An Effective Solution for Bluetooth Adhoc Networking

by

Sijun Jia

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

April 2009

Abstract

Bluetooth operates in the unlicensed ISM frequencies with a spread spectrum and frequency hopping, and it has the features such as robustness, low power consumption and low cost. Therefore, Bluetooth has been supported on the most number of devices of various types. Bluetooth connection is defined as in Piconet which has limits on range and the number of devices, so many researches have been done to connect more devices across longer ranges with Bluetooth, but there has not been an effective solution so far due to the protocol limitation, device versatility and mobility. In this thesis, we designed and implemented such a solution based on Piconet topology, existing Internet and adhoc networking protocols. In our solution, networking is performed on a high and abstract layer, so devices with different hardware or operating systems can join the network by installing a program without low level system change. We adjusted and implemented the standard Adhoc On-demand Distance Vector (AODV) protocol for IEEE 802.11 in our system to support route discovery and maintenance in Bluetooth network. We also used techniques such as pre-warm and redundant routes to improve the performance and robustness of the network. Our system support scalable peer to peer networking without any centralized controls. We tested our solution on real devices and on device emulators in large scale, and the result showed the system can form Bluetooth network efficiently in a scalable way.

Contents

List of Figures

List of Tables

Acknowledgement

Foremost, I would like to thank my supervisor, Prof. Son Vuong, who shared with me a lot of his expertise and research insight. This thesis would not have been possible without his support. I would like also to thank Prof. Charles Krasic for reviewing my thesis and providing great advices.

Sijun Jia

University of British Columbia

March, 2009

Chapter 1: Introduction

Bluetooth (IEEE 802.15.1) defines Personal Area Network (PAN) called Piconet, which enables up to 8 devices to form a master-slave network with the distance between two connected nodes to be limited within 10 meters. The Bluetooth specification also defines that a node can be shared by two or more Piconets which then will be bridged together through the shared node to form a Scatternet, but the specification does not define which node should be shared by which Piconets to form the most efficient Scatternet. The absence prompted many researches on Bluetooth Scatternet formation. While the existing researches brought many insights into the issue, they are short of providing an effective solution for the following reasons: first, Bluetooth devices may have very different hardware and operating system, so a solution is not effective if it assumes every device can be fully configured and controlled; second, Bluetooth devices have high mobility and Bluetooth network are sporadic and temporary, so a solution is not effective if it is not fully distributed or it involves large maintenance effort.

1.1 Motivation

The motivation of this thesis is to provide an effective solution on Bluetooth networking. Although Bluetooth is enabled on the most number of devices, the devices are not fully connectable because of the limitation of Bluetooth protocol. A Bluetooth network system is needed to connect all the devices in proximity and support corresponding applications. The system should be efficient and support many different devices with high mobility.

1.2 Contributions

This thesis has the following contributions:

- An innovated way to support networking on a high and abstract layer. The Bluetooth networking solution we provided depends on well defined interfaces that are independent from device hardware and operating systems, so devices from different manufacturers with different operating systems can connect with each other by simply installing a program. We think this way of networking is not just suitable for Bluetooth but also for other networks with versatile devices.
- Implemented adhoc on-demand networking (AODV) protocol for Bluetooth networking. AODV is fully distributed and the on demand route discovery eliminates heavy network initialization and maintenance. The route discovery is also relatively efficient considering that it results in multiple route updates for all the nodes on the path, and it also can avoid packet looping. The features provided by AODV are very important to Bluetooth networking.
- Utilized Bluetooth Piconet and Scatternet topology to pre-warm the network and enhance the network performance. Piconet and Scatternet topology provides efficient direct connections among neighbor nodes, so we utilized the direct connections to initialize and maintain the routing table on a node. This will significantly reduce the number of route miss and corresponding route discovery.
- Since our motivation is an effective solution for Bluetooth networking, we

implemented our solution as a program and test it on real devices and emulators. The program is easy to install and can support many different devices to form an efficient Bluetooth network. The program became an open source project which is hosted at the following site:

http://jban.dev.java.net

1.3 Thesis Outline

This thesis is comprised of six chapters. In chapter 2, we introduce related background knowledge on Bluetooth, IP networking and adhoc networking. When introducing a topic, we also give brief description of why or how it is related to or addressed in our solution. In chapter 3, we focus on existing researches on Scatternet formation. We analyze the researches and provide our suggestions. In chapter 4, we describe our effective solution in details. In chapter 5, we describe the testing result on real devices and device emulators. In chapter 6, the last chapter, we draw a conclusion to this thesis and describe some future work to extend our solution.

Chapter 2: Background

2.1 Bluetooth Technology

Bluetooth is a wireless protocol for short-range communications and intended to replace the cables. Bluetooth protocol operates on the secure, globally unlicensed Industrial, Scientific, and Medical (ISM) 2.4 GