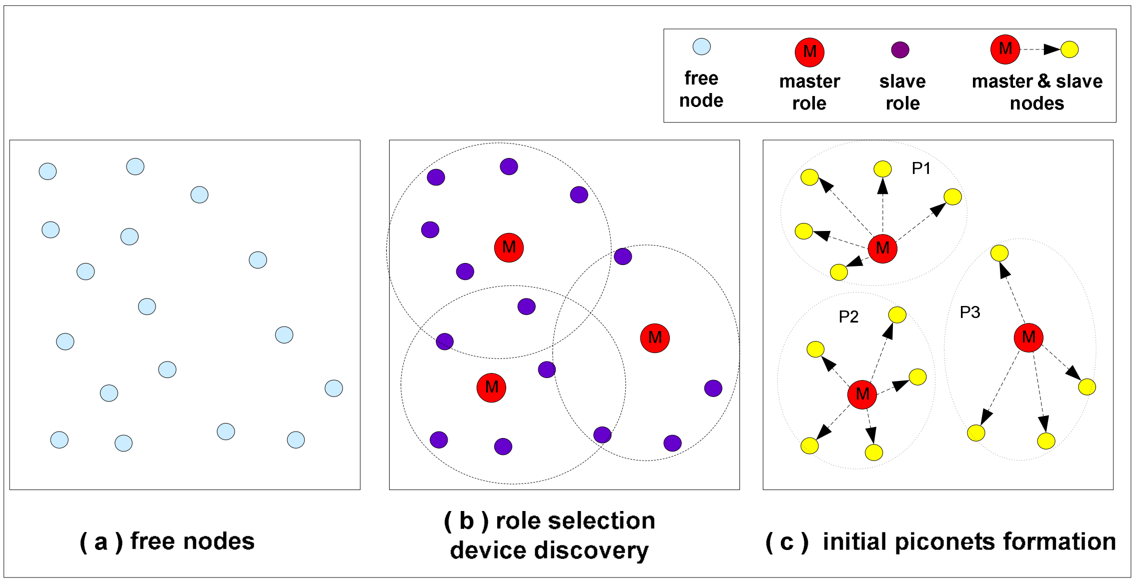


Figure 4.3 Piconet formation in a set of 17 BlueNodes

2) Piconet Interconnection

The master of each initial piconet selects at least two of its slaves to perform the bridge role, based on the device resources, by instructing them to re-enter the inquiry scan mode in order to be discovered and connected to by the masters of other piconets, which ensure piconets' interconnection by S/S bridges. When the slave has received a new connection from one of the other masters, it becomes a slave for both masters in connecting these two piconets as a S/S bridge. This bridge node then immediately makes itself unavailable to avoid more connections from other masters, which then guarantees that one bridge node only connects two piconets. The bridge sends its master list to both of its masters to avoid multiple connections being set up between the same pair of piconets. At the same time, the master of each piconet performs device discovery continuously to find the slaves of other piconets and connect them as its slaves. This is to build connections with different piconets through S/S bridges, until each master has at most seven slaves. Thus, all piconets are interconnected via S/S bridges and a BIEN with a mesh-like topology is formed (Shown in Figure 4.4).

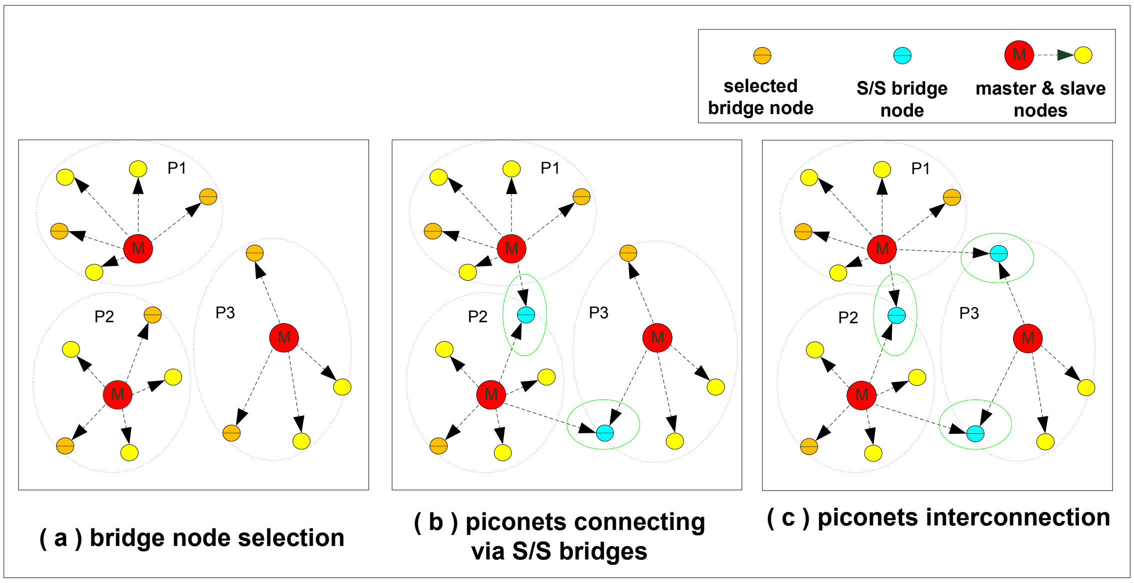

selected bridge node
S/S bridge node
M
master & slave nodes
P1
P2
P3
M
(a) bridge node selection
(b) piconets connecting via S/S bridges
(c) piconets interconnection

Figure 4.4 Piconets interconnection

Each master node in BIEN does not perform an inquiry scan or a page scan to avoid becoming a M/S bridge when other masters discover and connect it as a slave.

3) Network expansion

After the initial BIEN has formed, the masters that do not have seven slaves can perform device discoveries regularly to look for new nodes to join into their piconets. On the other hand, if a new free BlueNode wants to join an existing BIEN, the user can select the role of being either a slave or a master. As a slave it waits for a connection from a master of one of existing piconets. As a master it will connect with slaves from existing piconets to form a new piconet (See Figure 4.5).

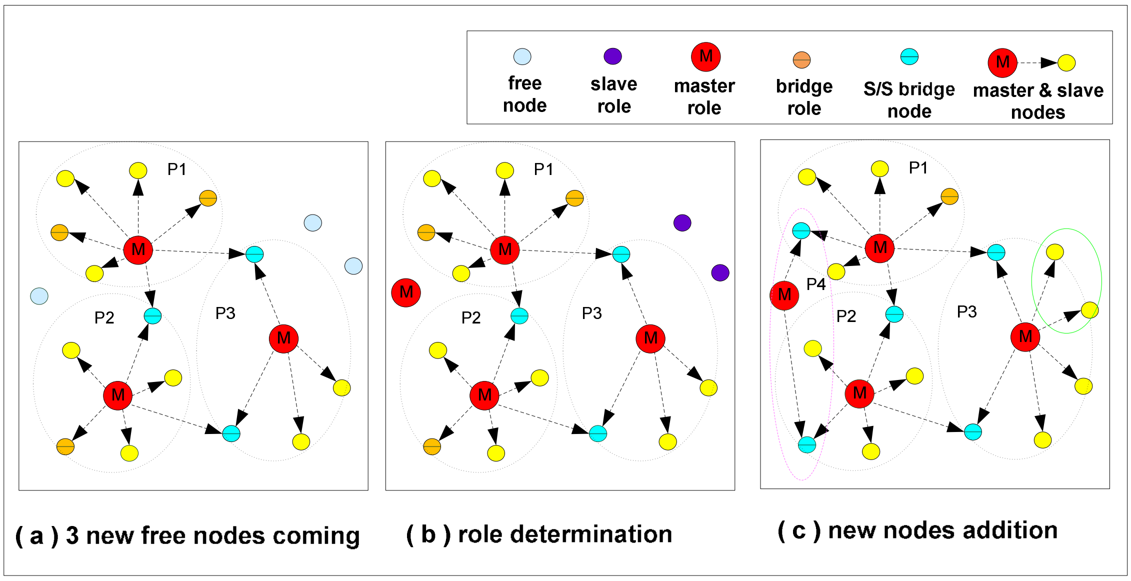


Figure 4.5 Network expansion through three new nodes joining the network

A BIEN created with BlueNodes compliant with the formation protocol discussed above has at most seven slaves in each piconet. Each piconet has multiple connections with different piconets via the S/S bridges. Each bridge only connects two piconets and there is only one bridge between a pair of piconets. With the exception of the bridge nodes, which participate in two piconets, all nodes including the master and slaves only belong to one piconet. This means the average number of roles each node is assigned is very small, which ensures that the network resources are spread evenly through the network to avoid bottlenecks in communication. With a few more links set up between different piconets, it has a very balanced network topology, with multiple paths between any pair of nodes in the network. This makes its routing robust and resilient to disconnections within the network.

4.3 Summary

This chapter presented the system architecture of the BIEN network proposed in this thesis. A schema for generating an efficient BIEN was also discussed. This is followed by a detailed description of the software application implementation of the BIEN.

5 BIEN software implementation

This chapter describes the BIEN software application implemented in this work. The first section outlines the architecture of the software application implementation. This is followed by a description of each software component. Finally, the chapter concludes with a short summary of the main points presented.

5.1 Overview

BIEN application is developed using Java technology with the *BlueCove* library. It sits on the top of the RFCOMM layer of the Bluetooth protocol stack and runs on the Windows XP, or Linux, operating system (Refer to Figure 4.2). BIEN is applied to configure Bluetooth devices as BlueNodes with the ability to connect to a network, route computations, and exchange information with others in the network. This is to enable the nodes to autonomously form a dynamic Bluetooth network for sharing data, or transferring files, without the requirement of a network infrastructure and administration. The J2SE-based BIEN software application consists of three software entities, namely a Bluetooth network formation application, a routing protocol application, and a BIEN function and GUI application (Shown in Figure 5.1).

The Bluetooth network formation application performs the utility of connecting the other Bluetooth devices to form a multi-hop Bluetooth wireless network. The routing protocol application performs the routing mechanism by implementing the RIP routing protocol to compute routes and update routing tables for forwarding packets to any other nodes in the network. It is above the network formation application, with both of them being located within the network layer of the *Internet Protocol Suite*. The BIEN function and GUI application is an implementation of the information exchange function. It also provides a graphical user interface (GUI) panel to enable users to use the function.

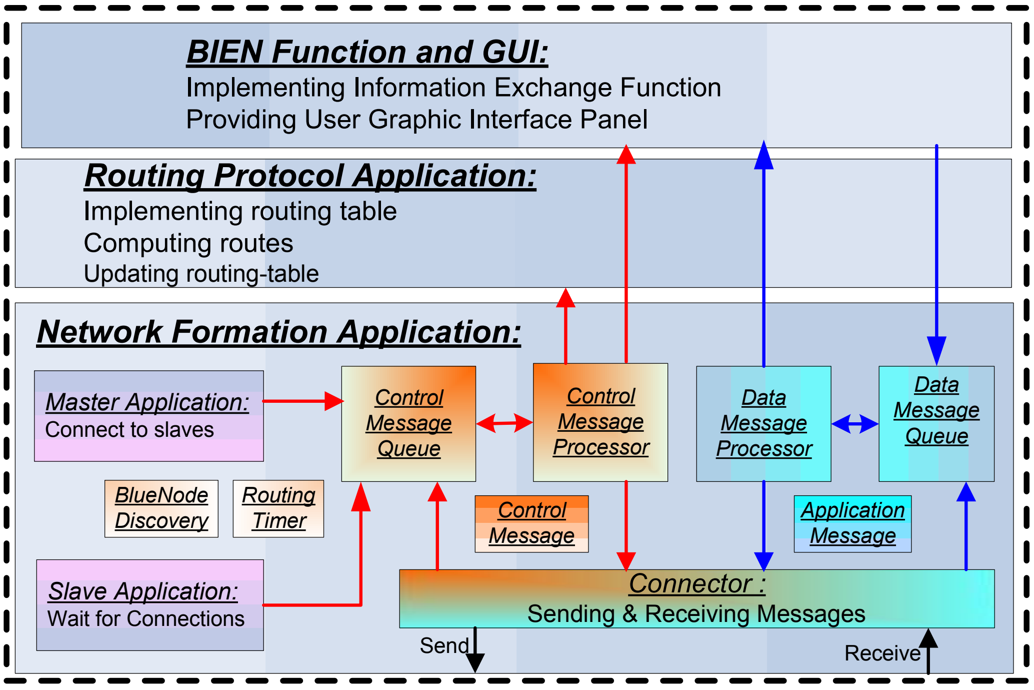


Figure 5.1 Block diagram of BIEN software application

The architecture of the application implementation is illustrated by a UML class block diagram in Figure 5.2 (The complete UML class diagram of this system is located in Appendix II). The features and technical details of how each software entity was implemented into executing their responsibilities are explained in the following sections.

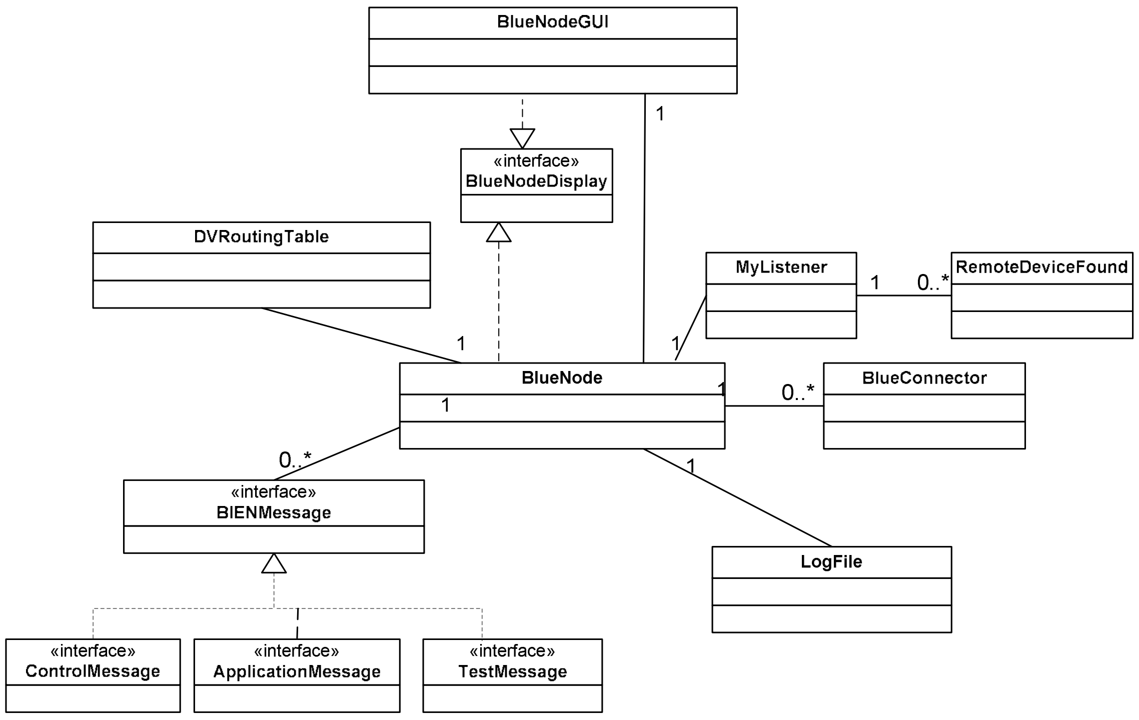


Figure 5.2 UML class block diagram of BIEN software application

5.2 Network formation application

This application implements the functions of BlueNode discovery, connection management and communication management for a Bluetooth device.

5.2.1 BlueNode discovery

This performs device discovery in order to find nearby Bluetooth devices that are configured as BlueNodes, with the purpose of forming a BIEN. In the Bluetooth specification, the operation of *device discovery* is referred to as an *inquiry*. To spontaneously form an ad hoc BIEN network, each Bluetooth device must be able to discover nearby BlueNodes. A *MyListener* class is developed in which a standardized way of device discovery, provided by JSR-82, is implemented to perform device discovery using the *inquiry* procedure. During the *inquiry* process the Bluetooth address, friendly name, and device class of the nearby discoverable devices will be received. The unique Bluetooth address of each Bluetooth device received is used to identify different Bluetooth devices in the BIEN network.

When a new Bluetooth device is discovered, the *MyListener* class continues to perform a service discovery to look for the BIEN service, by locating its unique service *Universally Unique Identifier* (*UUID)* in the device, to determine whether the device is a BlueNode that is able to form a BIEN. Once the procedure of BlueNode discovery is complete, the node that performed the procedure holds information regarding nearby BlueNodes addresses, friendly names, and device classes.

5.2.2 Connection management

5.2.2.1 Connection architecture

This function allows the local node to take on the role of master, slave, or both states at the same time, in order to set up connections with any other BlueNodes to form a BIEN. Bluetooth has a client-server architecture, in which a client initiates a connection and a server receives the connection when they are trying to connect together. Based on the BIEN formation protocol, the client application is assigned the role of master. Meanwhile, the server application is assigned the role of slave in the implementation. Here, the node which initiates a connection is the master (the client), and the one who receives a connection is the slave (the server). In order to enable a BlueNode to act as the master or the slave, a peer-to-peer architecture is adopted in which both master (client) and slave (server) applications are implemented and incorporated into the code.

The master role disables the inquiry scan and the page scan, and performs BlueNode discovery, BIEN service discovery, and makes a connection request to the other BlueNode performing the slave role. When a master node completes the device discovery and tries to connect to a node it has found as a slave, it uses the slave node's Bluetooth address to make a connection request to the slave. This method being referred to as *Connector.open(masterURL)* in the master application. The parameter of the method, the string *masterURL,* has the pattern:

masterURL="btspp://[SlaveBTAddress:Channel_identifier];master=true"

It includes the *Bluetooth Serial Port Profile (SPP);* that sets up an *RFCOMM* connection for communication; the slave's Bluetooth address and the channel identifier rather than the BIEN service *UUID,* as the slave node has been identified as a BlueNode in the stage of BlueNode discovery. This allows for the time required for a service search to be skipped while a master is connecting to a slave, so that it is more efficient in reducing connection delays. It also claims itself as a master in the connection by the inclusion of *"master =true"* in the URL string.

On the other hand, the slave role registers the BIEN service with the unique *UUID* for the application, enables inquiry scan and page scan to allow it to be discovered by master nodes and waits for an incoming connection request from a master node by calling the *Connector.open(slaveURL)* method to obtain a *StreamConnectionNotifier*. This then uses the *StreamConnectionNotifier* with the *acceptAndOpen()* method to wait for masters to discover this node and connect to it. The parameter used in the *Connector.open()* method is:

slaveURL="btspp://localhost: [BLEN_UUID] ;master=false"

It uses a SPP profile to accept and open a RFCOMM connection. The "*BIEN_UUID*" part of the URL shown above is used to set up a service for BIEN on a slave (i.e., server). The parameter "*master=false*" is used to request that a node trying to connect to this slave node must be the master in this connection. Once it receives a connection from a master it immediately exits from the inquiry scan mode by making itself undiscoverable to other masters. It may, however, re-enter the inquiry scan state at a later point to wait for other masters if it receives an instruction from the master to be ready to act as a bridge.

The execution procedure of the master application and the slave application can be separately summarized by the following flow charts (Shown in Figure 5.3).

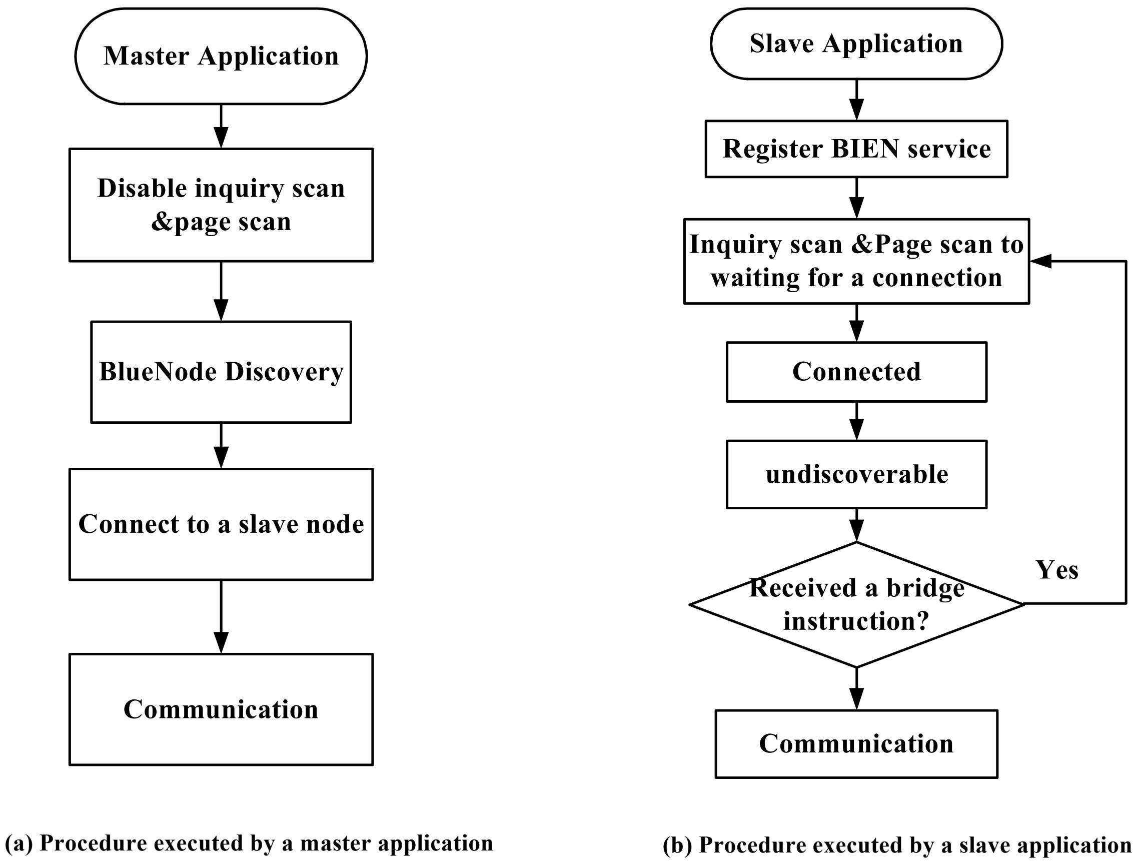

Master Application
Disable inquiry scan &page scan
BlueNode Discovery
Connect to a slave node
Communication
Slave Application
Register BIEN service
Inquiry scan &Page scan to waiting for a connection
Connected
undiscoverable
Received a bridge instruction?
Yes
Communication
(a) Procedure executed by a master application
(b) Procedure executed by a slave application

Figure 5.3 Execution procedures of master and slave applications

5.2.2.2 Connection construction

As master and slave applications are implemented in separate threads, they allow a BlueNode to be capable of the role of master, slave, M/S bridge or S/S bridge to connect other BlueNodes to form scatternets, by means of the multiple threading techniques of Java technology. This section explains how the application performs connections between BlueNodes to form scatternets.

- Master-Slave connection

If a BlueNode selects the role of master and tries to connect to a node performing the slave role, a thread of a new master application object is created in this master node, which makes a connection request to the slave node. On the other hand, if a BlueNode decides to be a slave, a thread of a slave application object is created and runs in the slave node to wait for connections from masters. When the slave thread of the slave node receives a connection from a master application thread in a master node, both nodes connect together as a slave and master respectively. Figure 5.4 shows a connection set up between two BlueNodes, in which one is a master application thread running in a master node, (i.e., BlueNode_1), and the other is a slave application thread running in a slave node, (i.e., BlueNode_2). These two nodes connect together to form a piconet in which a master node has only one slave node.

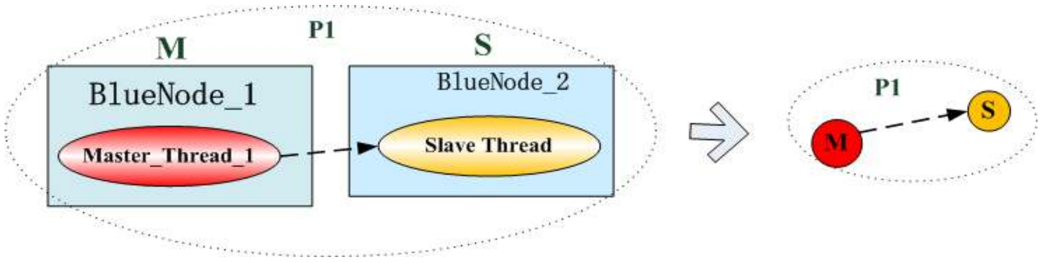


Figure 5.4 Implementation of master-slave connection

Every time a master node wants to connect to a new slave node, the application will create a new master application thread to connect to the new slave node, so that there are multiple master threads running in the master node, with the number of master threads being equal to the number of the slave nodes. Each one is connected to the slave application thread running in one of its slave nodes. Thereby, a piconet with one master node and multiple slave nodes has been formed. Figure 5.5 shows an example of how the application implements a piconet with a master node (BlueNode_1) connecting three slave nodes (i.e., the BlueNode_2, the BlueNode_3 and the BlueNode_4). There are three master application threads running in the master node, each one of them connecting to the slave application thread in a slave node.

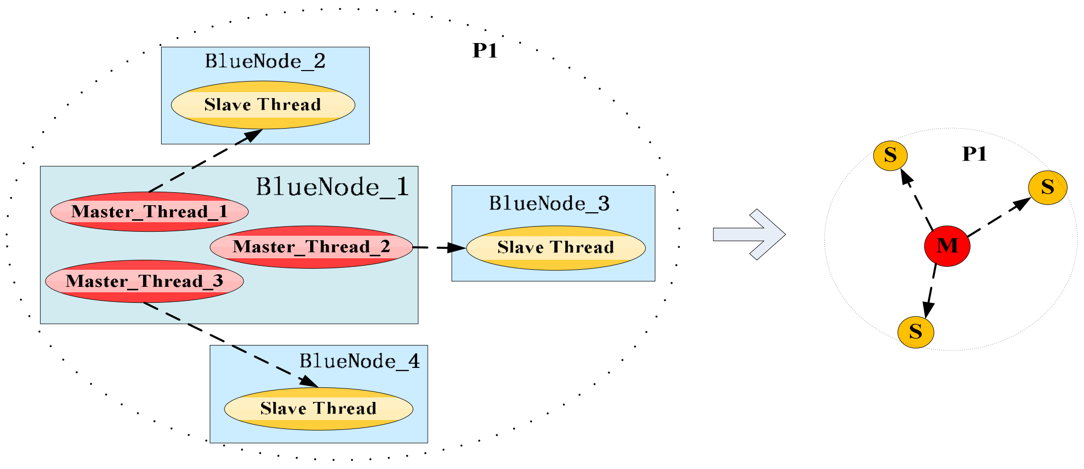


Figure 5.5 Example of the implementation of a piconet with three slaves

- Piconet connections

If the master node of a piconet wants to connect to one of the slaves of another master, it just creates a new master application thread to connect to the slave application thread in the slave node, as discussed above. At the same time, if the slave node has received an instruction from its master to be a bridge, the slave application thread running in it makes it discoverable and waits for a new connection from other masters. This is possible as the slave application is a server application, which allows the node to accept and open more than one connection from multiple Bluetooth clients. Once the slave application thread receives a connection from another master, the slave node has become a bridge for its two masters. Here the two piconets have joined together via a S/S bridge to form a scatternet (See Figure 5.6). In Figure 5.6, master node M2 (BlueNode_5) has connected to the slave node (BlueNode_3) of the master node M1 (BlueNode_1). As a result, the slave node (BlueNode_3) is a S/S bridge for both masters.

If a slave application receives more than one connection from different masters of the multiple piconets, it has become a multiple S/S bridge, by which all the masters can interconnect these multiple piconets.

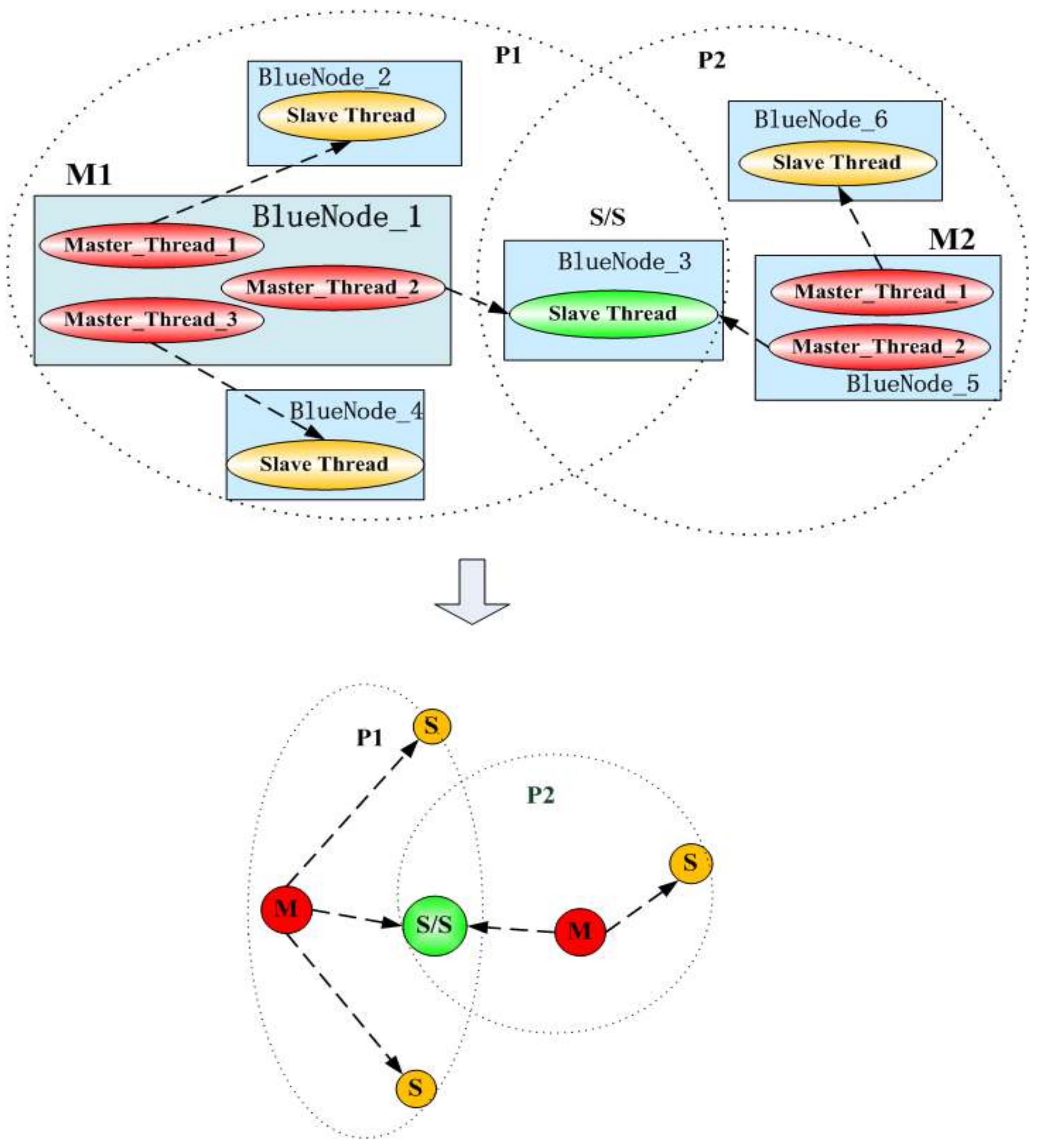


Figure 5.6 Application implementation of piconets connection via a S/S bridge

If the master node of a piconet wants to be a slave in other piconets, it will create a slave application thread running with the master application threads in the node. The slave thread will make the node enter an inquiry scan and page scan mode to wait for discovery by and connection with other master nodes. Once it receives a connection from a master, it has become a slave of the master node in the other piconet but at the same time it is still the master in its own piconet. This means that it has become a M/S bridge via which the two piconets are connected together to form a scatternet (See Figure 5.7). In Figure 5.7, there are multiple master application threads and a slave application thread running in BlueNode_1, so BlueNode_1 is the master of BlueNode_2, BlueNode_3, and BlueNode_4 in Piconet_1. At the same time, it is also a slave of a master node (BlueNode_5) of Piconet_2. Hence, BlueNode_1 is a M/S bridge joining the two piconets together.

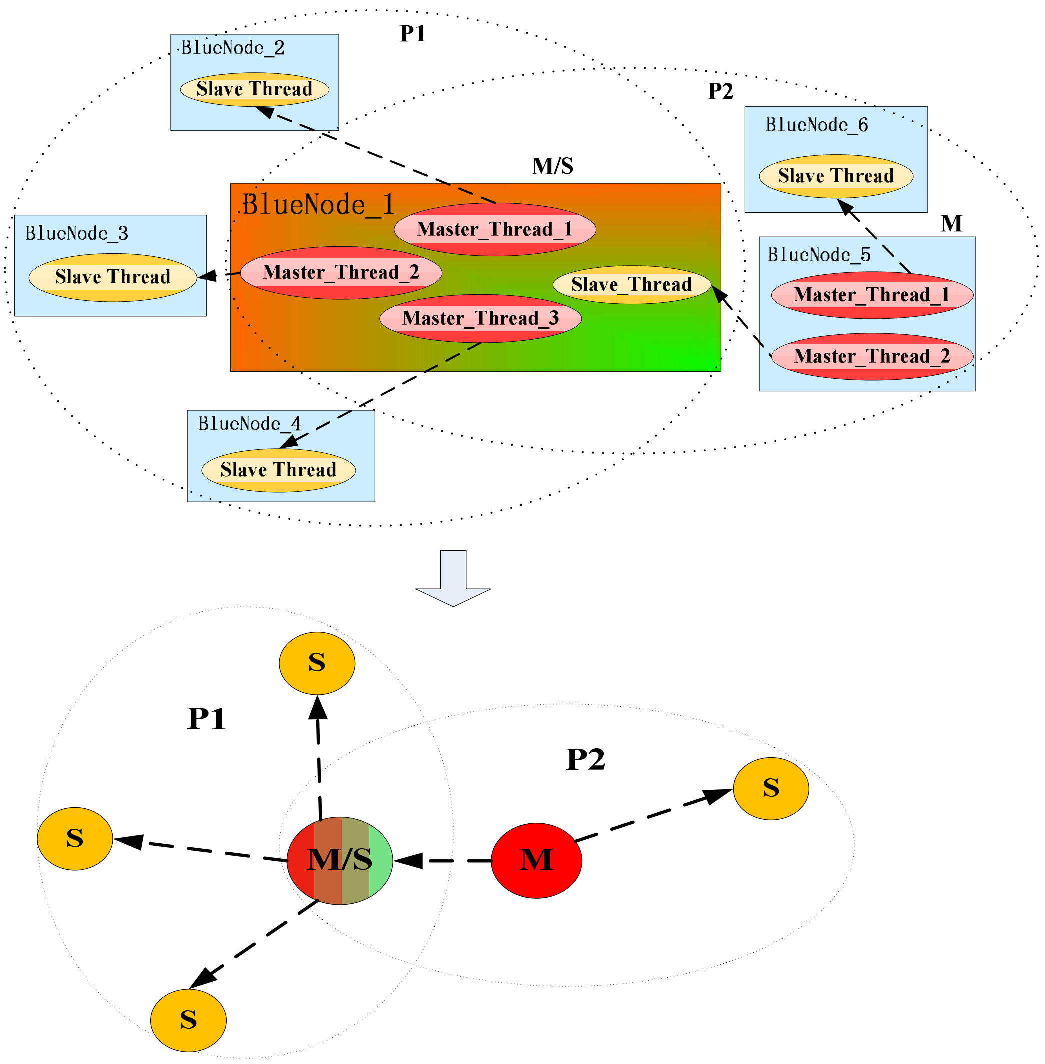


Figure 5.7 Application implementation of piconet connection via a M/S bridge

5.2.3 Communication management

5.2.3.1 Messaging system

The structure of the messaging system implemented for communication in the BIEN network is shown by a UML class diagram in Figure 5.8. The basic pattern of a BIEN message contains the fields of source address and destination address (See Figure 5.9). They ensure that the sending node and the destination node of the message are addressed while the content data it carries depends on the type of message.Two types of messages are adopted for communication in the BIEN network system, namely *Control Messages* and *Application Messages*.

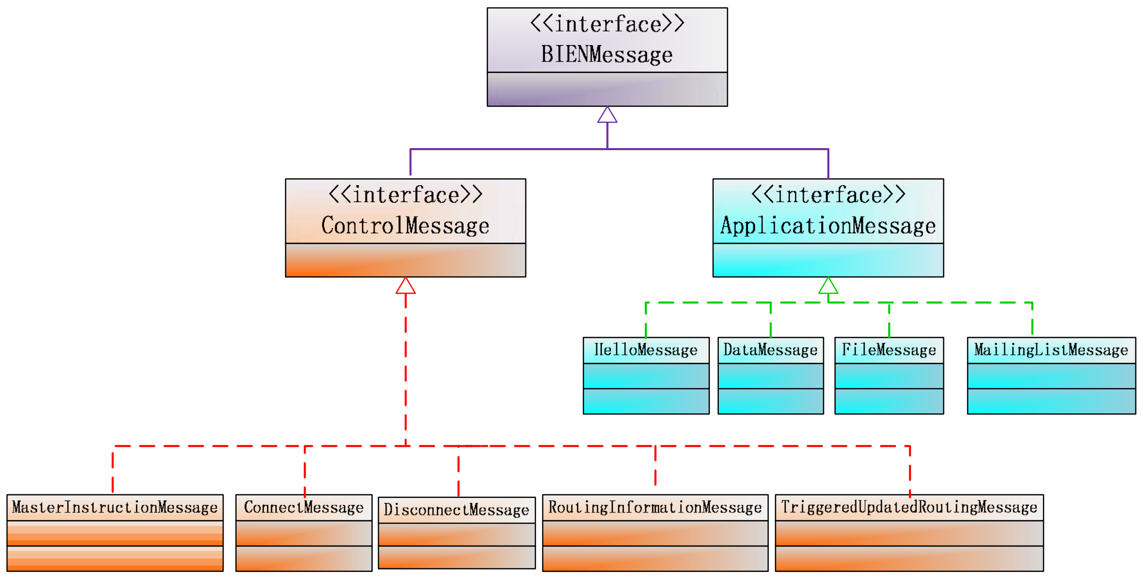


Figure 5.8 UML class diagram of BIEN messaging system

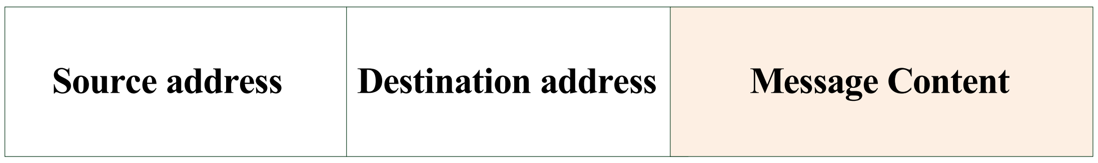


Figure 5.9 Basic BIEN Message pattern

A *Control Message* is used for carrying messages of routing information from the layers of the BIEN network formation and the RIP routing mechanize between the nodes for the purpose of network construction and maintenance. Since the routing information is only exchanged between a local node and its neighbors in the RIP routing protocol, each *Control Message* must be either from the local node to its neighbors or from its neighbors to the local node. It includes the messages types, *ConnectMessage*, *MasterInstructionMessage*, *RoutingInformationMessage,* *TriggeredUpdatedRoutingMessage* and *DisconnectMessage*, which are described as follows:

- *ConnectMessage* is only used by a master node when it has connected to a slave node. It is used to inform the slave of master information;
- *MasterInstructionMessage* is use by master nodes instructing some of their slaves to be ready to be bridges. Once a slave receives this message from its master, it makes itself discoverable, and waits to being discovered and connected to by other masters. Hence, this message is only used by master nodes;
- *RoutingInformationMessage* carries a message of routing information for regular routing table exchange with neighbouring nodes;
- *TriggeredUpdatedRoutingMessage* is used for BlueNodes sending updated routing information to their neighbors when a change on the network topology, such as a node joining or leaving the network, is detected, which triggers routing updating and exchange; and
- *DisconnectMessage* carries messages from nodes leaving the network for a purposeful disconnection of a node. As Java application can only detect a connection loss by a failing read or write operation, there is no immediate way to detect a connection loss. In the BIEN application, the approach adopted for a node that intended to disconnect from the network is to send a *DisconnectMessage* to its immediate neighbors to inform other nodes of its departure, so that they can update their routing tables and trigger routing information updating in their neighbors by sending the *TriggeredUpdatedMessage.*

The *Application Messages* carry user application data from the application layer of the BIEN network for use in information exchange between nodes, whereby:

- *FileMessage* carries different patterns of files, such as *text* files, *PDF* files and so on, images, such as *jpg*, *png* and *gif* images, and music files, such as mp3 files and so on;
- *DataMessage* is used to carry *Strings* that represent data when users chat by text;
- *HelloMessage* is used by the system to inform the sender of a received message when the destination that the message is forwarded to is unreachable and the message is undeliverable; and
- *MailingListMessage* is used for exchanging user information in the network, including their friendly names and Bluetooth addresses.

5.2.3.2 Communication and message process

Every connection that a BlueNode receives is processed in a separate thread. A *BlueConnector.java* class is implemented to deal with communication with one immediate neighbour node. Every time a BlueNode, as a master, is connected to a new BlueNode, as a slave, the master or the slave application manages the incoming connection by creating a new *BlueConnector* object that holds the new connection. Each *BlueConnector* uses the related connection that it holds to independently deal with all communication, including message sending and receiving using an *ObjectOutputStream* or *ObjecInputStream* object respectively, between the local node and its neighbour node that are at each end of the connection.

The two types of messages, the *Control Message* and the *Application Message*, are processed in two separate threads. This allows for fast routing information updating and exchange in the network. The two types of messages are separately held by two synchronized *Message Queues* for processing. All messages, whether they are outgoing from the local node or received by all *BlueConnector*s from neighbouring nodes, have to go into these two *Queues* according to their type before being processed. Two *Message Processors* are implemented to respectively process the queued messages. Both message processors can concurrently access the synchronized *BlueConnector* objects in order to send messages.

Each processor is essentially a state machine that processes the queued messages (See Figures 5.10 and 5.11). For each message type, each of them first checks whether the local node is the destination of the message or not. If it is, they will perform relevant message processes in regards to the message content and type. Otherwise, the *Application Message Processor* will look for a route to the destination in the routing table for forwarding the message. If there is a route available in the routing table, the processor simply forwards the message to the next hop; otherwise, it will discard the message and send a *HelloMessage* back to the sender of the message to inform the sender that the message is undelivered. In the case of the *Control Message Processor*, if the destination of the message is not a local node, the message must be sent to one of the neighboring nodes from the local node. Hence, it just gets the *BlueConnector* object related to the destination of the message and simply uses it to send the message to the neighbouring node.

The following flow charts summarize the operations of the two *Message Processors* when they process each queued message (See Figures 5.10 and 5.11). Whenever the

BlueNode application starts running, these two *Message Processors* start their message process and run it until the application is terminated.

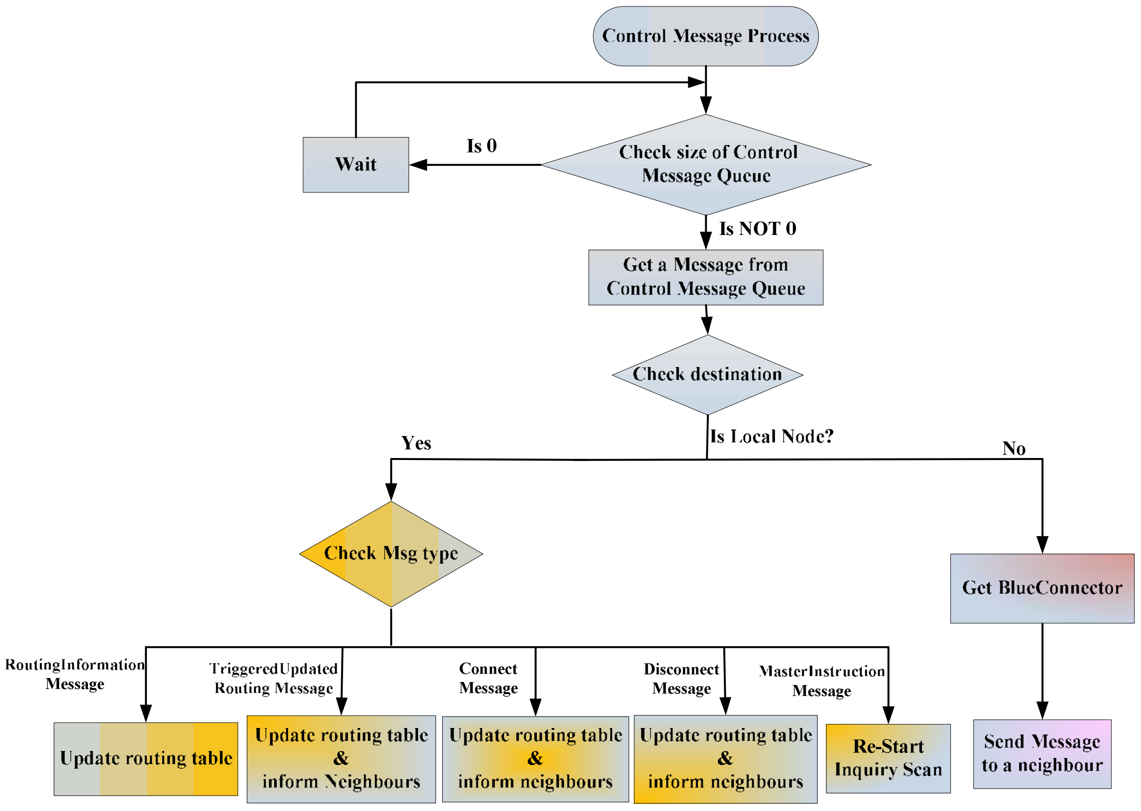

Control Message Process
Wait
Is 0
Check size of Control Message Queue
Is NOT 0
Get a Message from Control Message Queue
Check destination
Is Local Node?
Yes
No
Check Msg type
Get BlueConnector
RoutingInformation Message
TriggeredUpdated Routing Message
Connect Message
Disconnect Message
MasterInstruction Message
Update routing table
Update routing table & inform Neighbours
Update routing table & inform neighbours
Update routing table & inform neighbours
Re-Start Inquiry Scan
Send Message to a neighbour

Figure 5.10 Control Message process

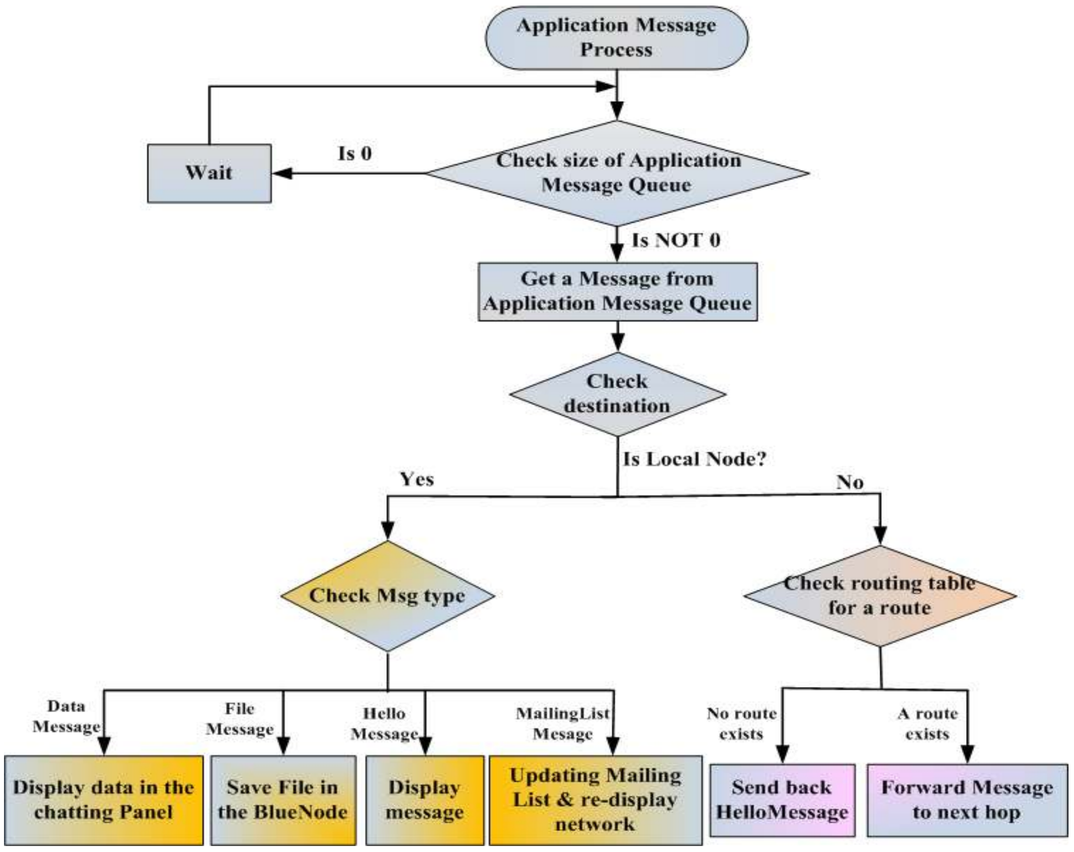

Application Message Process
Wait
Is 0
Check size of Application Message Queue
Is NOT 0
Get a Message from Application Message Queue
Check destination
Is Local Node?
Yes
No
Check Msg type
Check routing table for a route
Data Message
File Message
Hello Message
MailingList Mesage
No route exists
A route exists
Display data in the chatting Panel
Save File in the BlueNode
Display message
Updating Mailing List & re-display network
Send back HelloMessage
Forward Message to next hop

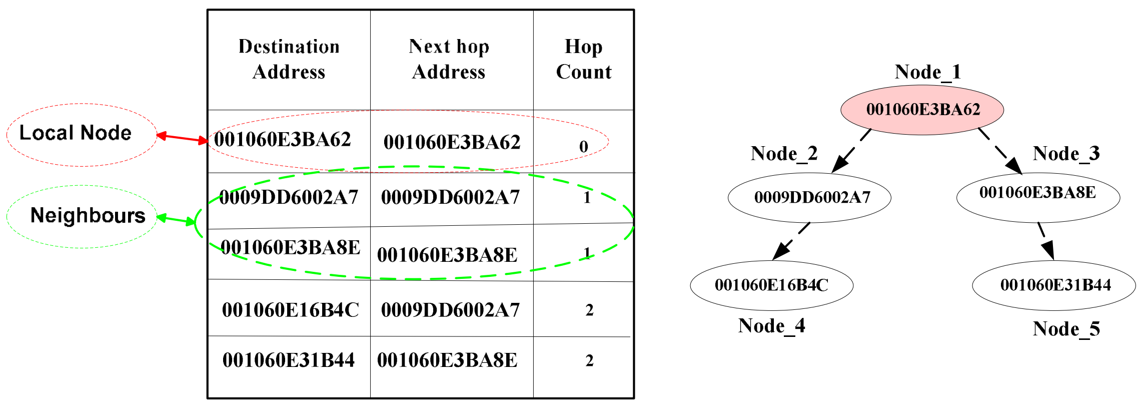
Figure 5.11 Application Message process

5.3 RIP routing protocol implementation application

This is a software implementation of the RIP routing mechanism and runs on the top of the network formation application. It provides an effective routing mechanism for the BIEN network. The main functions of this application are implementation of a routing table, routing computation, and routing table updates, based on the RIP routing protocol.

5.3.1 Implementation of the routing table

Each BlueNode holds and maintains its own routing table. The routing table implemented in the application contains routing information to every other BlueNode in the BIEN network. The unique Bluetooth MAC address of each Bluetooth device is used to identify different BlueNodes in the network. Each entry in the routing table includes the Bluetooth address of a destination node, the number of hops to that destination, and the Bluetooth address of the next hop node via which messages can be forwarded to the destination along the shortest path (See Figure 5.12).



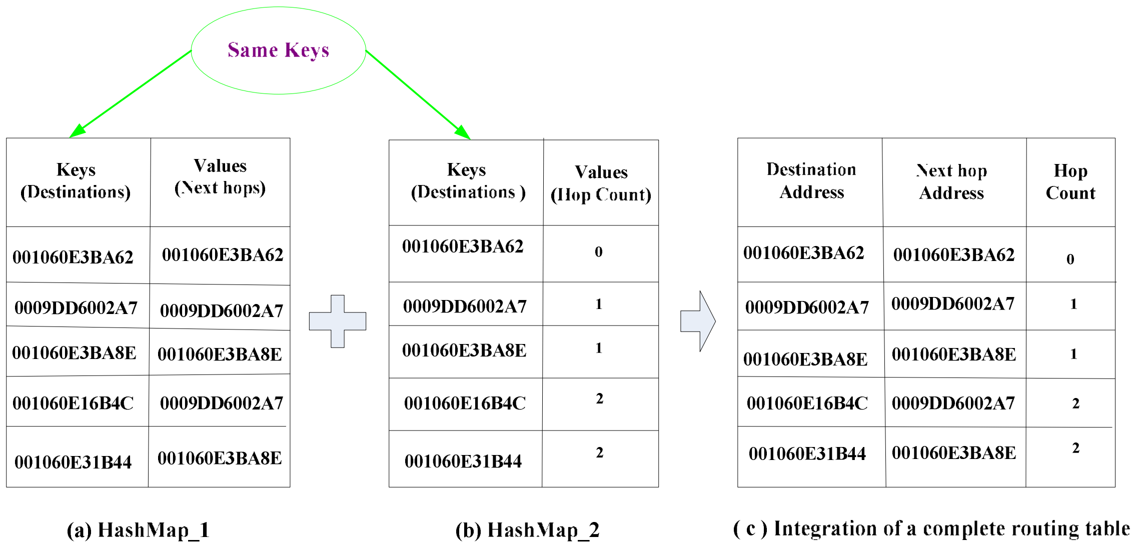
Destination Address	Next hop Address	Hop Count
001060E3BA62	001060E3BA62	0
0009DD6002A7	0009DD6002A7	1
001060E3BA8E	001060E3BA8E	1
001060E16B4C	0009DD6002A7	2
001060E31B44	001060E3BA8E	2

(a) Routing table of network in (b) held by Node_1 (b) An example of network

Figure 5.12 Structure of a routing table

Figure 5.12(a) illustrates an example of a routing table held by Node_1, with an address of 001060E3BA62, in the network shown in Figure 5.12(b). The routing table includes all routes from the node to other nodes in the network. The first entry is a destination node, listing its address of 001060E3BA62, the address of the next hop node to the destination; which is the same as the destination address; and the hop count to the destination; which is 0. This shows that the destination is the local host node itself that is the owner of the routing table, Node_1. The second and third entries contain routes to two neighbours of the local node, i.e. Node_2 and Node_3. This is due to the next hops to these destinations are themselves and the hop count to them from the local node both being 1, respectively. The fourth and fifth entries, respectively, contain two routes to Node_4 (001060E16B4C) and Node_5 (001060E31B44) through the neighbouring nodes in the second entry, or third entry, with a hop count of 2.

The routing table is implemented by two *ConcurrentHashMap* structures. One holds the next hops to all destinations from a local node and the other holds the number of hops to those destinations. As both the *HashMaps* use the same Bluetooth address of destination nodes as keys of the *HashMap* structure, they can work together to form a complete routing table. Figure 5.13 illustrates an implementation of the routing table given as an example in Figure 5.12.



Keys (Destinations)	Values (Next hops)
001060E3BA62	001060E3BA62
0009DD6002A7	0009DD6002A7
001060E3BA8E	001060E3BA8E
001060E16B4C	0009DD6002A7
001060E31B44	001060E3BA8E

(a) HashMap_1

Keys (Destinations)	Values (Hop Count)
001060E3BA62	0
0009DD6002A7	1
001060E3BA8E	1
001060E16B4C	2
001060E31B44	2

(b) HashMap_2

Destination Address	Next hop Address	Hop Count
001060E3BA62	001060E3BA62	0
0009DD6002A7	0009DD6002A7	1
001060E3BA8E	001060E3BA8E	1
001060E16B4C	0009DD6002A7	2
001060E31B44	001060E3BA8E	2

(c) Integration of a complete routing table

Figure 5.13 Implementation of routing table

5.3.2 Routing Computation

5.3.2.1 Routing metric

Based on the RIP routing protocol, the metric used to measure a path length from the source to the destination is a hop count, which is the number of hops in the path between the source and the destination. Each hop in a path is assigned a constant value of 1, so the distance between a node and any one of its immediate neighbours is 1. For example, in Figure 5.14, the path length between master node M1 or M2 and each of its slaves is 1. The hop count value will increase by one every time a packet crosses a node from a source to its destination, so the hop count between two slaves in the same piconet; such as slave node S1 and bridge node S/S in piconet_1 in Figure 5.14; is 2. The path length from slave node S1 to another master node M2 in piconet_2 is 3.

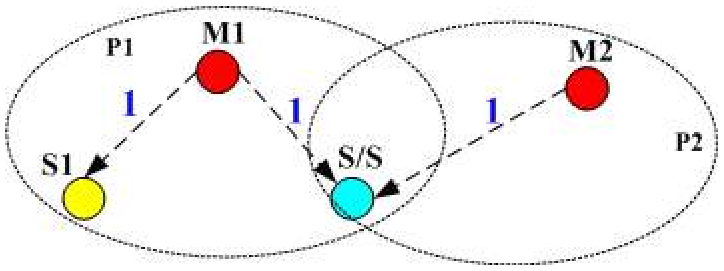


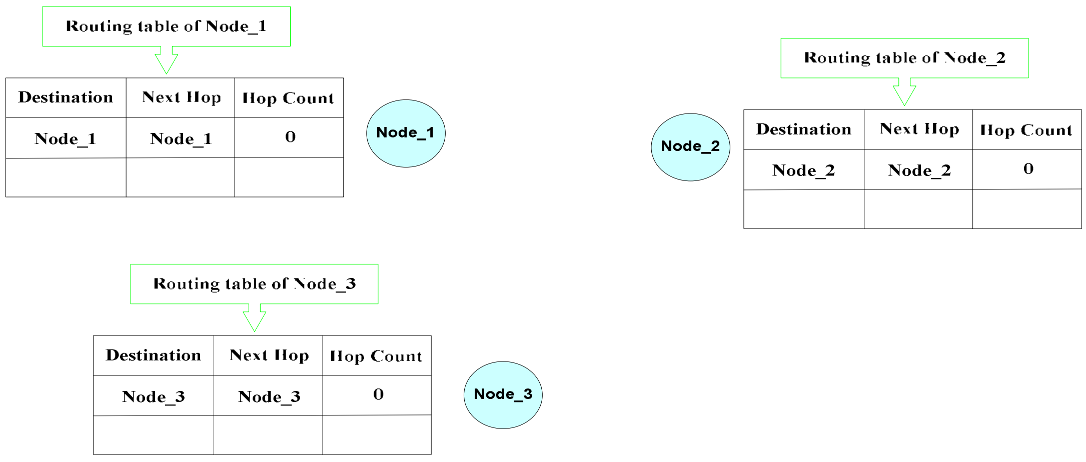
Figure 5.14 Example of a hop count computer

The maximum number of hops in a path is limited to 10 for the reasons discussed in the section 4.1.3. Any destination with a hop count value more than 10 is treated as infinite, which means the destination is unreachable.

5.3.2.2 Routing table computation

Based on the *Bellman-Ford* algorithm, a *DVRoutingTable.java* class is developed to perform the routing table computation in the application, which is described in the following:

- When a BlueNode first starts, its initial routing table contains only one entry. This is the destination, and the next hop node to the destination is itself, with the hop count to the destination being 0. Figure 5.15 illustrates an example of initial routing tables held, respectively, by three free BlueNodes; i.e., Node_1, Node_2 and Node_3. Each one has one entry of itself, as they have not connected together.



Routing table of Node_1

Destination	Next Hop	Hop Count
Node_1	Node_1	0

Routing table of Node_2

Destination	Next Hop	Hop Count
Node_2	Node_2	0

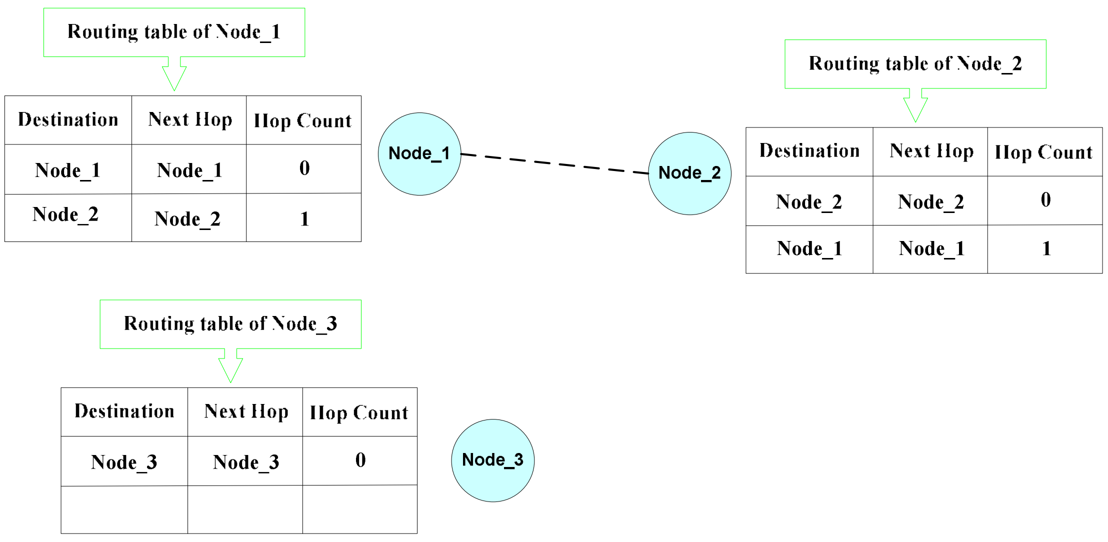
Routing table of Node_3

Destination	Next Hop	Hop Count
Node_3	Node_3	0

Figure 5.15 Example of initial routing tables of three free BlueNodes

- When it has connected with other nodes, new entries with these new immediate neighbours as the destinations will be inserted into the routing table. Each of the new entries uses the destination node as the next hop and has a hop number of 1. This is because these neighbours are directly connected to the node and there is only one hop from it to each of them. Thus, the next hops of the best path from the local node to these destinations are themselves.

For example, when Node_1 and Node_2 in Figure 5.14 have connected together, their routing tables have been updated by adding a new entry for the new immediate neighbour (See Figure 5.16(a)). The same process occurs on the routing tables of Node_1 and Node_3 when they connect together later (See Figure 5.16 (b)). The routing table of Node_1 shows that both Node_2 and Node_3 are its immediate neighbours.



Routing table of Node_1

Destination	Next Hop	Hop Count
Node_1	Node_1	0
Node_2	Node_2	1

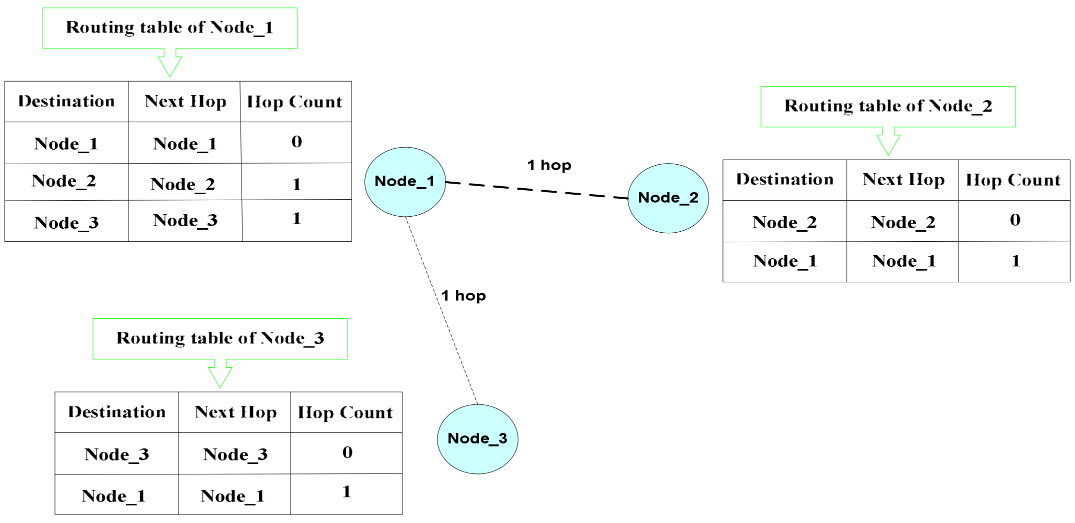
Routing table of Node_2

Destination	Next Hop	Hop Count
Node_2	Node_2	0
Node_1	Node_1	1

Routing table of Node_3

Destination	Next Hop	Hop Count
Node_3	Node_3	0

(a) Node_1 and Node_2 Connected together



Routing table of Node_1

Destination	Next Hop	Hop Count
Node_1	Node_1	0
Node_2	Node_2	1
Node_3	Node_3	1

Routing table of Node_2

Destination	Next Hop	Hop Count
Node_2	Node_2	0
Node_1	Node_1	1

Routing table of Node_3

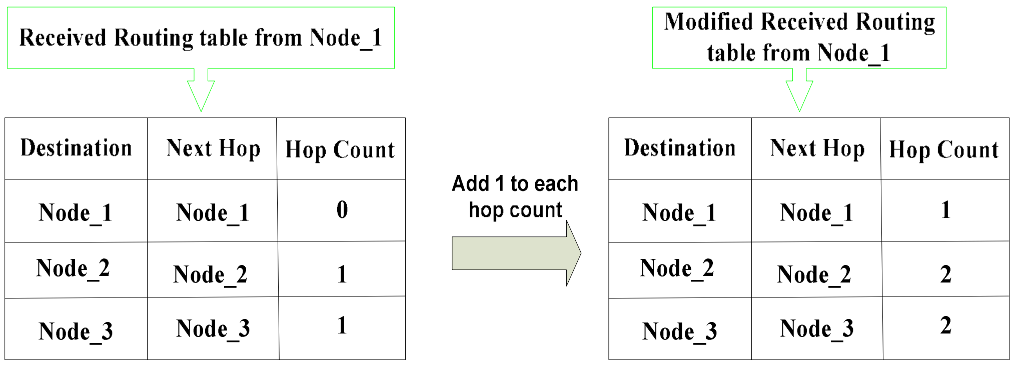
Destination	Next Hop	Hop Count
Node_3	Node_3	0
Node_1	Node_1	1

(b) Node_1 and Node_3 Connected together

Figure 5.16 Routing table computing when two nodes are connected together

- When the BlueNode receives a routing table from one of these neighbours, it first adds 1 to the hop count of each entry indicated in the received routing table to get a new length for each path from itself to the destination of the entry via the routing table sender, as the distance from itself to the sender is 1 hop.

For the same example discussed above, when Node_2 received a routing table from Node_1 in Figure 5.16 (b), it modified the received routing table by adding 1 to the hop count of each entry indicated in the received routing table, in order to get a new length of a path from itself to all destinations. Figure 5.17 shows the process of modifying the routing table received from Node_1.


Received Routing table from Node_1
Add 1 to each hop count
Modified Received Routing table from Node_1

Destination	Next Hop	Hop Count
Node_1	Node_1	0
Node_2	Node_2	1
Node_3	Node_3	1

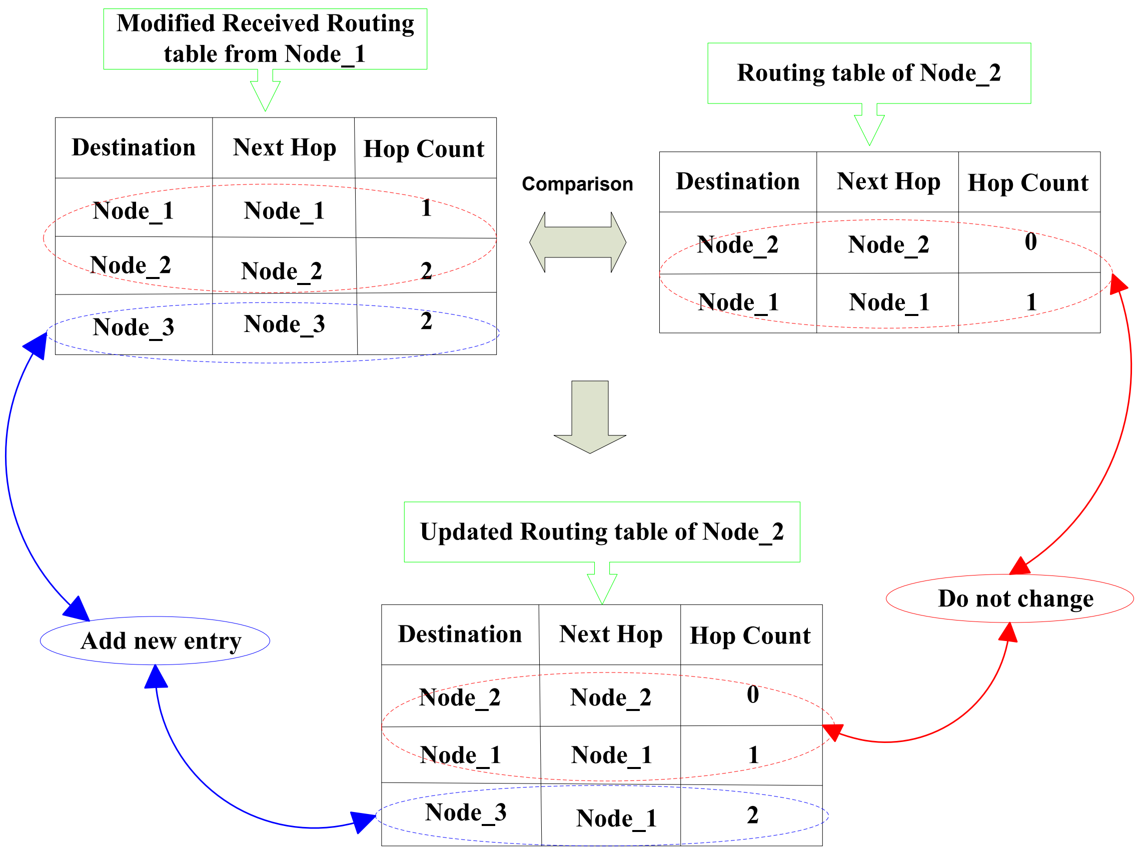
Destination	Next Hop	Hop Count
Node_1	Node_1	1
Node_2	Node_2	2
Node_3	Node_3	2

Figure 5.17 Node_2 modifies the routing table received from Node_1

- It then compares the modified routing table with its own routing table to compute the best routes to all destinations in the network in order to update its routing table:
  - If the received routing table contains new entries to destinations that do not exist in its own one, it simply adds these new entries into its own routing table and marks the sender of the routing table as the next hop to those destinations for those entries (See the third entry of the updated routing table of Node_2 in Figure 5.18);
  - If the received routing table has an entry where a hop count to a destination is smaller than the existing value in its own routing table, it replaces the hop count in its own routing table with the smaller one and marks the sender of the received routing table as the next hop to the destination for this entry;
  - If an entry of a destination exists in both routing tables and the next hop to the destination in its own table is the sender of the received routing table, but the hop counts are different in the two routing tables, it uses the hop count value in the received routing table to replace the old one for the entry in its table; and,

- If the new hop count of an entry is more than 10, it will remove the entry for the destination from its routing table.

Figure 5.18 shows the process by which Node_2 computes routing in the example discussed above. Node_2 checks each entry in the received routing table and compares it with its own entries. The first two entries in the received table also exist in its own table but the hop counts are bigger than those in its own table, meaning that, currently, the paths to these two destinations in the two entries are better than those in the received table. Thus, it does not change the current path to the destinations in the two entries in its own routing table. The third entry of the received table, however, does not currently exist in its own table, so it inserts the new entry in its table and marks the sender, i.e., Node_1, as the next hop. It shows that the length of the best path from the local node (Node_2) to the destination (Node_3) via the next hop (Node_1) is 2.


Modified Received Routing table from Node_1
Comparison
Routing table of Node_2
Updated Routing table of Node_2
Add new entry
Do not change

Modified Received Routing table from Node_1

Destination	Next Hop	Hop Count
Node_1	Node_1	1
Node_2	Node_2	2
Node_3	Node_3	2

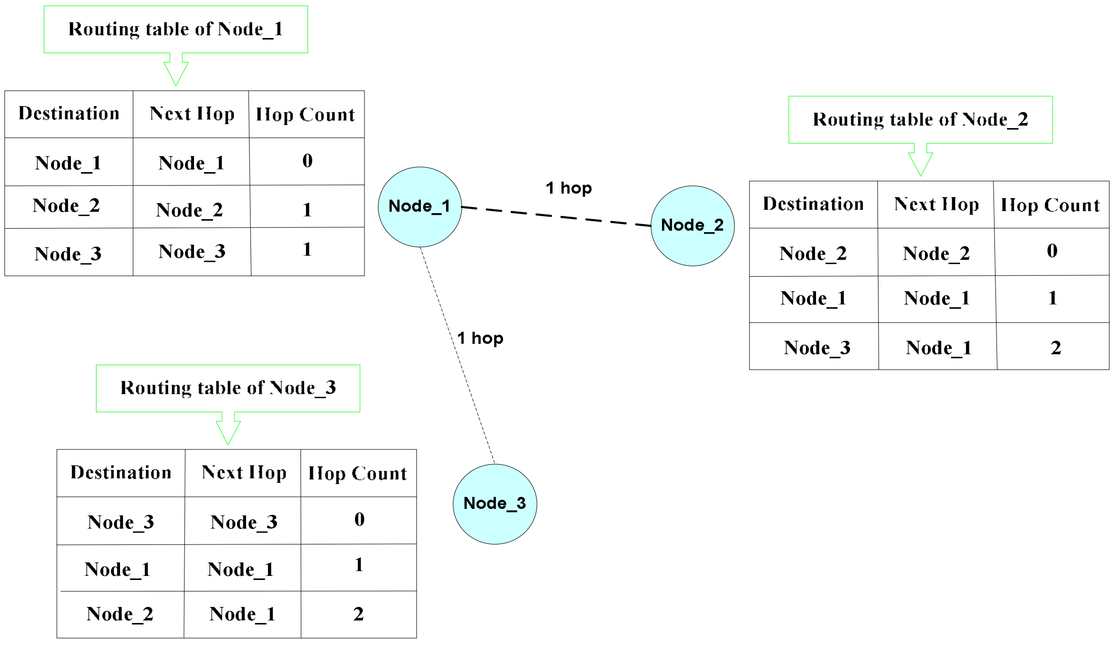
Routing table of Node_2

Destination	Next Hop	Hop Count
Node_2	Node_2	0
Node_1	Node_1	1

Updated Routing table of Node_2

Destination	Next Hop	Hop Count
Node_2	Node_2	0
Node_1	Node_1	1
Node_3	Node_1	2

Figure 5.18 Example of routing computation



Routing table of Node_1

Destination	Next Hop	Hop Count
Node_1	Node_1	0
Node_2	Node_2	1
Node_3	Node_3	1

Routing table of Node_2

Destination	Next Hop	Hop Count
Node_2	Node_2	0
Node_1	Node_1	1
Node_3	Node_1	2

Routing table of Node_3

Destination	Next Hop	Hop Count
Node_3	Node_3	0
Node_1	Node_1	1
Node_2	Node_1	2

Figure 5.19 Routing table held by each node after routing table exchange in the example given above

At the same time, the other nodes i.e., Node_1 and Node_3 in the example; also use the same procedure to calculate routings when they received routing messages from their neighbours. Eventually, each of the nodes knows the best routes to each other node in the network (See Figure 5.19).

5.3.3 Routing update

A *Timer* is implemented to regulate the routing updating operation. The interval between periodic routing updates is set to 30 seconds so that a BlueNode may send its full routing table by a *RoutingInformationMessage* to all of its immediate neighbours every 30 seconds and update its routing table based on the routing information received from those neighbours. Whenever the BlueNode application begins running, the routing table update process runs until users terminate the application. Since each BlueNode starts running at a different time, this does not result in traffic congestion caused by all of the nodes simultaneously attempting to send routing information to their neighbours.

The application also implements the split horizon and triggered updates mechanism to perform instant routing updates whenever any changes are detected on the network. Hence, when a BlueNode has detected any changes on the network; including receiving a *ConnectMessage* or a *DisconnectMessage* informing it of a neighbour joining or leaving the network, or detecting a connection loss whenever a *BlueConnector* fails a read or write operation; it updates its routing table and immediately sends the table via a

TriggeredUpdatedRoutingMessage to all of its neighbours to inform them of the change. On the other hand, when a BlueNode receives a *TriggeredUpdatedRoutingMessage,* it checks the received routing table to determine whether there are any differences from the one it currently holds. If there are, it updates its routing table based on the received routing information and immediately sends another *TriggeredUpdatedRoutingMessage* to all its neighbours, with the exception of the sender of the original *TriggeredUpdatedRoutingMessage*. This is the adoption of the split horizon mechanism, which reduces the probability of routing loops, so as to prevent incorrect routing information from being propagated.

This triggered update is separate from the regular routing table updates. As it is performed immediately after any change of a routing table entry is detected, and there is no need to wait until the next regular update, it is helpful in spreading changes of topology throughout the network faster than would otherwise be possible through the regular update cycle.

In addition to the split horizon and triggered update mechanisms, the application also limits the maximum number of hops allowed in a path to 10, based on the Bluetooth ad hoc environment. Any destination entry with a hop count of more than 10 is treated as infinite and will be removed from a routing table, which prevents routing loops from continuing indefinitely, and accelerates network convergence. This stability feature works with split horizon and triggered update mechanisms together and can quickly propagate routing updates throughout the network. This has greatly improved the stability of the BIEN network.

Whenever the BIEN application begins to run, the routing table update process runs until users terminate the application. This routing mechanism is deployed in each BlueNode, so that each node can act as a router to determine and maintain routes to all possible destinations in the BIEN network, no matter what role it undertakes.

More detailed information about the RIP routing protocol and its stability features can be found in [44].

5.4 BIEN function and GUI implementation application

The application is implemented at the application layer of the Bluetooth Protocol Stack to provide the information exchange function for BIEN users, when exchanging data with anyone else in the network. A BIEN system graphical user interface (GUI) is also developed for users to perform BIEN formation operations and access the information exchange function.

5.4.1 Implementation of the information exchange function

The information exchange function includes the file transfer service and communication by text. The files that can be transferred include most types of files, including text, document, script, image, and music files.

In the implementation of the file transfer service, a file is transferred by a type of *byte array* carried by a *FileMessage*. In addition to a source address and a destination address, a *FileMessage* object also has two another fields, namely a *File name,* which is a *String* object that holds the name of the transferred file, and a *File data bytes,* which is a *byte array* object that holds the data bytes of the file (See Figure 5.20). Each time a BlueNode wants to send a file to another node in the network, the application first converts the file to bytes in a *byte array* using a *FileInputStream* object. Then it creates a new *FileMessage* object that carries the name and the data bytes of the file and puts the message into the *Application Queue* for sending.

Source address	**Destination address**	**File name**	**File data bytes**

Figure 5.20 Format of a *FileMessage* object

At the other end, when the destination node receives the *FileMessage*, the application takes the file name and data bytes parts off the message and uses a *FileOutputStream* object to convert the data bytes to a file with the same name.

When communicating by text, the application creates a *DataMessage* object (See Figure 5.21) for each text string that users enter in the *EnterMessage* panel of the GUI (Shown in Figure 5.22), and then puts the message into the *Application Queue* for sending. Once the destination node receives the *DataMessage*, it simply takes the text string from the message and displays the string in the *Conversations* panel of its BIEN system GUI (See Figure 5.22).

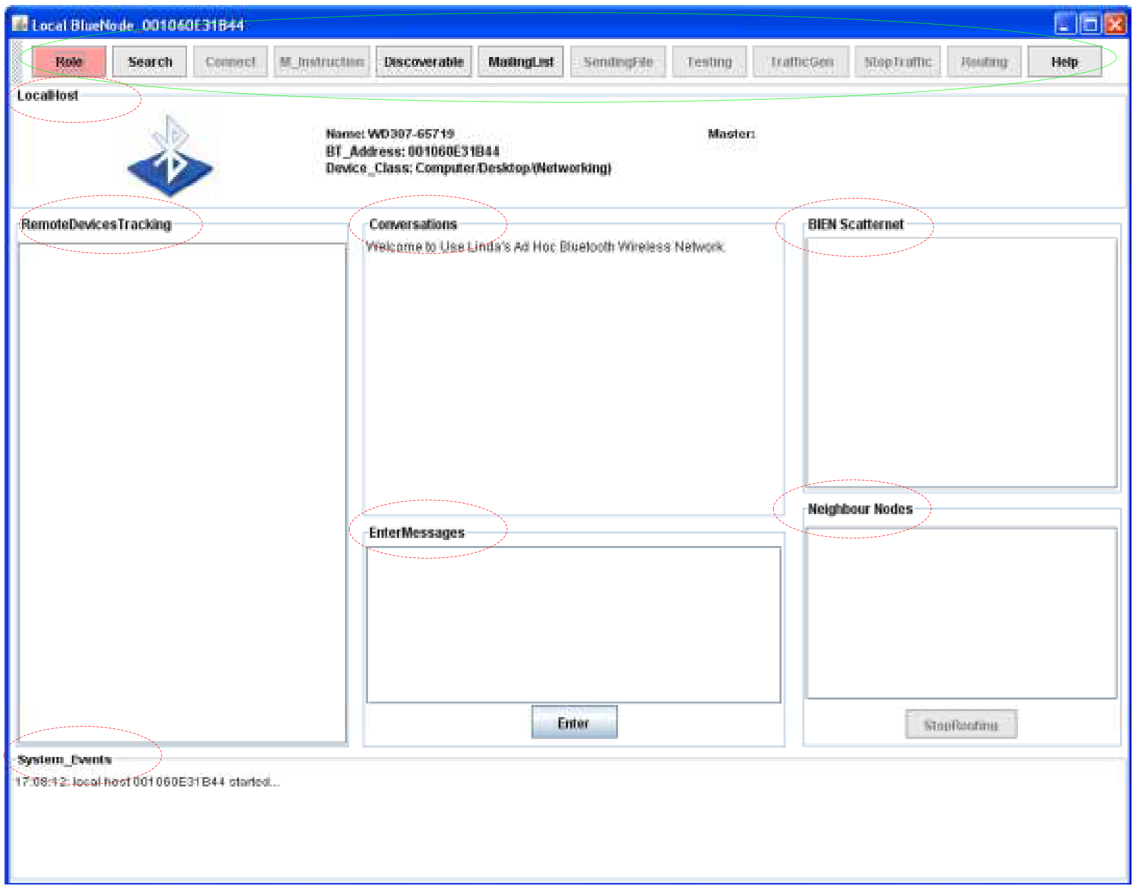
Source address	**Destination address**	**Text String**

Figure 5.21 Format of a *DataMessage* object

5.4.2 Implementation of BIEN system GUI

The GUI allows the BIEN users to perform operations of role selection, in terms of being a master or slave before joining a BIEN, connecting to other BlueNodes to join the BIEN, and accessing these information exchange services after joining a BIEN.

Figure 5.22 displays the main frame of the system GUI when the BIEN application starts running. The GUI consists of seven major display areas, labeled by the names with red circles. These panels are the *LocalHost* Panel, the *RemoteDevicesTracking* Panel, the *Neighbour Nodes* Panel, the *BIEN Scatternet* panel, the *EnterMessage* Panel, the *Conversations* Panel, and the *System Events* panel.



Name: WD307-65719
BT_Address: 001060E31B44
Device_Class: Computer/Desktop/(Networking)
Master:

Welcome to Use Linda's Ad Hoc Bluetooth Wireless Network

17:08:12: local host 001060E31B44 started...

Figure 5.22 Frame of BIEN system GUI

The *LocalHost* panel displays the information regarding the local BlueNode, including the friendly name, Bluetooth address, and the device class of the Bluetooth device. The master node of this device after it has joined a BIEN network is also displayed here. The *LocalHost* panel in Figure 5.22 shows that the local node is a desktop, with the name WD 307-65719 and a Bluetooth address of 001060E31B44. It does not have master displayed, as it is a free node at that moment.

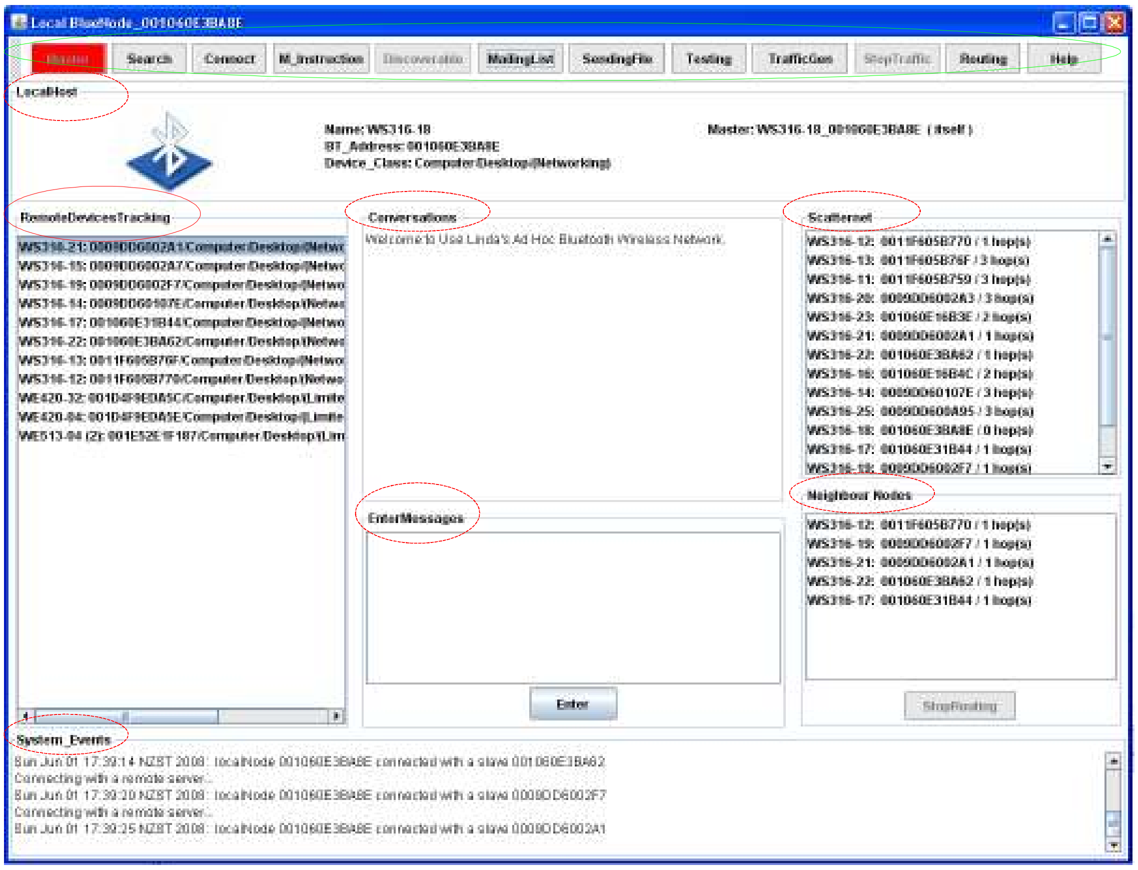
The *RemoteDevicesTracking* panel displays a list of the nearby BlueNodes that the local node has discovered after finishing device inquiry step.

When a BlueNode has connected with other BlueNodes, these nodes are its immediate neighbours and are listed in the *Neighbour Nodes* panel in which each one has a format of:

“Device name: Bluetooth address / hop count“

This shows the neighbour node’s device name, the Bluetooth address, and the hop count from the local node to the neighbour node; which is 1, as they are immediate neighbours. At the same time, because of the underlying routing updates with these neighbouring nodes, its routing table holds the routes to all other nodes in the network, which are listed in the *BIEN Scatternet* panel by device name, Bluetooth address, and hop count from the local node to the related node. All nodes are listed with the same format as those displayed in the *Neighbour Nodes* panel (See Figure 5.23).

Figure 5.23 is the system GUI of a master node, namely WS316-18 with the address 001060E3BA8E. There are five immediate neighbours listed in the *Neighbour Nodes* panel, which are the slave nodes of the node, as it is a master node. All nodes that are connected to the network at that moment are listed in the *BIEN Scatternet* panel. For example, there is a node WS 316-14 with an address of 0009DD60107E in the list, which shows that there are 3 hops for the path from the local master node to this particular node.



Name: WS316-18
BT_Address: 001060E3BA8E
Device_Class: Computer/Desktop/(Networking)

Master: WS316-18_001060E3BA8E (itself)

WS316-21: 0009DD6002A1/Computer/Desktop/(Netw
WS316-15: 0009DD6002A7/Computer/Desktop/(Netw
WS316-19: 0009DD6002F7/Computer/Desktop/(Netwo
WS316-14: 0009DD60107E/Computer/Desktop/(Netwo
WS316-17: 001060E31B44/Computer/Desktop/(Netwo
WS316-22: 001060E3BA62/Computer/Desktop/(Netwo
WS316-13: 0011F605B76F/Computer/Desktop/(Netwo
WS316-12: 0011F605B770/Computer/Desktop/(Netwo
WE420-32: 001D4F9EDA5C/Computer/Desktop/(Limite
WE420-04: 001D4F9EDA5E/Computer/Desktop/(Limite
WE513-04 (2): 001E52E1F187/Computer/Desktop/(Lim

Welcome to Use Linda's Ad Hoc Bluetooth Wireless Network.

WS316-12: 0011F605B770 / 1 hop(s)
WS316-13: 0011F605B76F / 3 hop(s)
WS316-11: 0011F605B759 / 3 hop(s)
WS316-20: 0009DD6002A3 / 3 hop(s)
WS316-23: 001060E16B3E / 2 hop(s)
WS316-21: 0009DD6002A1 / 1 hop(s)
WS316-22: 001060E3BA62 / 1 hop(s)
WS316-16: 001060E16B4C / 2 hop(s)
WS316-14: 0009DD60107E / 3 hop(s)
WS316-25: 0009DD600A95 / 3 hop(s)
WS316-18: 001060E3BA8E / 0 hop(s)
WS316-17: 001060E31B44 / 1 hop(s)
WS316-19: 0009DD6002F7 / 1 hop(s)

WS316-12: 0011F605B770 / 1 hop(s)
WS316-19: 0009DD6002F7 / 1 hop(s)
WS316-21: 0009DD6002A1 / 1 hop(s)
WS316-22: 001060E3BA62 / 1 hop(s)
WS316-17: 001060E31B44 / 1 hop(s)

```
Sun Jun 01 17:39:14 NZST 2008 : localNode 001060E3BA8E connected with a slave 001060E3BA62
Connecting with a remote server...
Sun Jun 01 17:39:20 NZST 2008 : localNode 001060E3BA8E connected with a slave 0009DD6002F7
Connecting with a remote server...
Sun Jun 01 17:39:25 NZST 2008 : localNode 001060E3BA8E connected with a slave 0009DD6002A1
```

Figure 5.23 BIEN GUI of a master node

The *EnterMessages* Panel allows BIEN users to type text in when they communicate via the text service to communicate with anyone else in the network. When a user selects a node from the BlueNode list of the *BIEN Scatternet* panel for message sending and presses the *Enter* button under the *EnterMessages* panel, the text that the user typed in will be sent to the selected node and displayed on the *Conversations* panel of both the sender and the receiver, with the address of the sender and the time of sending or receiving also displayed. This utility of talking via a text service is similar to the Windows Live Messenger that provides an Internet messaging service developed by Microsoft. It is, however, implemented for short range conversations using ad hoc Bluetooth wireless network technology without the need for cables or central administration, and the communication is free.

The *System Events* panel displays all system events related to the local BlueNode, such as the time when the node starts running, role selection, node connections and disconnections, application exceptions, and so on (See Figures 22 and 23).

There is also a tool bar on the top of the GUI, which has buttons by which users can perform the operations of a BIEN network formation in order to access this information exchange service.

When a BlueNode starts running, the initial BIEN system GUI displays on the screen (Refer to Figure 22). As a free node, all display areas are empty. The user can perform device discovery by pressing the *Search* button on the tool bar to look for nearby BlueNodes, or make it discoverable by other nodes by pressing the *Discoverable* button. The user, however, has to press the *Role* Button to display a role select panel (See Figure 5.24) in order to make a role selection, before the BlueNode can connect with other nodes to form or join a BIEN.

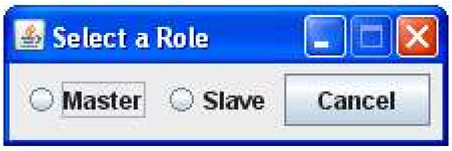
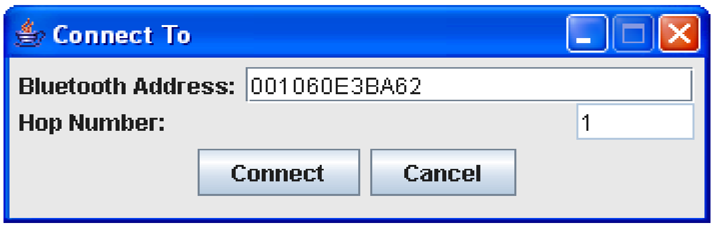


Figure 5.24 Role selection panel

If the BlueNode takes on the role of master, the user can press the *Search* button to find other BlueNodes to connect to. The user may then connect them one-by-one by selecting one at a time from the list of the discovered nodes in the *RemoteDevicesTracking* panel and pressing the *Connect* button. A connection panel will pop up, in which the address of the selected node and a default hop number of 1 is filled in (Shown in Figure 5.25).



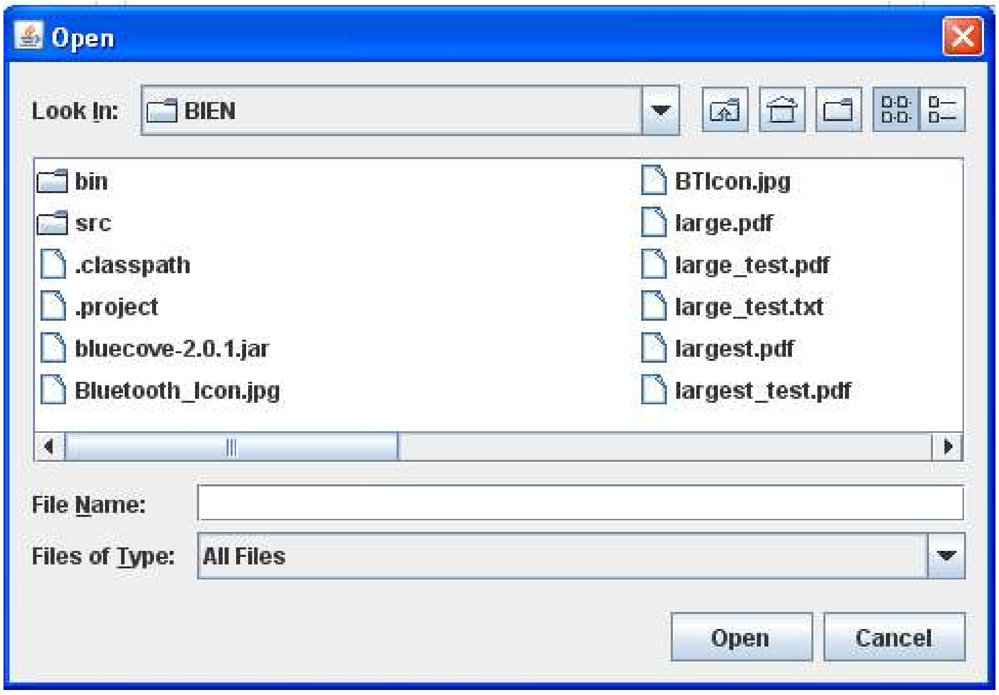
Bluetooth Address: 001060E3BA62
Hop Number: 1

Figure 5.25 Connection panel

By pressing the *Connect* button on the connection panel, the local master node will connect to the selected node automatically. The user can also type in the address of a selected BlueNode in the field *Bluetooth Address* in the connection panel to connect to this particular node.

By pressing the *M_Instruction* button, the master node can automatically send *MasterInstructionMessages* to some of its slaves, selected from the *Neighbour Nodes* panel. This will inform them to get ready to be a bridge and wait for new connections from other masters. This master node can continue to perform piconets' interconnections by finding slaves of other piconets and connecting to them.

When sending files, the user only needs to select a destination for the file send to from the *BIEN Scatternet* panel and press the *SendingFile* button to select a file from a *JFileChooser* dialog (shown in Figure 5.26), and then click the *Open* button on the dialog. The selected file will be automatically sent to the receiver.



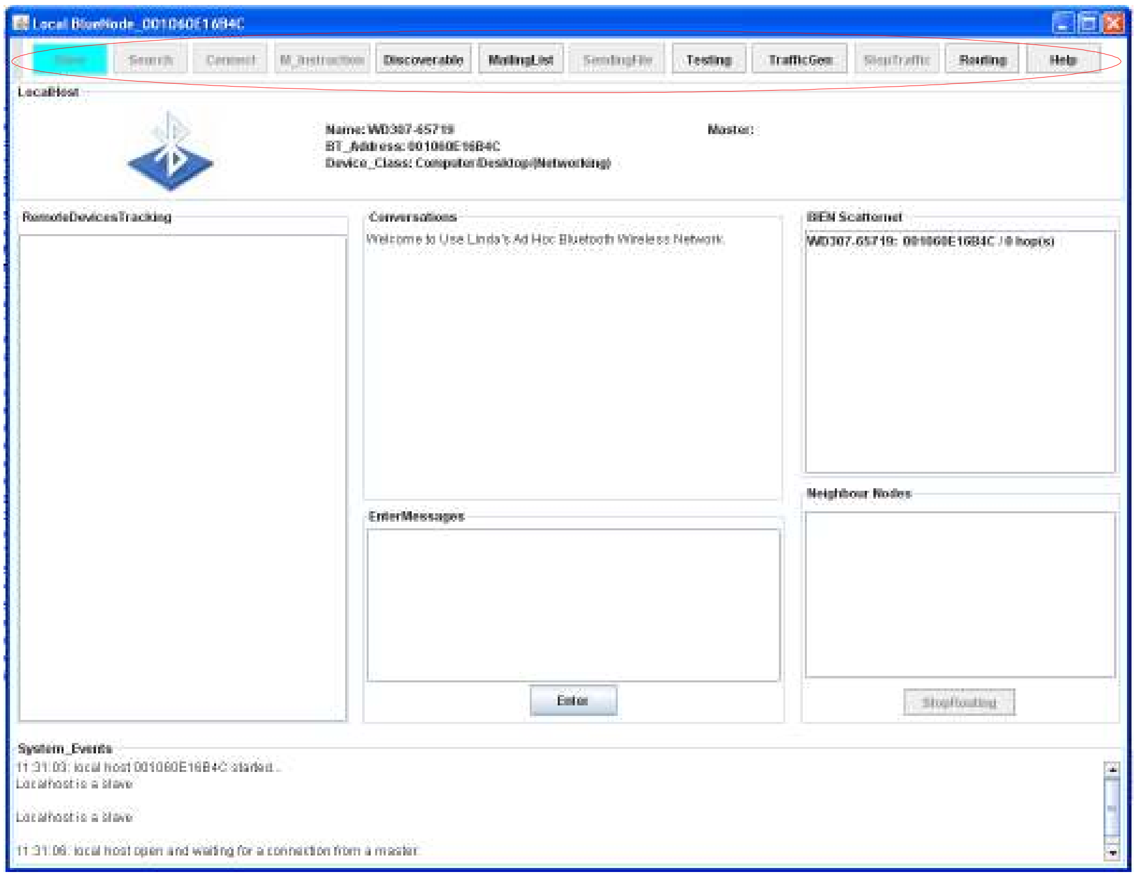
Look In: BIEN

bin
src
.classpath
.project
bluecove-2.0.1.jar
Bluetooth_Icon.jpg
BTIcon.jpg
large.pdf
large_test.pdf
large_test.txt
largest.pdf
largest_test.pdf

File Name:
Files of Type: All Files

Figure 5.26 File selection dialog

On the other hand, if the BlueNode selects to be a slave, it will automatically start an inquiry scan and a page scan to ensure its discovery by, and connection with, a master. The *Search* button, *Connect* button, and *M_instruction* button are disabled in the system GUI (See Figure 5.27). This is because they are only designed for use by masters on the basis of the BIEN formation protocol. The *SendFile* button will be enabled once the node has been connected by a master node.



Name: WD307-65719
BT_Address: 001060E16B4C
Device_Class: Computer/Desktop/(Networking)
Master:

Welcome to Use Linda's Ad Hoc Bluetooth Wireless Network.

WD307-65719: 001060E16B4C / 0 hop(s)

```
11:31:03: local host 001060E16B4C started...
Localhost is a slave

Localhost is a slave

11:31:06: local host open and waiting for a connection from a master.
```

Figure 5.27 BIEN GUI of a slave node

The following flow chart (Figure 5.28) summarizes the operation of a BlueNode for users. To summarize, Bluetooth devices configured with a set of these software applications can work as BlueNodes, with an identical ability to connect with other BlueNodes to form a BIEN of any topology and any size within aforementioned constrains .

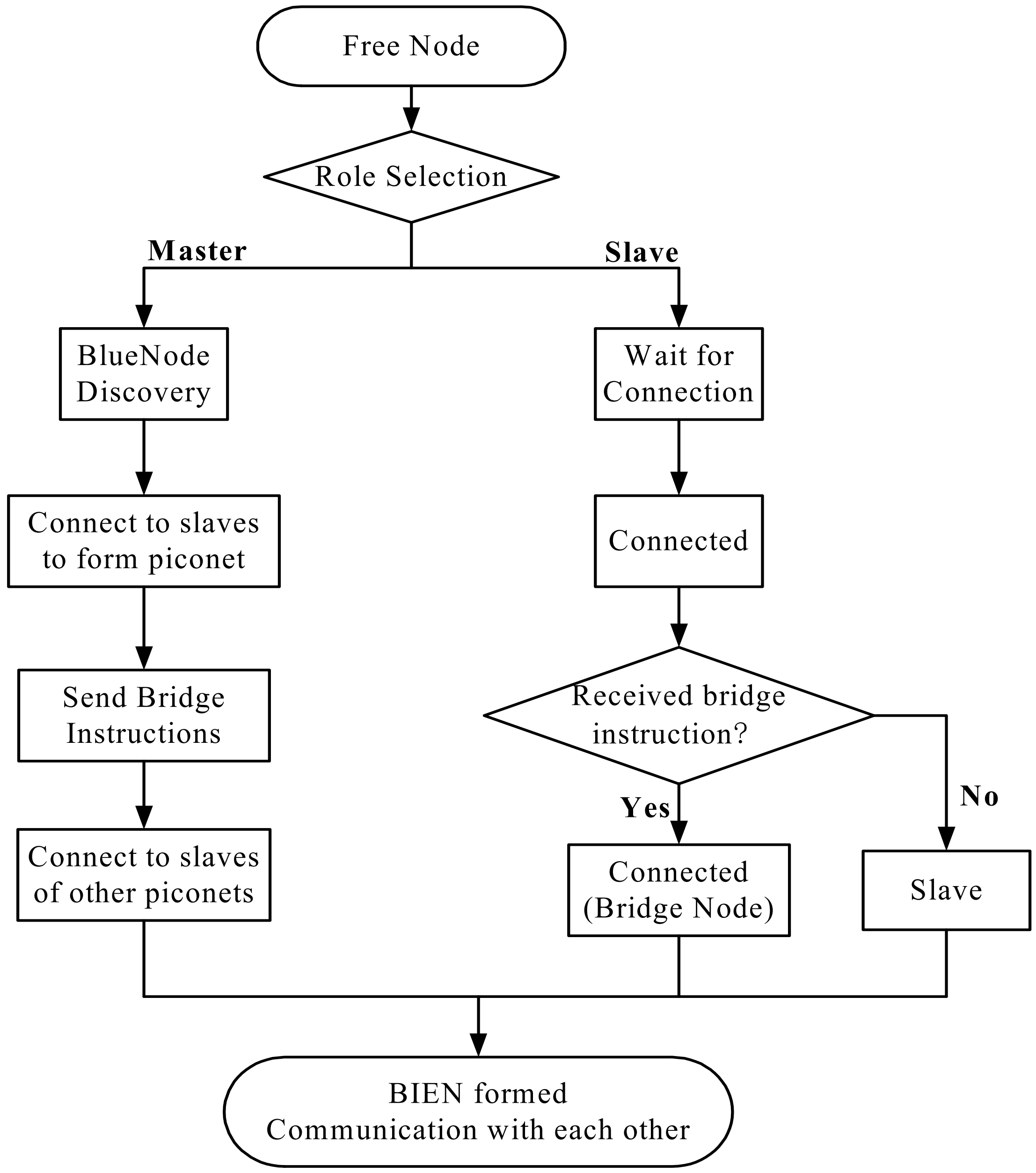

Free Node
Role Selection
Master
Slave
BlueNode Discovery
Wait for Connection
Connect to slaves to form piconet
Connected
Send Bridge Instructions
Received bridge instruction?
Yes
No
Connect to slaves of other piconets
Connected (Bridge Node)
Slave
BIEN formed Communication with each other

Figure 5.28 Operation procedure of a BlueNode

5.5 Summary

This chapter has discussed the BIEN software application implemented in this work, including the network formation application, RIP routing mechanism implementation, and the BIEN network application and GUI implementation application. A Bluetooth device configured with the BIEN software application can work as a BlueNode that has the ability to connect with other BlueNodes and compute routing to autonomously form a dynamic Bluetooth network for information exchange with other nodes in the network without the requirement of any network infrastructure or network administration.

6 Experiment and performance evaluation

This chapter presents the experiment deployment and experimental results, which were used to measure and evaluate the performance of the BIEN networks set up in this work.

First, the experimental methods used to test the BIEN software application developed and acquire the data in order to evaluate the performance of the BIEN networks set up in this work are described. This is followed by an evaluation of the network behaviour and performance. Finally, this chapter concludes with a short summary of the main points discussed in the chapter.

6.1 Experimental Deployment

This section describes the experiments deployed and conducted to construct BIENs and to evaluate their performance in a computer classroom at AUT University. These experiments were used to test the BIEN software application and confirm its correct behaviour, as well as to collect data to evaluate the performance of BIEN networks and investigate how well Bluetooth technology supports the ad hoc multi-hop wireless network technology.

6.1.1 Experimental methods

As a research project, the experiments deployed focus on evaluating the relation between network capacity and topology by testing end-to-end performance in terms of throughput and the latency of communication links with various parameters, including the hop number between nodes and the slave number in piconets. The following outlines the methods used to test the BIEN network capacity in this work.

6.1.1.1 Measurement metrics

The metrics used to test the developed application and evaluate the performance of the BIEN network are throughput and latency.

- Throughput

Throughput is used to measure the capacity that a path can provide to an application when there is no traffic load. Here it refers to the rate of data delivery over a path between two BlueNodes; i.e., end-to-end throughput. It is a measure of the amount of data bytes that are transferred between the two nodes over a period of time, in kilobits per second (Kbps).

The method used to perform the measurement was to continuously send files from a source node to the destination node and measure the time taken for each file to reach the destination. The throughput for a transferred file was obtained by dividing the file size by the time taken by the file to reach the destination (See Equation 6-1). The average throughput for all of the transferred files was calculated by dividing the sum of the bytes of all files by the time taken by these files to reach the destination (See Equation 6-2).

$$Throughput = \frac{S_T \times 8}{t_T \times 1000} \quad \text{(Kbps)} \qquad \textbf{(Equation 6.1)}$$

$$Average\_Throughput = \frac{\sum S_{TK} \times 8}{\sum t_{TK} \times 1000} \quad \text{(Kbps)} \qquad \textbf{(Equation 6.2)}$$

Where S_T is the size of a file transferred in bytes, t_T is the time taken to transfer the file, measured in seconds, and K is the sample number.

The measured time for a file transfer includes the time required for the sender to send the file, the time required for the file to pass through a certain path from the sender to a receiver across multiple hops, and the time required for the receiver to receive the file.

In fact, as the throughput is measured by means of application file transfer, the results are actually the application level throughput; i.e., the goodput. This is lower than the maximum throughput, because it does not consider some overheads, such as packet frames, protocol overheads, and collisions.

- Latency

This is a measure of how fast the BIEN network runs. This depends on transmission delays determined by the physical properties of Bluetooth, the behaviour of the software applications, the performance of the routing protocol and network topology, and the processing delays when passing through multiple hops. It refers to the time taken for a message to be sent from the sender to the receiver through a path of the network. In this work, the measurement is performed by measuring the *round-trip time* (RRT), using the *ping* technique. The RRT ping latency is determined by the time during which a ping message travels from the source node to the destination node, plus the time for the message returning to the source node. This excludes the time that the destination node requires to process the packet. This RRT latency is measured in milliseconds (ms). Therefore the latency may be calculated by subtracting the sending time of the ping message from the receiving time of the message, minus the processing time that the destination node takes to return the message. It can be presented as:

$$RRT\_Latency = T_{re} - T_{se} - T_{pr} \qquad \textbf{(Equation 6.3)}$$

Where T_{re} is the receiving time of a ping message on the sending node, T_{se} is the sending time of the ping message on the sending node, and T_{pr} is the processing time of the ping message on the destination node.

6.1.1.2 Test Messages

To facilitate experimental measurement, Test Messages are implemented as a part of the BIEN Messaging system (Shown in Figure 6.1). It consists of four message types, namely *PingMessage*, *TestFileMessage*, *TestDataMessage* and *TrafficMessage*. As an extension to the BIEN Message, these messages have the basic BIEN Message pattern (See Figure 5.9), plus one more field of a Sequence Number for the purpose of data recording and analysis (See Figure 6.2).

- A *PingMessage* is applied to perform a *ping* test in the Latency measurement and is processed as a *Control Message* in the software application. The object has a size of 47 bytes.
- A *TestFileMessage* and a *TestDataMessage* have the same functions respectively as the *FileMessage* and the *DataMessage* of the *Application Message,* and are used to simulate the two messages types in the experiments. The total size of a *TestFileMessage* object includes 48 bytes from the message object itself and the size of a file that it carries, while the size of a *TestDataMessage* object is 47 bytes.
- A *TrafficMessage* carries data of 8 Kbytes and is used to randomly generate traffic load by each BlueNode for the purpose of network performance testing. It is processed as an *Application Message*.

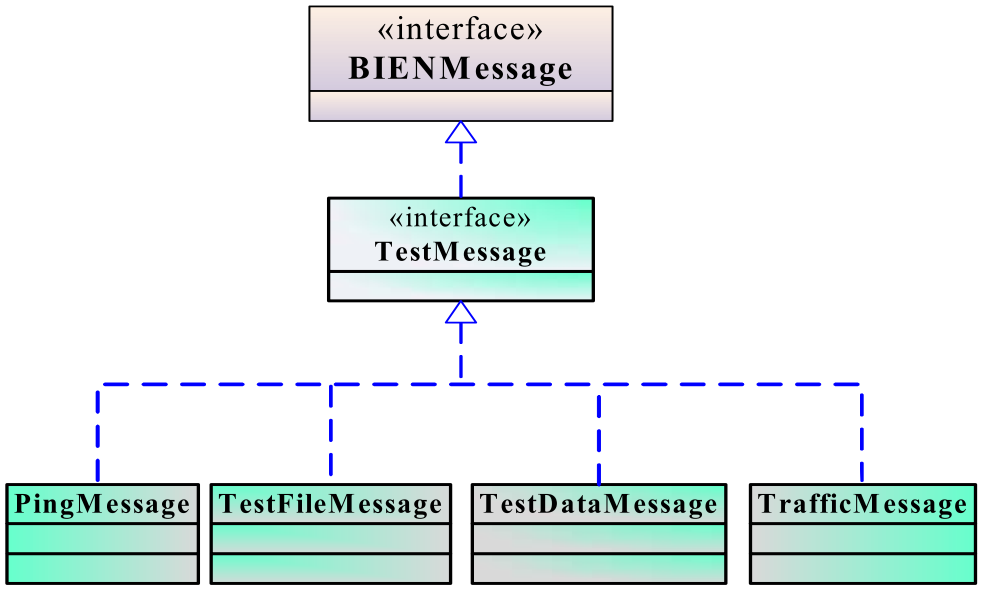


Figure 6.1 UML class diagram of test messaging system

Source address	**Destination address**	**Message Content**	**Sequence Number**

Figure 6.2 Basic pattern of Test Messages

6.1.1.3 Trace mechanism

In order to collect the experimental data, a trace mechanism is developed, based on the *LogFile.java* class, to perform operation tracking. This is embedded in the BIEN software application for a BlueNode to record each communication operation, including sending and receiving control and application data, in a text file for performance analysis.

Whenever a BlueNode sends or receives a message, the time for the operation is automatically recorded in milliseconds by the computer system timer, using the method "*System.currentTimeMillis()*", and written in a text file by the *LogFile* object in a format of: the operation time in milliseconds, the operation type; i.e., '-' for sending and '+' for receiving; the address of the message sender, the address of the message receiver, the message type, the sequence number of a Test Message, and the file size that the *FileMessage* carries. The format of the log file and an example of such a file are shown in Figure 6.3.

time	operation	sender	receiver	message type	sequence number	size

```
1211606239430 - 0009DD600A95 001060E31B44 testFile  43 56316
1211606244133 - 0009DD600A95 001060E31B44 testFile  44 56316
1211606249071 - 0009DD600A95 001060E31B44 testFile  45 56316
1211606295555 + 001060E16A90 0009DD600A95 trigRouting - 0
1211606425696 + 001060E16A90 0009DD600A95 trigRouting - 0
1211606425914 + 001060E16A90 0009DD600A95 trigRouting - 0
1211606253930 - 0009DD600A95 001060E31B44 testFile  46 56316
1211606259321 - 0009DD600A95 001060E31B44 testFile  47 56316
1211606264321 - 0009DD600A95 001060E31B44 testFile  48 56316
1211606269227 - 0009DD600A95 001060E31B44 testFile  49 56316
```

Figure 6.3 Format of log file and an example of a log file

For example, a line in the log file is as follows:

"1211606239430 - 0009DD600A95 001060E31B44 testFile 43 56316"

This states that a node with the address of 0009DD600A95, at the current time of 1211606239430 milliseconds, has sent a *TestFileMessage* carrying a file with a size of 56.316 Kbytes to the next hop node via which to the destination node (i.e. 001060E31B44) of the message.

For the RRT ping latency testing, if a node, such as 0009DD6002F7, sent a *PingMessage* to a node, such as 0009DD6002A1, at the time of 1215322842009 and received it back from the destination node at the time of 1215322842430, these operations will be automatically recorded in the log file of the sender node as follows:

```
1215322842009 -  0009DD6002F7 0009DD6002A1  ping  0
1215322842430 + 0009DD6002A1 0009DD6002F7  ping  0
```

Based on the RRT ping latency measurement discussed above, the RRT latency may be calculated using Equation 6-3, as:

$$RRT\_\text{Latency} = 1215322842430 - 1215322842009 - ProcessTime \qquad \textbf{(Equation 6.4)}$$

At the same time, the log file of the destination node will record the ping operation as follows:

```
1215322842142 + 0009DD6002F7 0009DD6002A1 ping 0
1215322842204 -  0009DD6002A1 0009DD6002F7 ping 0
```

This states that the destination node (i.e. 0009DD6002A1) received the ping message from the sending node (i.e. 0009DD6002F7), at the time of 1215322842142 and forwarded the ping message back to the sender at the time of 1215322842204. Hence, the processing time it used to forward the message back to the sender may be calculated by subtracting the receiving time from the sending time:

$$\textit{Process time}=1215322842204-1215322842142=62\ (ms)$$

Thus, the RRT latency can be obtained using Equation 6-4:

$$RRT\_Latency = 1215322842430-1215322842009-62= 359\ (ms)$$

The system clock of each BlueNode is synchronized with the time of a NTP time server at AUT University before each test is conducted. When the nodes in a BIEN network start communicating with each other, the log file in each node concurrently records every communication. Thus, the transfer time of a sent *TestFileMessage* can be obtained by subtracting the sending time recorded in the log file of the message sending node from the receiving time in the log file of the receiving node. It is calculated in milliseconds. Since the size of the file carried by the message can also be obtained in both log files, the current throughput of the message transmission path can be calculated using Equation 6-1.

6.1.2 Test Bed

The experiments were conducted in a computer classroom, WS 316 (shown in Figure 6.4), which is on the third floor of the WS building of AUT University. This indoor test bed provided a realistic environment for the BIEN experiments. The experiments were conducted over one hundred times over more than 30 test days.

Figure 6.4 The computer lab used for setting up the test bed

Each experiment was a combination of a deployment scenario and a test procedure. Based on the BIEN formation schema discussed in the Section 4.2, the BIENs set up in the experiment consisted of three piconets, with each piconet having the same number of slaves, that varying from three to five in different BIENs. There was only one S/S bridge interconnecting any two piconets and each S/S bridge only participated in two piconets. There is only one bridge between any two piconets and each piconet connected more than one piconet and the topology formed was mesh-like (Shown in Figure 6.5(a), (b) and (c)).

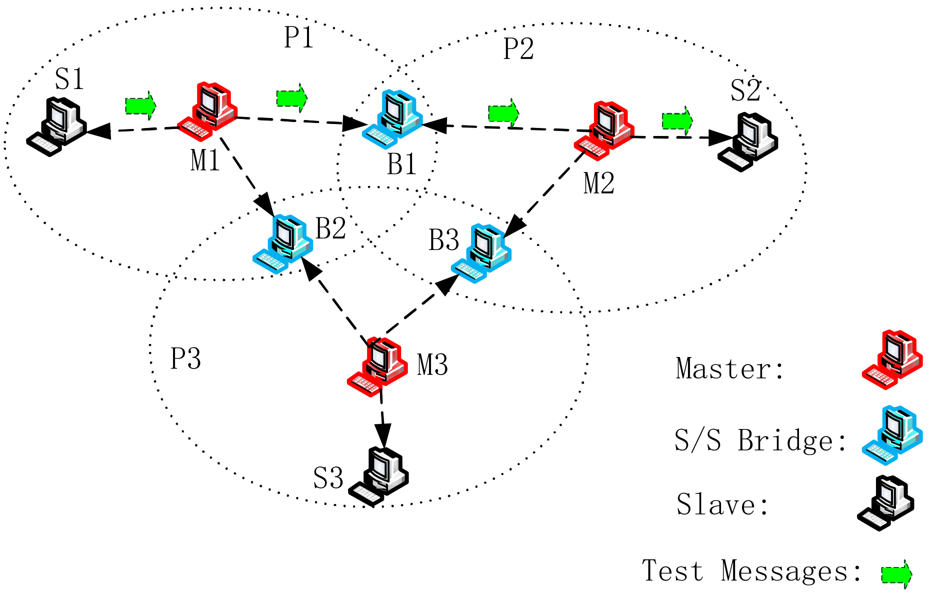


Figure 6.5 (a) Topology of first experimental BIEN with three slaves in each piconet

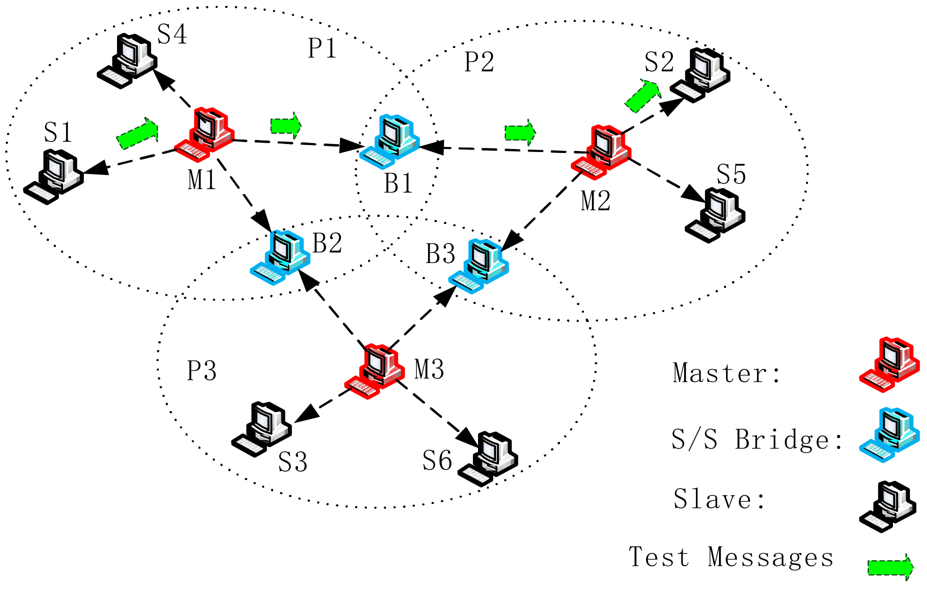


Figure 6.5 (b) Topology of second experimental BIEN with four slaves in each piconet

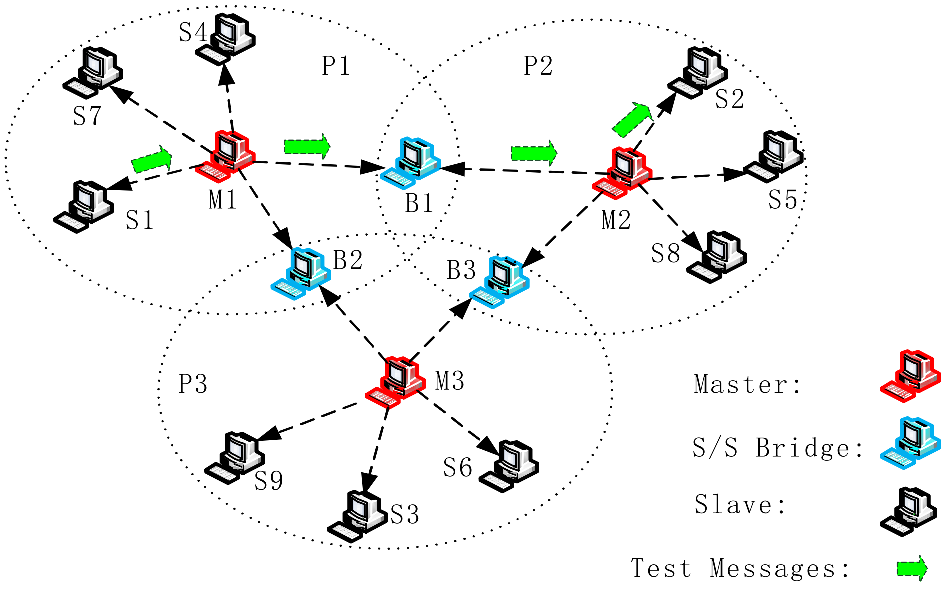


Figure 6.5 (c) Topology of third experimental BIEN with five slaves in each piconet

Figure 6.5 Structures of three experimental BIENs

These experimental BIENs are formed by a set of BlueNodes, each one of which has been configured with an identical BIEN software application. Each node consists of a stationary PC with a Bluetooth USB dongle. All the PCs are Intel Pentium IV 3GHz standard PCs with 1.00 GB of RAM based on Windows XP SP2. The dongles plugged into the PCs of the master nodes and the bridge nodes are XH8880 DSE Bluetooth USB dongles, while those used on the slave nodes are EA0021 DSE Bluetooth dongles. Both of these types of dongles support Bluetooth specification V2.0 + EDR, with a data rate of up to 3Mbps. The Bluetooth protocol stack used is the built-in Microsoft Bluetooth

Stack in Service Pack 2 for Windows XP. The BIEN software application is run in the *Eclipse* IDE, with *BlueCove* Java APIs. All BlueNodes were placed in three rows in the computer classroom. Two nodes in a row were placed one-by-one spaced by one meter, with the distance between rows being two meters (Shown in Figure 6.6).

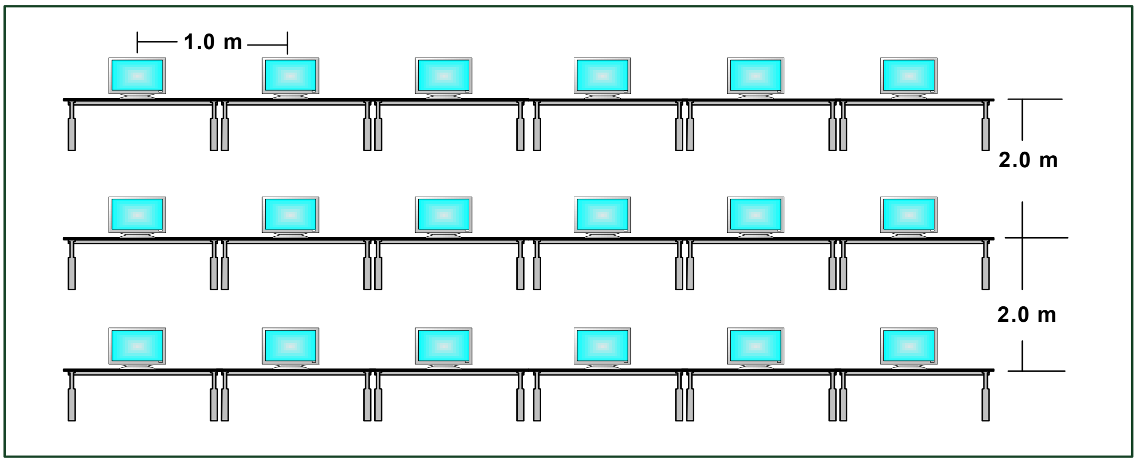
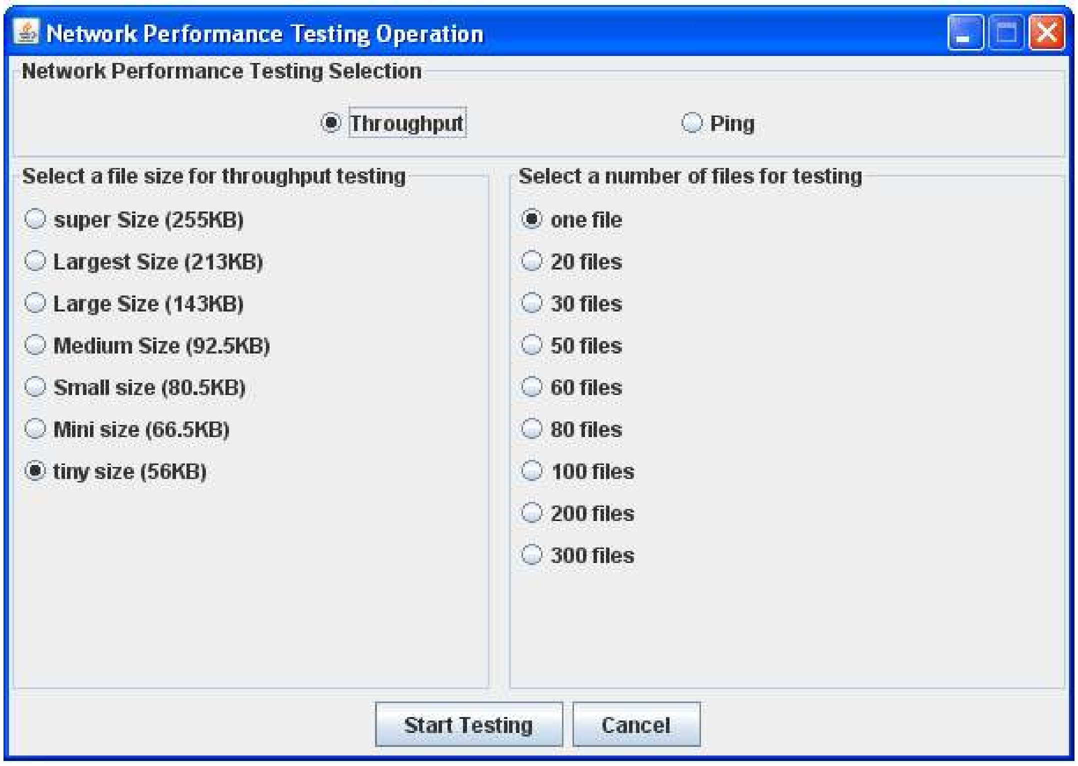


Figure 6.6 The layout of the test bed

A *TestTask.java* class is developed to perform the experimental task. To measure the throughput, or latency, of a communication path, the sender may select the destination node of the path from the *BIEN Scatternet* panel of the BIEN system GUI (Refer to Figures 5.23 and 5.27) and then press the *Testing* button in the GUI to display a *Testing Operation* panel (Shown in Figure 6.7)


Network Performance Testing Operation

Network Performance Testing Selection

[✓ Throughput] Ping

Select a file size for throughput testing
- super Size (255KB)
- Largest Size (213KB)
- Large Size (143KB)
- Medium Size (92.5KB)
- Small size (80.5KB)
- Mini size (66.5KB)
- [✓ tiny size (56KB)]

Select a number of files for testing
- [✓ one file]
- 20 files
- 30 files
- 50 files
- 60 files
- 80 files
- 100 files
- 200 files
- 300 files

Figure 6.7 Testing operation panel

Although the environment in which all of the experiments were conducted had been kept as little interference as possible, there were still factors existing around the test bed that could not be avoided and which may have affected the experimental results, such as 802.11b wireless activities and other Bluetooth activities from the cell-phones of people outside the computer classroom.

- Throughput measurement

The measurement of throughput on a certain communications path is performed by clicking the *Throughput* button in the subpanel on the top of the *Testing Operation* panel, selecting the size and the number of files the sender wants to send in the left and right subpanels, respectively, and then pressing the *Start Testing* button on the bottom of the panel. Thus, the sending node has been configured by the *TestTask* object to start sending the selected number of *TestFileMessages* one-by-one, as fast as the Bluetooth device allows, through the path to the destination node. Each *TestFileMessage* carries a file of the selected size. At the same time, the trace mechanism of the node automatically records the sending time of each message in its log file for performance analysis. On the other hand, while the destination node continuously receives the messages, the receiving time for the files has also been written in its log file (See Figure 6.8).

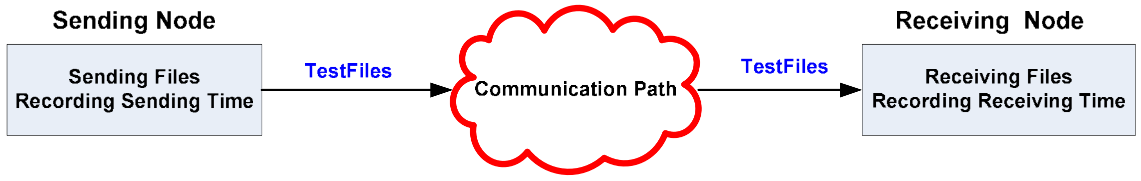


Figure 6.8 End-to-end throughput measurement

The different file sizes used in the throughput measurement are designed to obtain the maximum throughput value, as throughput varies with the size of the files transferred.

- Latency measurement:

The ping testing for the RRT latency measurement is performed by clicking the *Ping* button and selecting the number of ping messages in the right panel. Thus, the node has been configured to continuously send the selected number of *PingMessages* to the destination node once the *Start Testing* button is pressed. The destination node will automatically forward the message back to the sender if the message it receives is a

PingMessage. Both sending and receiving nodes will record the time at which the message was sent and received (See Figure 6.9).

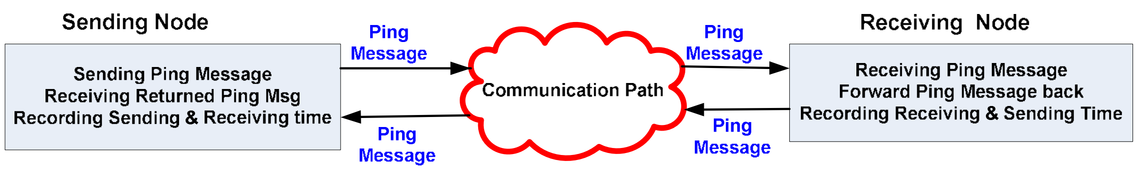


Figure 6.9 RRT ping latency measurement

In the ping test experiments, only one *PingMessage* was sent from the sender to the receiver for each ping test. After the sender received the message back, it started performing another ping test.

6.2 Performance evaluation

This section presents the experimental and evaluation results of the BIEN system behaviour and network performance. In the experiments, three BIENs had been set up (See Figure 6.5(a), (b) and (c)). They have similar, symmetrical, topologies, with each piconet having identical slave number that varies from 3 to 5 in the different experiments. As the structures of these experimental BIENs are symmetrical in which the longest paths have a length of 4 hops, the path selected for the end-to-end performance evaluation is the one from slave node *S1* of *Piconet_1* to another slave node *S2* of Piconet_2; i.e., *[S1→M1→B1→M2→S2]*; in all experimental BIENs.

In the experiments, only the *Test Messages* were sent or received by each node. All the data in this thesis are the result of measurements taken on these BIENs set up in the test bed.

6.2.1 Throughput evaluation

This refers to measuring the end-to-end application throughput on the selected path in the experimental BIENs and evaluating the impacts on the throughput, using varied path lengths and different numbers of slaves in the piconets in the different BIENs.

As the throughput of a network path varies with the size of the files that are transferred through the path, in order to obtain the maximum throughput of a path, the sizes of the files transferred on the path were increased from 400 bytes to 255 Kbytes until the throughput calculated based on the experiment data gradually became saturated. That is, until there were no more significant changes in the throughput. The files carried by the *TestFileMessages* included text and PDF files. The throughputs presented here are the maximum results obtained by transferring a number of files of a certain size (This data is displayed in the following figures).

As the throughput of a network path also depends on the load of the path, in order to obtain a reliable throughput, the load of a tested path is increased in each of the experiments by increasing the number of the *TestFileMessages* transferred on the path in each test until the throughputs no longer decrease significantly with the subsequent increases in the number of transferred files. The experimental results show that the throughput of a tested path has become stable when the number of the transferred files has increased to 50. Therefore, the experimental results of the throughput presented here are from tests in which 50 files were transferred.

6.2.1.1 Throughput versus path length

This is examined in order to evaluate the relation between throughput and path length in each experimental BIEN. The path length was varied from 2 to 4. The selected paths are the ones mentioned above, or a part of them. This includes a 2-hop path (e.g., *[S1→ M1 → B1]*), a 3-hop path (e.g., *[S1→M1→ B1→ M2],)* and the 4-hop path (e.g., *[S1→M1→B1→M2→S2])* in each BIEN (Shown in Figure 6.5 (a), (b) and (c)).

The first BIEN shown in Figure 6.5 (a) has three slaves in each piconet. The throughputs of the selected paths with different length in the network are shown in Figure 6.10.

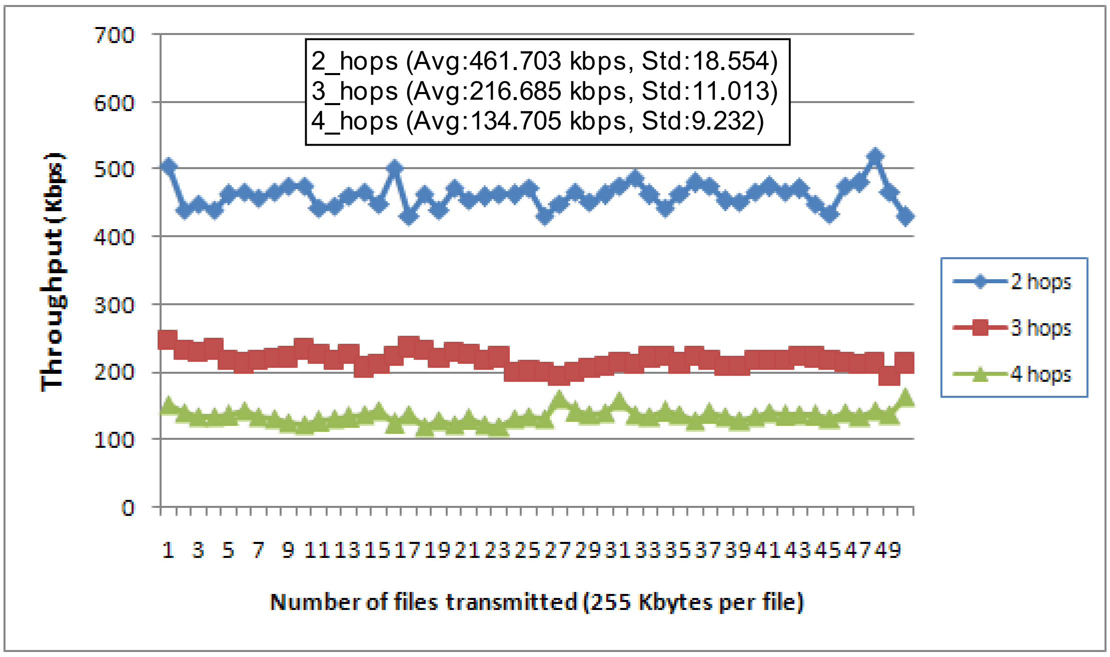


Figure 6.10 Application throughput versus path length in the BIEN shown in Figure 6.4 (a)

In Figure 6.10, it can be observed that, as the hop count increases to 2, 3 and 4, the network delivered a decreasing throughput, with averages of 461.703 Kbps, 216.685

Kbps, and 134.705 Kbps, respectively. This is due to the increased switching time between piconets as a result of longer paths. The throughput decreases by 53.1% with the increase in the path length from 2 to 3 hops *([S1, M1& B1] → [S1, M1, B1 & M2])* is reasonably larger than that of 37.8% in relation to the increase in the path length from 3 to 4 hops *([S1, M1, B1 & M2] → [S, M1, B1, M2 & S2]*). This is due to the path incorporating a bridge node B1 when the length increases from 2 to 3, which is a bottleneck to the traffic.

The second BIEN shown in Figure 6.5 (b) has four slaves in each piconet. The trends of throughput change corresponding to this path length are similar to those of the first BIEN (See Figure 6.11). The average throughput of the 2-hop path is 341.216 Kbps and that of the 3-hop path is 184.865 Kbps, which decreases by 45.8%, while the average throughput of the 4-hop path is 125.016 Kbps, which decreases by 32.4% from the throughput of the 3-hop path.

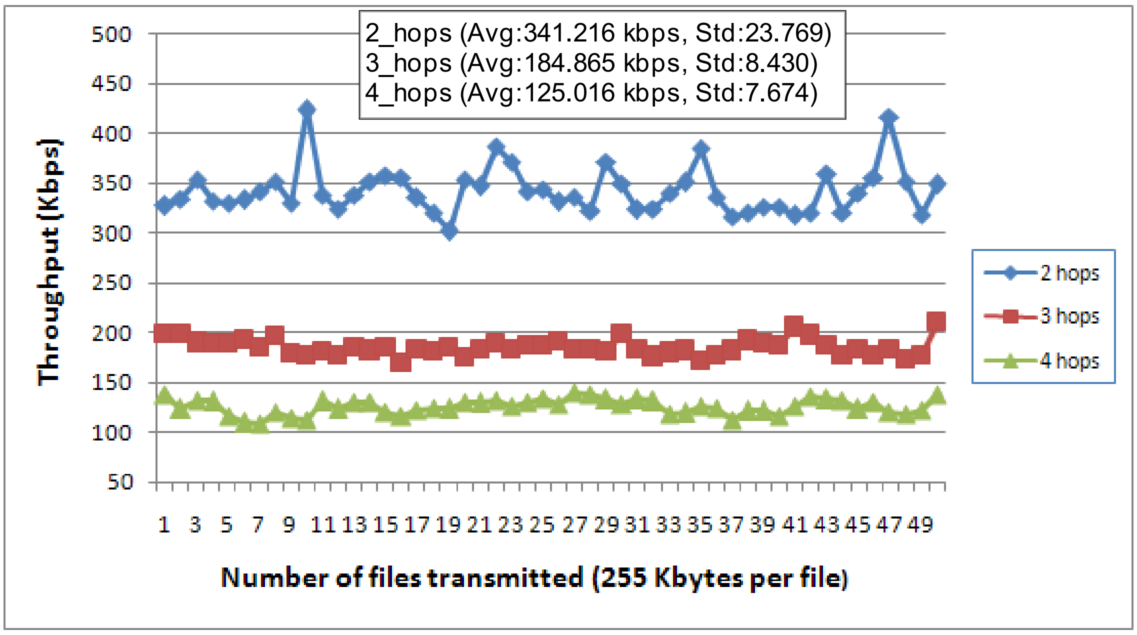


Figure 6.11 Application throughput versus path length in the BIEN shown in Figure 6.4 (b)

The third BIEN shown in Figure 6.5 (c) has five slaves in each piconet. The relation of throughputs versus path length is shown in Figure 6.12. The average throughput of the 2-hop path, the 3-hop path, and the 4-hop path, respectively, are 236.035 Kbps, 143.045 Kbps, and 123.802 Kbps, with respective decreases of 39.4% and 13.45% when the path length increases from 2 to 3 hops and from 3 to 4 hops.

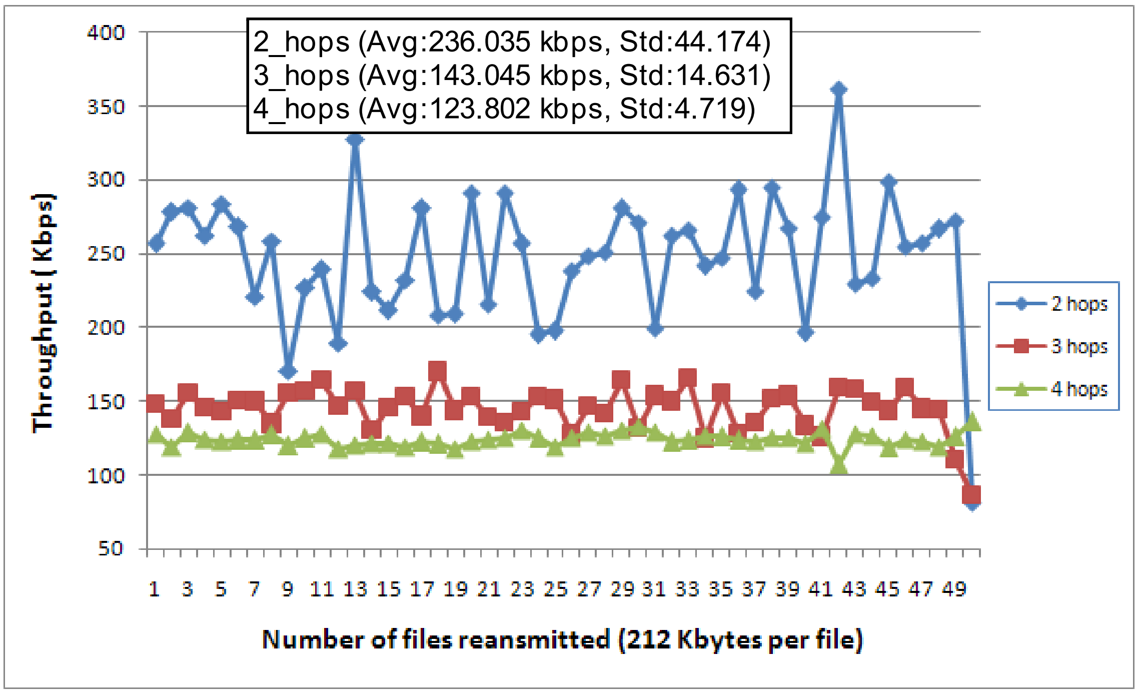


Figure 6.12 Application throughput versus path length in the BIEN shown in Figure 6.4 (c)

6.2.1.2 Throughput versus slave number in piconets

The impact on throughput of paths with different numbers of slaves in the piconets is also evaluated. Figure 6.13 shows the throughputs of the 2-hop path in the three BIENs (See Figures 6.4 (a), (b) and (c)). The number of slaves in a piconet varies from three to five.

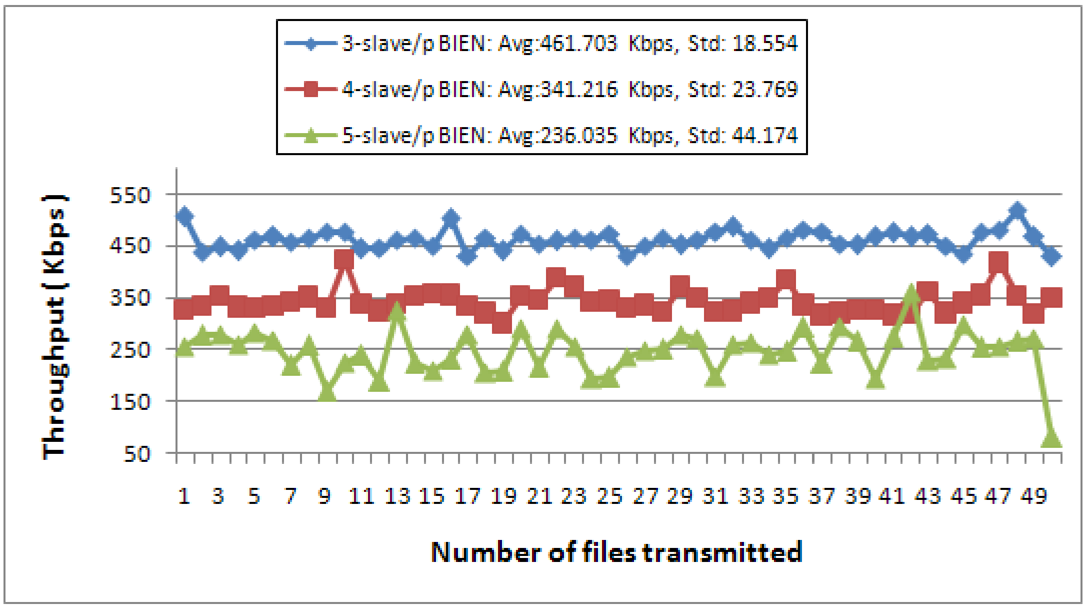


Figure 6.13 Throughput of 2-hop path versus the number of slaves of a piconet in BIENs

It can be observed that the throughput reduces with an increase in the number of slaves in the piconets in the different BIENs, in which the average throughput reduces from

461.703 Kbps to 341.216 Kbps and to 236.035 Kbps, respectively, when the number of slaves increases from 3 to 4, and 5. This is due to the chance that a slave gets to communicate with its master has been decreased and the overhead of the master has increased as a result of the increase of the number of slaves in the piconet, since all active slaves of a piconet share a common 1Mbps channel and use the TDM protocol to communicate through the master.

The throughputs of both the 3-hop path and the 4-hop path with different numbers of slaves of a piconet in the BIENs are shown, respectively, in Figures 6.14 and 6.15. In both networks, the trends of the throughput of the paths versus the varied number of slaves in a piconet are similar to that for the 2-hop path. The drop in throughputs of the 4-hop path is less than the drop of the 2-hop path and the drop of 3-hop path when increasing the number of slaves in the piconet.

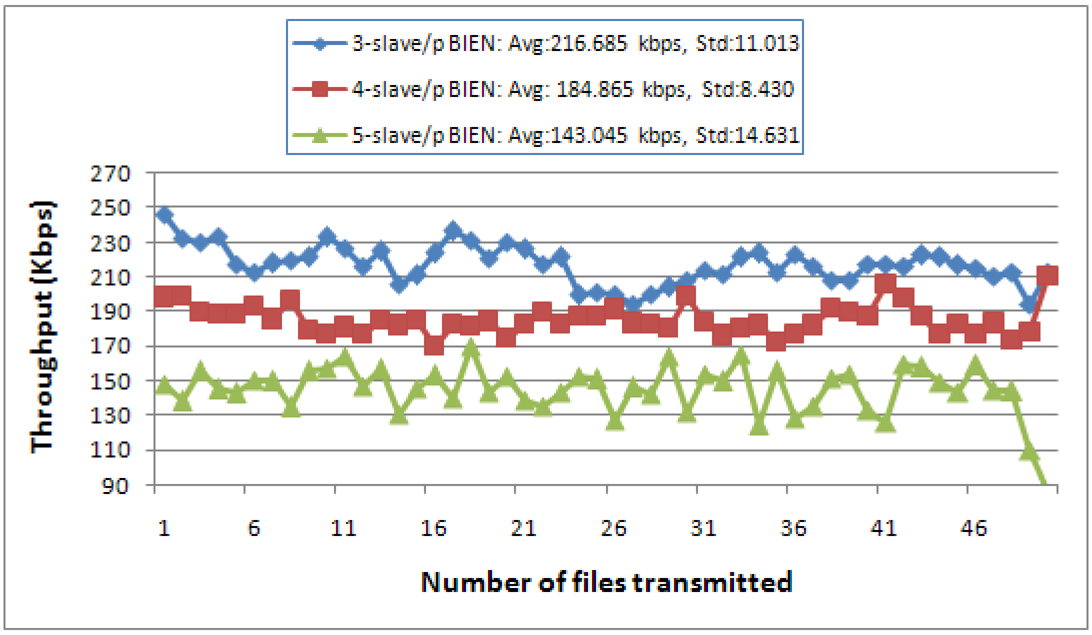


Figure 6.14 Throughput of 3-hop path versus the number of slaves of a piconet in BIENs

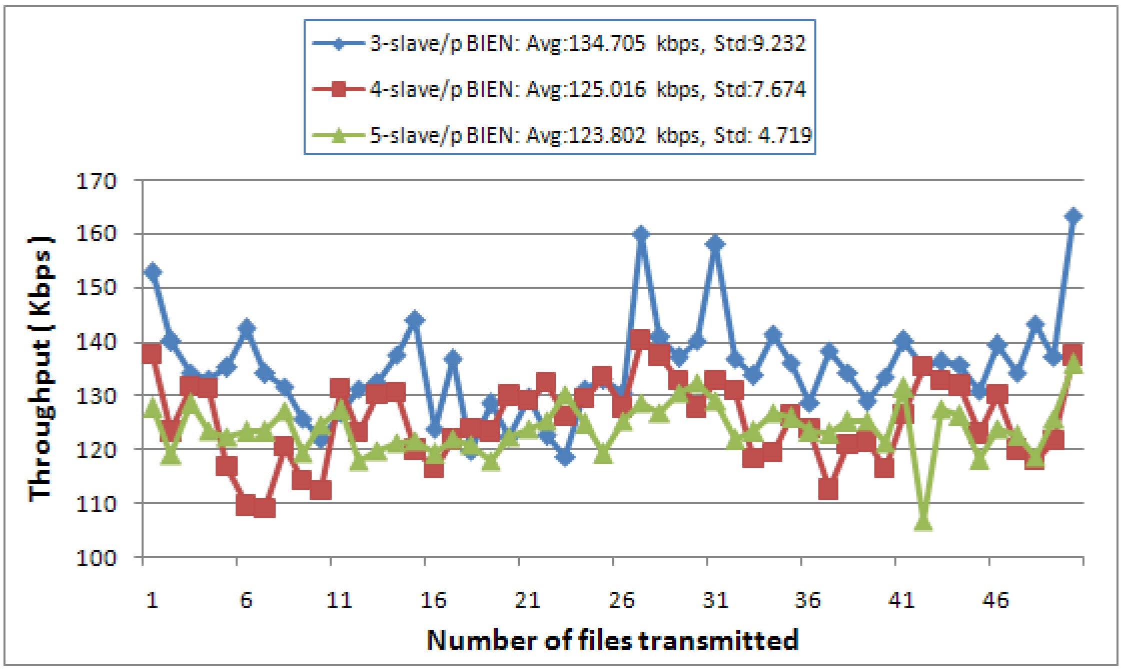


Figure 6.15 Throughput of 4-hop path versus the number of slaves of a piconet in BIENs

Figure 6.16 makes a summary of the relation of the average throughput of a path versus the path length, and the number of slaves in a piconet. It shows that the throughput reduces with the increased length of the path in all BIENs. It can also be observed that the throughput on a path with the same length reduces with an increase in the number of slaves in a piconet in all of the BIENs (Shown in Table 6-1).

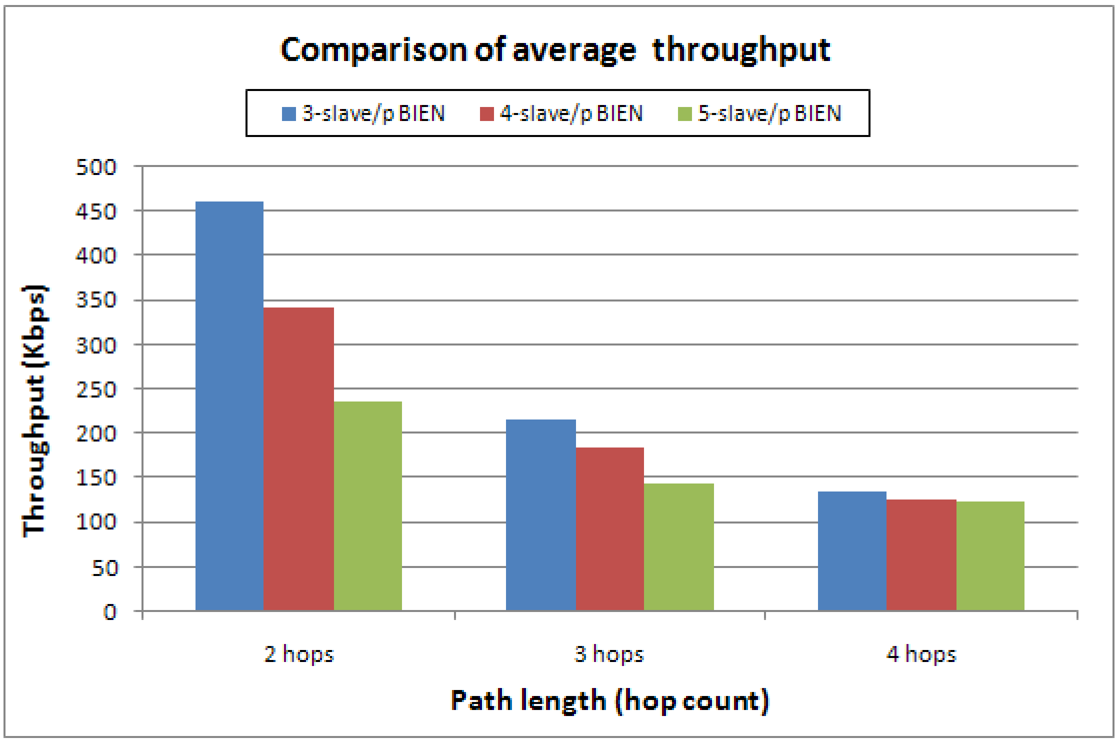


Figure 6.16 Comparison of average throughputs by different path length and varying slave numbers of a piconet in BIENs

Table 6-1 Decrease of throughput of a path with increase in the number of slaves in piconets

Path length (hops)	Increase of slave number in piconets	
	3 → 4	4 → 5
2 hops	26.09%	30.82%
3 hops	14.68%	22.62%
4 hops	7.19%	0.98%

6.2.2 RRT latency evaluation

This part of the test evaluation focuses on measuring the end-to-end RRT ping latency on selected paths in the experimental BIENs in order to evaluate the network's underlying performance related to the BIEN software application behaviour, the RIP routing protocol, and the network topology.

In the experiments, the RRT ping latency of the paths in the three BIENs mentioned above is tested in two traffic situations; one where the networks are free, and another one where networks are moderately loaded. The paths selected for the tests were the same paths used in the throughput tests.

- Traffic-free latency

In the tests of traffic-free latency, there is no other traffic in the network except for the *PingMessages* travelling from a sender (i.e., the node S1 in the experimental BIENs) to the receiver and then back to the sender.

For the first experimental BIEN with three slaves per piconet (See Figure 6.5 (a)), the impact of the varied path length on the RRT latency of the selected paths is shown in Figure 6.17. It has been observed that the latency increases with the increase of path length. This is mainly due to the increase in transmission delays and processing delays, as the ping messages pass through multiple hops as the path length increases. The average latency increases from 136.18 ms to 262.43 ms when the path length increases from 2 hops to 3 hops, while it increases to 348.45 ms when the path length becomes 4 hops. The corresponding latency percentages increase by 92.71% and 32.78% with the path increases from 2 to 3 hops and 3 to 4 hops, respectively. The reason for the large latency rise of 92.71% is due to the path incorporating bridge node B1 when the length increases from 2 to 3 hops, which creates bottlenecks for the traffic.

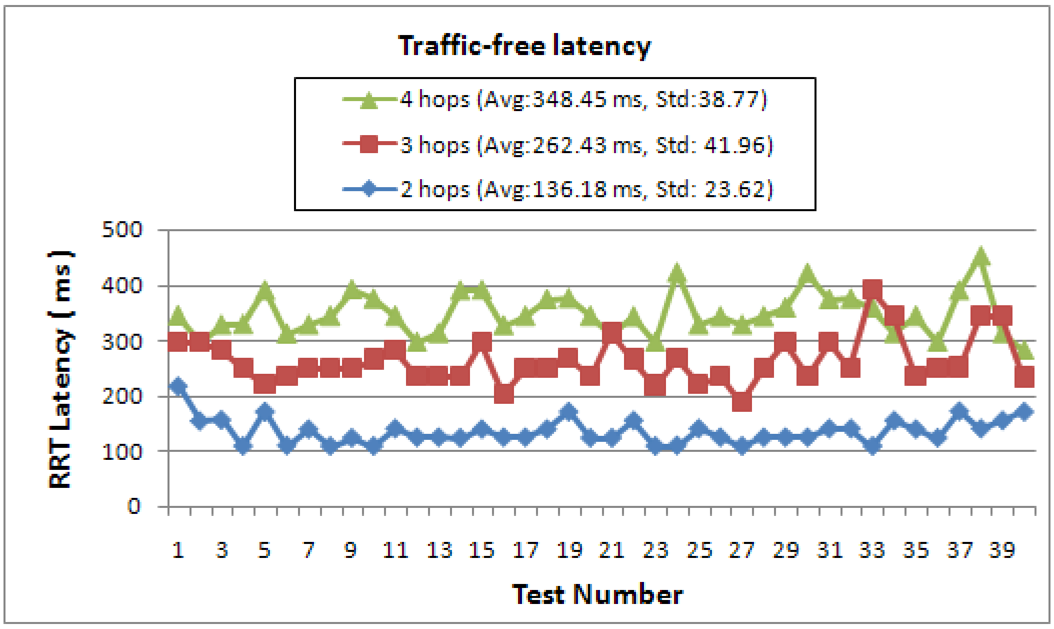


Figure 6.17 Comparison of traffic-free latency by path length in the BIEN shown in Figure 6.4 (a)

The RRT latencies of the same paths with varied path length in the second experimental BIEN and the third experimental BIEN are shown in Figures 6.18 and 6.19, respectively. Similar trends in the latency with an increase of the path length to that of the first BIEN have been, respectively, observed in the two figures. In the second experimental BIEN, which has four slaves in each piconet, the average latency rises from 155.50 ms to 279.40 ms, and to 380.90 ms when the path length increases from 2 to 3 hops and then to 4 hops, respectively, while the average latency of the third BIEN changes from 162.93 ms to 284.35 ms, and then to 386.40 ms, with a respective increase in the path length. The latency in these two BIENs also has a large increase when the path goes through bridge node B1; i.e., 79.68% and 74.45%, respectively.

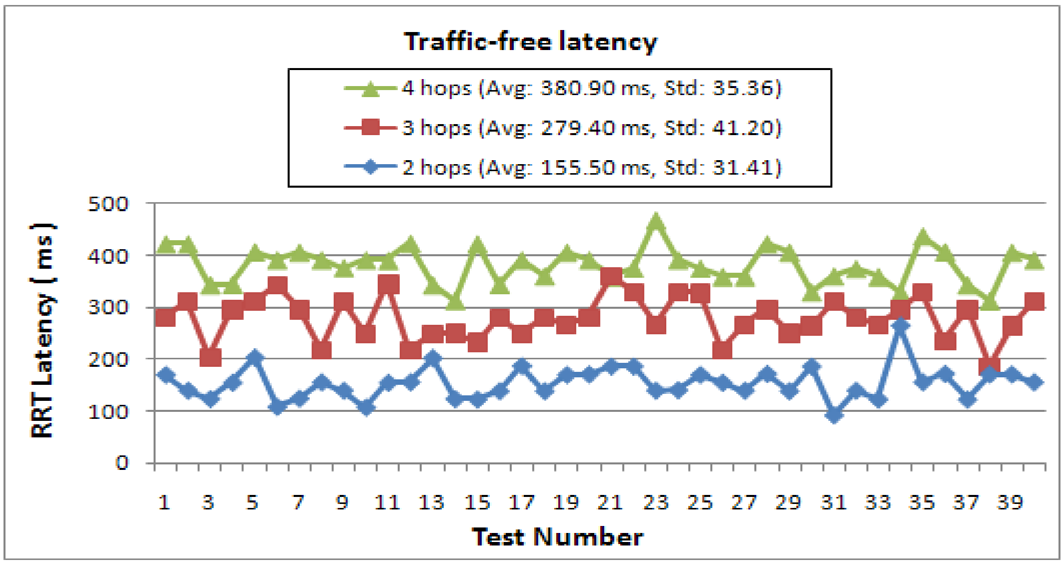


Figure 6.18 Comparison of traffic-free latency by path length in the BIEN shown in Figure 6.4 (b)

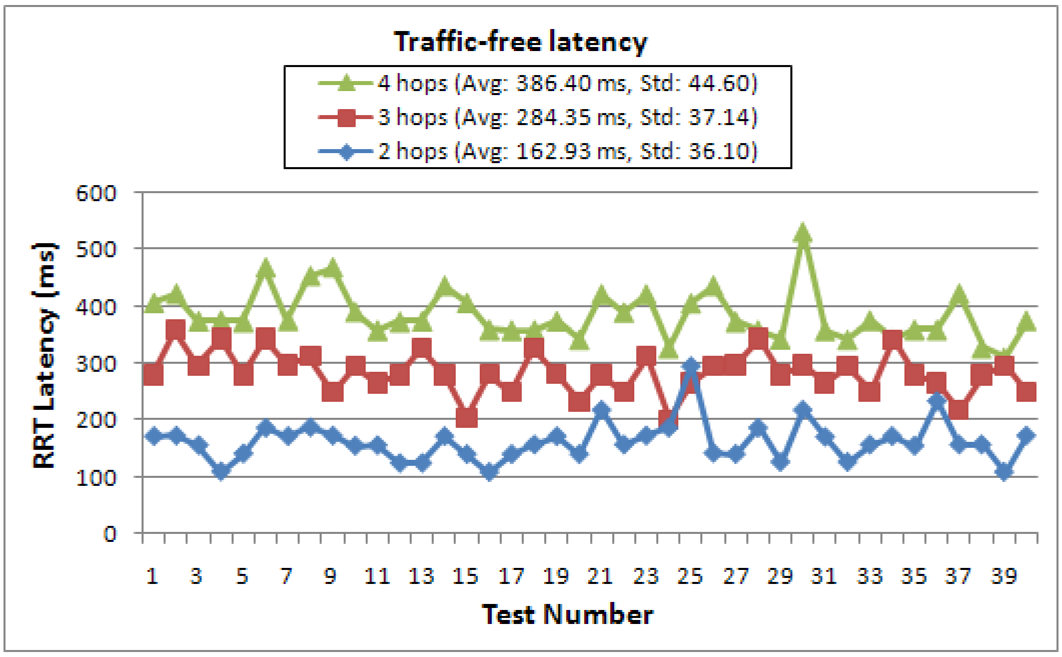


Figure 6.19 Comparison of traffic-free latency by path length in the BIEN shown in Figure 6.4 (c)

Figure 6.20 shows a comparison of the average traffic-free latency by varying the path length and the different numbers of slaves in the different experimental BIENs. As well as showing the trend that the latency increases with increases in the path length in all of the BIENs, it can also be observed that the latency of a path with the same length rises with the number within slaves of a piconet in the experimental networks. This is due to the increased switching overhead inside a piconet when the number of slaves of the piconet increases.

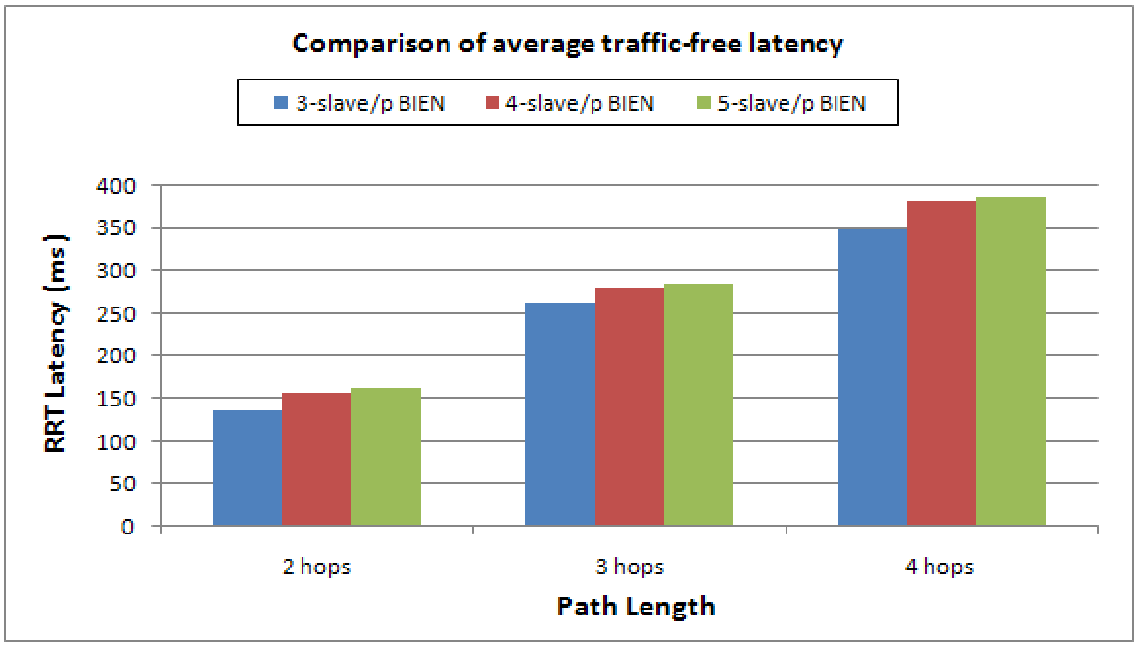


Figure 6.20 Comparison of average traffic–free latency in the experimental BIENs

- Traffic-dependent latency

The methods used to test the traffic-dependent latency were the same as those used in the tests of the traffic-free latency. In these tests of traffic-dependent latency, however, all the nodes in the experimental BIEN, except the sender of the *PingMessages,* automatically kept generating traffic loads through the network by continuing to send random numbers of *TrafficMessages* to randomly selected destination nodes in the network at random time intervals.

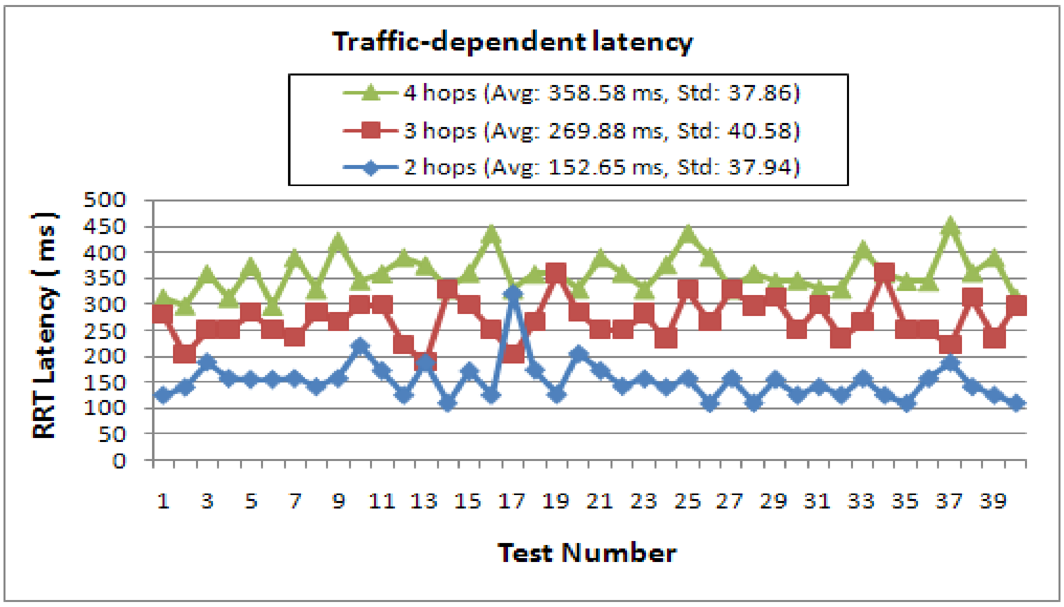


Figure 6.21 Comparison of traffic-dependent latency by path length in the BIEN shown in Figure 6.4 (a)

Figure 6.21 shows a comparison of the traffic-dependent latency by varying the path length in the first experimental BIEN. It can be seen that, similarly to the traffic-free latency results, the traffic-dependent latency increases with an increase in the path length. The average latency increases from 152.65 ms to 269.88 ms when the path length increases from 2 hops to 3 hops, and it increases to 358.58 ms when the path length increases to 4 hops. The corresponding percentages of the latency increases are 76.80% and 32.87% with path increases from 2 to 3 hops and from 3 to 4 hops, respectively. As seen in the tests of the traffic-free latencies above, there is a large latency rise of 76.80% when the path passes through bridge node B1, which is the traffic bottleneck that causes the increase in the latency.

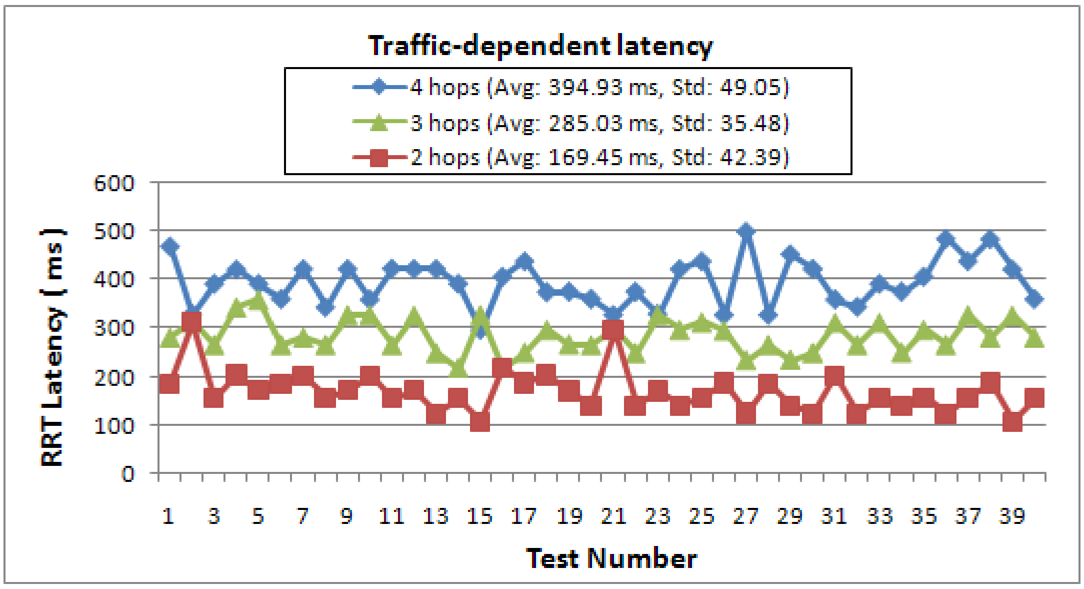


Figure 6.22 Comparison of traffic-dependent latency by path length in the BIEN shown in Figure 6.4 (b)

In the tests of traffic-dependent latency for the second and third experimental BIENs, the impact of path length on latency in these two BIENs is shown in Figures 6.22 and 6.23, respectively. It has been observed that the latency in both networks increases with path length, as it does in the first BIEN. In Figure 6.22, the average traffic-dependent latency in the second BIEN increases from 169.45 ms to 285.03 ms, and to 394.93 ms when the path length increases from 2 hops to 3 hops and then to 4 hops, respectively, while the corresponding latencies in the third BIEN are 180.85 ms, 307.65 ms, and 434.90 ms (See Figure 6.23). The latency in both networks also increases greatly, by 70.11% and 68.2%, respectively, in the second and third BIENs, when the path length passes through bridge node B1.

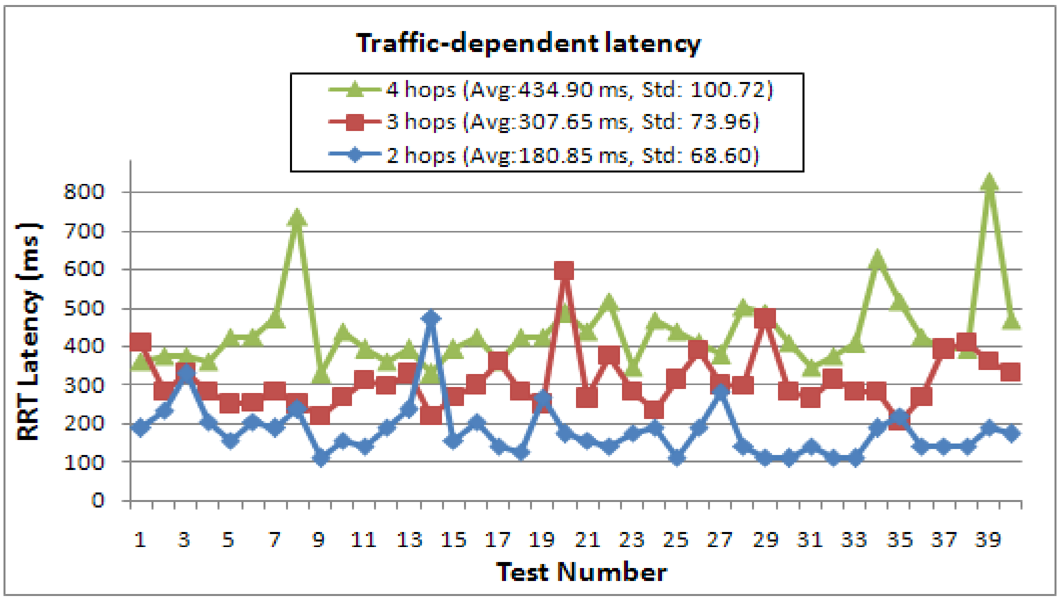


Figure 6.23 Comparison of traffic-dependent latency by path length in the BIEN shown in Figure 6.4 (c)

Figure 6.24 shows a comparison of the average traffic-dependent latency by the varying path length and the varied number of slaves in the three experimental BIENs. As discussed above, latency increases with path length in all of the BIENs. It can also be seen that the latency of a path with the same length increases with the number of slaves in a piconet in the networks due to increased overheads inside a piconet when a piconet has a greater number of slaves.

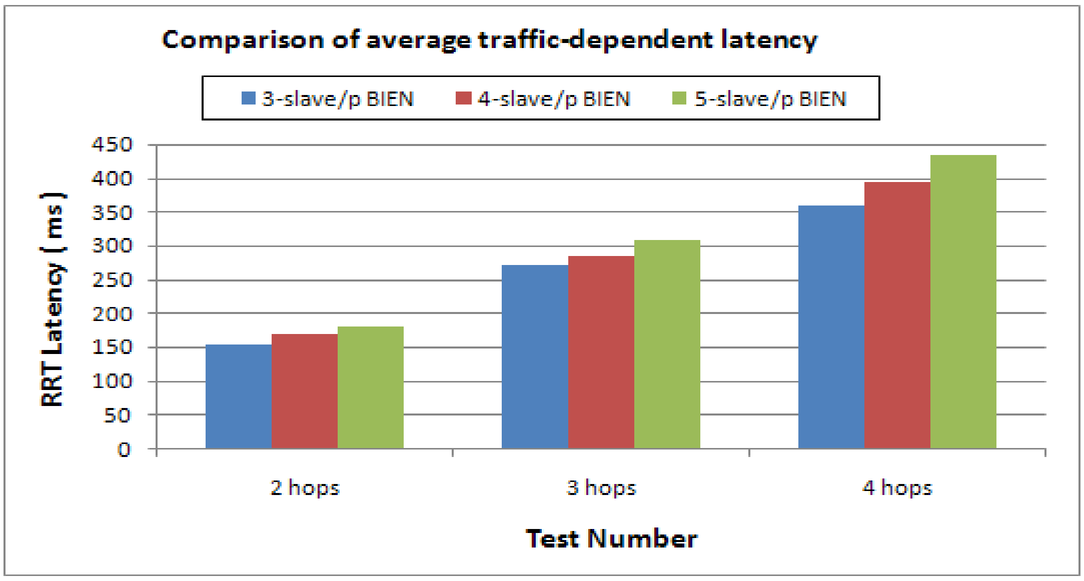


Figure 6.24 Comparison of average traffic-dependent latency in the experimental BIENs

Figure 6.25 presents a summary of the relation of the average RRT latency in both traffic-free and traffic-dependent situations, versus the path length and the number of slaves in a piconet discussed above. In Figure 6.25, latency increases with path length whether or not there is traffic in all of the networks. It is also found that the latency of a path gradually increases with the number of slaves in a piconet in the different BIENs, in both traffic situations. The average traffic-dependent latencies of the paths are, however, slightly bigger than those of the traffic-free of the same paths in the networks, due to the increased processing time as a result of the increased traffic load. In this figure, the trend of the latencies, in terms of path length and the number of slaves of a piconet in all of the experimental BIENs have confirmed the correct behaviour of the BIEN software application, high efficiency of the implemented routing protocol, and stable performance in the latency of the BIEN network.

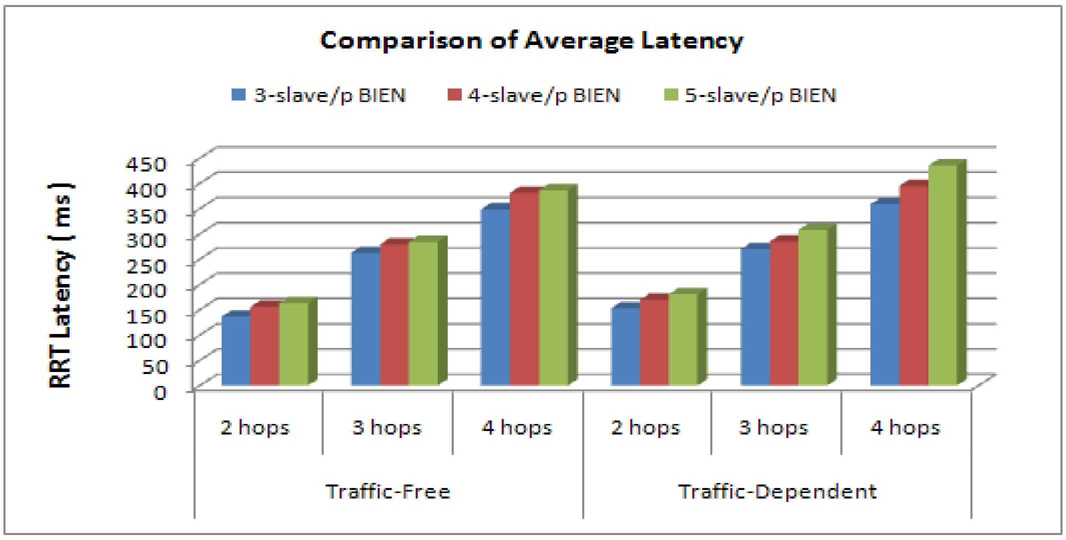


Figure 6.25 Comparison of average latency by different path length and the varying number of slaves in a piconet in the BIENs

6.3 Summary

This chapter presented the experimental deployment, the three experimental BIEN networks set up in the experiments, and the measurement methods used in testing and evaluating the performance in terms of the throughput and RRT latency of the networks. These BIENs have delivered reliable throughput and low latency over the multiple hops with varying numbers of slaves within the piconets. Although some graphs shown above contain instances of sharp increases or significant troughs as a function of the x-axis parameter, this is due to overhead of the traffic of routing update and data transferring from the rest of the networks when the experiments were conducting. Additional contribution comes from the unavoidable interference around the experimental environments. The experimental results have confirmed a clear and strong association between the performance for throughput and latency, in relation to path length and the number of slaves in a piconet. They have also further proved the necessity for minimizing the number of the bridge nodes in the scatternets due to their traffic bottleneck effect in such networks. Therefore, in order to achieve good performance in terms of throughput and latency, a maximum of three to four hops in a transmission path in Bluetooth wireless network is recommended.

7 Conclusions and future work

This thesis presented the Bluetooth information exchange network (BIEN), an indoor ad hoc Bluetooth wireless network designed to support information exchange between Bluetooth devices. This last chapter summarizes the achievements of this work and discusses possible directions for future work.

7.1 Summary

The BIEN developed here is an approach to implement an ad hoc Bluetooth multi-hop wireless network, using commercial Bluetooth devices for short-range communication. It demonstrates the feasibility of Bluetooth technology supporting a spontaneous wireless network for short-range communication.

The thesis initially introduced the background and current knowledge regarding Bluetooth technology, Bluetooth ad hoc wireless network technology, the existing solutions to Bluetooth scatternet formation, wireless network routing protocols, and the hardware and software tools used for developing the BIEN application. This was very important for the efficient design of the BIEN system and the BIEN formation schema, and the development of the BIEN application.

The BIEN application developed in this work allows Bluetooth devices to connect to each other to spontaneously form a BIEN of any topology and any size. Through the implementation of the RIP routing protocol, each Bluetooth device within the network works as a router and freely exchanges information with any other device in the network by means of file and image transfer, and communicating by text.

The developed BIEN application was evaluated in a test-bed in which three BIENs were built. These experimental BIENs were created in conformation to the developed BIEN formation schema, using a group of Bluetooth devices that were configured with the

BIEN application. They had similar topologies, but the number of slaves in a piconet in these networks varied from three to five. The performance evaluation was made with a focus on the throughput and latency of the communication paths. The positive associations of performance in throughput and latency with path length and slave number in a piconet have confirmed the correct behaviour of the BIEN application, the high efficiency of the RIP routing protocol by the availability of routes with the minimum latency time, and the steady performance in the throughput and latency of the BIEN networks. The experimental results have also further confirmed the necessity for minimizing the number of bridge nodes in Bluetooth networks.

Overall, although the principal tool used for developing the Bluetooth network application, Bluetooth APIs (e.g. JSR-82), is still under development, this thesis demonstrated how Bluetooth technology can be used to design, develop, and deploy ad hoc wireless networks, using commercial Bluetooth devices for communication. Most current research on the topic of Bluetooth network formation is based on simulation results. Due to this, the most important contribution of this thesis is that it presents a practical proof of how well Bluetooth technology supports ad hoc wireless network technology. It also demonstrated how well the RIP routing protocol can be used to perform routing in an ad hoc Bluetooth wireless network. All these results are helpful in designing Bluetooth scatternet formation algorithms and in the development of routing mechanism for such networks.

7.2 Future work

Due to time limitations, some aspects of the BIEN have not been numerically investigated in this research. The developed application can continue to operate as an evaluation tool for Bluetooth networks of differing topologies, sizes and varying coverage in terms of distance for the node spacing. Based on the current work, future work could focus on enhancing the application functionality and improving the performance of BIEN:

- Mobility issue: This could be tested by adapting the BlueNode application to Bluetooth mobile devices, such as Cell phones and PDAs, to form outdoor ad hoc Bluetooth mobile BIENs for information exchange.
- Alternative ad hoc routing protocols: An investigation of the possibility of improving the routing efficiency of the BIENs by implementing other ad hoc wireless routing protocols, such as DSDV and AODV.

- Facilitating a Bluetooth-Zigbee interface for connecting Bluetooth networks to low power processing networks.

- Security issue: An implementation of security features to the BIEN, such as authentication and encryption of usage of the network, could prevent bogus users, or information interception.

References

[1] Mahmoud QH. Wireless application programming with J2ME and bluetooth. 2003 February, 2003 [cited 2006 October 10]; Available from: http://developers.sun.com/mobility/midp/articles/bluetooth1/

[2] Bluetooth SIG. Specification of the Bluetooth system. 2004 [cited 2006 25 June]; Available from: http://www.bluetooth.com/Bluetooth/Technology/Building/Specifications/

[3] Bluetooth SIG. Specification of the Bluetooth system. 2007 [cited 2007 October 25th]; 2.1 + EDR:[Available from: http://bluetooth.com/Bluetooth/Learn/Technology/Specifications/

[4] Bluetooth SIG. Architecture - data transport. How Bluetooth technology works 2007 [cited 2007 17th March]; Available from: http://www.bluetooth.com/Bluetooth/Technology/Works/Data_Transport_Architecture.htm

[5] Bluetooth SIG. Core system architecture. How Bluetooth technology works 2007 [cited 2007 Jan 5th]; Available from: http://www.bluetooth.com/Bluetooth/Technology/Works/

[6] Hopkins B. Wireless J2ME application with Java and Bluetooth. 2002 [cited 2007 July 5]; Available from: http://java.sys-con.com/node/37089

[7] Hopkins B. Getting started with Java and Bluetooth. 2004 [cited 2006 July 5]; Available from: http://today.java.net/pub/a/today/2004/07/27/bluetooth.html

[8] Enrique Ortiz C. Using the Java APIs for Bluetooth wireless technology. part 1 - API overview 2004 [cited 2007 May, 8th]; Available from: http://developers.sun.com/mobility/apis/articles/bluetoothintro/

[9] Miklos G, Racz A, Turanyi Z, Valko A, Johansson P. Performance aspects of bluetooth scatternet formation. MobiHoc 2000; 2000 August 2000; Boston; 2000. p. 147-8.

[10] Basagni S, Bruno R, Petrioli C. A performance comparison of scatternet formation protocols for networks of Bluetooth devices. First IEEE International Conference on Pervasive Computing and Communications (PerCom'03); 2003: IEEE Computer Society; 2003. p. 341.

[11] Misic J, Misic VB. Performance modeling and analysis of Bluetooth networks: Auerbach Publications 2006.

[12] McDermott-Wells P. Bluetooth scatternet models. IEEE Potentials. 2004:36-9.

[13] Zaruba GV, Basagni S, Chlamtac I. Bluetrees-scatternet formation to enable bluetooth-based ad hoc network. ICC 2001. 2001 11-14 June;1:273-7.

[14] Tan G, Miu A, Guttag J, Balakrishnan H. Forming scatternet from bluetooth personal area networks. 2001 [cited 2006 October 20th]; Available from: http://publications.csail.mit.edu/
[15] Law C, Mehta AK, Siu KY. Performance of a new bluetooth scatternet formation protocol. Proc 2nd ACM Symposium on Mobile Ad Hoc Network and Conputing (MobiHoc' 01); 2001; Long Beach , USA; 2001. p. 182-92.
[16] Dong Y, Wu J. Three Bluetree formations for constructing efficient scatternets in Bluetooth. the 7th Joint Conference on information sciences; 2003; 2003. p. p 385-3.
[17] Petrioli C, Basagni S, Chlamtac I. Configuring bluestars: multihop scatternet formation for bluetooth networks. IEEE Transactions on Computers. 2003;52(6):779-90.
[18] Basagni S, Bruno R, Petrioli C. Performance evaluation of a new scatternet formation protocol for multi-hop bluetooth networks. 5th International Symposium on WPMC; 2002; 2002. p. 208-12.
[19] Wang Z, Thomas RJ, Haas Z. Bluenet-a new scatternet formation scheme. The 35th Annual Hawaii International Conference on System Sciences (HICSS-35'02); 2002; Hawaii; 2002. p. 273-7.
[20] Stojmenovic I. Dominating set based Bluetooth scatternet formation with localized maintenance. The Workshop on advances in Parallel and Distributed Computational Models; 2002 April 2002; Ford Lauderdale, FL; 2002.
[21] Petrioli C, Basagni S, Chlamtac I. BlueMesh: degree-constrained Multi-hop scatternet formation for Bluetooth networks. Mobile Networks and Applications. 2004;9:33-47.
[22] Cgun-choong F, Kee-chaing C. Bluerings - Bluetooth scatternets with ring structure IASTED Wireless and Optical Communications WOC; 2002 July; Banff, Canada; 2002.
[23] Lin TY, Tseng YC, Chang KM, Tu CL. Formation, routing, and maintenance protocols for the bluering scatternet of Bluetooths. The 36th Hawaii International Conference on System Sciences (HICSS'03); 2003 January; Hawaii, USA; 2003. p. 313-22.
[24] Salonidis T, Bhagwat P, Tassiulas L, Lamaire R. Distributed topology construction of bluetooth personal area networks. IEEE INFOCOM 2001; 2001 April 22-26, 2001; Alaska, US: IEEE; 2001. p. 1577-86.
[25] Liu Y, Lee MJ, Saadawi TN. A Bluetooth scatternet-route structure for multihop ad hoc networks. IEEE journal on selected areas in communications. 2003 February;21(2).
[26] Misic VB, Misic J. Performance of Bluetooth bridges in scatternets with limited service scheduling. Moblie Networks and Applications. 2004 Feburary 2004;9(1):73-87.
[27] Misic J, Misic VB. Performance modelling and analysis of Bluetooth networks. Boca Raton, Florida: Auerbach Publications, Taylor & Francis Group, LLC 2006.
[28] Misic VB, Misic J. Minimizing end-to-end delays in Bluetooth scatternet with a slave/slave bridge. Computer Communications and networks; 2002 14-16 October; 2002. p. 634-9.
[29] Basagni S, Bruno R, Mambrini G. Comparative performance evaluation of scatternet formation protocols for networks of Bluetooth devices. Wireless Networks. 2004 March 2004;10(2).
[30] Li XY, Stojmenovic I, Wang Y. Partial Delaunay triangulation and degree-limitedlocalized Bluetooth scatternet formation. IEEE transactions on parallel and distributed systems. 2004 March 8th;15(4):350-61.
[31] Cano J-C, Ferrandez-Bell D, Manzoni P. Evaluation Bluetooth performance as the support for context-aware application. Telecommunication Systems. 2005 2005;28(3,4):333-47.

[32] Cano J-C, Manzoni P, Toh C-K. UbiqMuseum: a Bluetooth and Java based context-aware system for ubiquitous computing. Wireless Personal Communications. 2006;38:187-202.
[33] Cano J-C, Cano J, Manzoni P. On the design of pervasive computing applications based on Bluetooth and a P 2 P concept. IEEE. 2006.
[34] Gpnzalez-Castano FJ, Garcia-Reinoso J, Gil-Castineira F, Costa-Montenegro E, Pousada-Carballo JM. Bluetooth-assisted context-awareness in educational data networks. Computers & Education. 2005;45:105-21.
[35] Krco S. Bluetooth based wireless sensor networks-Implementation issues and solutions. Telecommunication Forum (Telfor). 2002.
[36] Cuomo F, Pugini A. A linux based Bluetooth scatternet formation kit: from design to performance results. IEEE ICPS Workshop on Multi-hop Ad Hoc Networks: From Theory to Reality (REALMAN); 2005; 2005.
[37] Forno F, Malnati G, Portelli G. Design and implementation of a Bluetooth ad hoc network for indoor positioning. IEE Proceedings- Software. 2005 7 October;152(5):223-8.
[38] Inayatullah M, Shah SW, Khan Z, Raza SS, Suhail A. Implementation of table driven routing protocol for small sized MANET using Bluetooth in Java. Emerging Technologies ICET 2007 2007 12-13 November; 2007. p. 27-30.
[39] Royer E, Georgia C-K. A review of current routing protocols for ad hoc wireless networks. IEEE personal communication 1999 April.
[40] Mir NF. Computer and communication networks: Prentice hall 2006.
[41] Tanenbaum AS. Computer networks. 4th ed: Prentice Hall 2003.
[42] Black PE. Bellman-Ford algorithm. 2006 [cited 2007 15th July]; Available from: http://www.nist.gov/dads/HTML/bellmanford.html
[43] Perkins CE, Bhagwat P. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. ACM SIGCOMM'94; 1994 Sept, 1994; London, U.K.; 1994. p. 234-44.
[44] CISCO. Routing Information Protocol (RIP). Internetworking Technology Handbook [cited 2007 Oct 20]; Available from: http://www.cisco.com/en/US/docs/internetworking/technology/handbook/RIP.html
[45] Black PE. Dijkstra's algorithm
2006 [cited 2007 15th July]; Available from: http://www.nist.gov/dads/HTML/dijkstraalgo.html
[46] Perkins CE, Royer EM. Ad-hoc on-demand distance vector routing Mobile Computing Systems and Applications, WMCSA '99 Second IEEE Workshop on; 1999 25-26 Feb 1999; New Orleans, LA, USA; 1999. p. 90-100.
[47] Johnson D. The dynamic source routing protocol (DSR) for mobile ad hoc networks for IPv4 Rice University, Microsoft Research; 2007 February 2007.
[48] Jayakumar G, Gopinath G. Ad hoc mobile wireless networks routing protocols - a review. Compurer Science. 2007;3(8):574-82.
[49] Poor R. Wireless mesh networks. 2003 [cited 2007 25th September]; Available from: http://www.sensorsmag.com/articles/0203/38/main.shtml
[50] Eagles D. Wireless mesh networks. Australia; 2007 5th January, 2007.
[51] Held G. Wireless mesh networks: Auerbach Publications
Taylor & Francis Group, LLC 2005.
[52] Akyidiz IF, Wang X, Wang W. Wireless mesh networks: a survey. Computer Network. 2005;47:445-87.
[53] Hiertz GR, Max S, Weib E, Berlemann L, Denteneer D, Mangold S. Mesh technology enabling ubiquitous wireless networks. WICON'06, The 2nd Annual International Wireless Internet Conference 2006 August 2-5, 2006; Boston, MA, United States; 2006.

[54] cambridgeshiretouristguide.com. Cambridge strawberry fair. 2006 [cited 2008 15th January]; Available from: http://www.cambridgeshiretouristguide.com/Articles/Article_55.asp
[55] Meraki.com. Meraki Mesh 2007 2008 [cited 2008 20th January]; Available from: http://meraki.com/oursolution/mesh/
[56] smesh.org. SMesh. [cited 2008 19th January]; Available from: http://smesh.org/
[57] Sunkavalli S, Rarnalmurthy B. MTSF: a fast mesh scatternet formation algorithm for bluetooth networks. Global Telecommunications Conference, 2004 GLOBECOM'04 IEEE; 2004 29 November- 3 December. 2004; 2004. p. 3594-8.
[58] Jayanna D, Zaruba GV. A dynamic and distributed scatternet formation protocol for real-life Bluetooth scatternets. the 38th Hawaii International Conference on System Sciencs-2005; 2005; Hawaii, US: IEEE; 2005.
[59] Sun Microsystems I. Java technology overview. Java technology [cited 2003 December 11th]; Available from: http://java.sun.com/overview.html
[60] Sun Microsystems I. About java technology. The Java technology phenomenon [cited 2003 November 20th]; Available from: http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html
[61] Process JC. JSR 82: Java APIs for Bluetooth. 2006 [cited 2006 May 10th]; Available from: http://www.jcp.org/en/jsr/detail?id=82
[62] Intel Research, Volunteers. BlueCove. 2004-2008 [cited 2007 10 September]; 2.0.3:[Available from: http://sourceforge.net/project/showfiles.php?group_id=114020
[63] Foundation E. Eclipse. 2004 [cited 2006 June 25]; Available from: http://www.eclipse.org/
[64] EclipseMe. EclipseMe. 2008 [cited; Available from: http://eclipseme.org/docs/index.html

Appendix A:

Conference paper

The paper attached to this appendix has been published in the proceedings of the Australasian Telecommunications Networks and Applications Conference (ATNAC 2008), Adelaide, December 7-10 2008.

Appendix B:

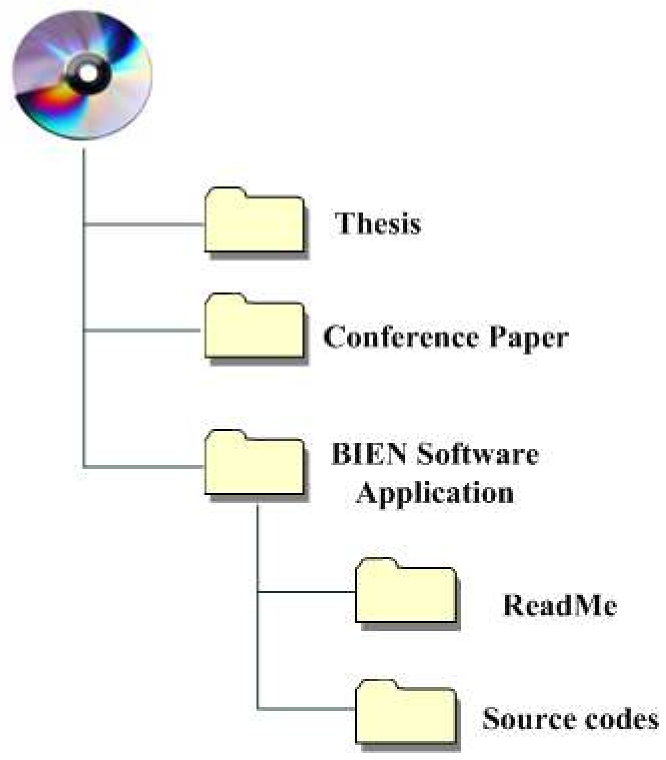
UML class diagram of the BIEN software application

This UML class diagram attached in this appendix was generated by Borland® Together® 2006 Release 2 for Eclipse.

Appendix C:

Thesis on CD-ROM

The CD-ROM attached to the back cover contains most of the files related to the research, including an electronic copy of this thesis and the conference paper in the appendix B, and the source codes of the BIEN software application. The structure of the CD contents is shown below:


Thesis
Conference Paper
BIEN Software Application
ReadMe
Source codes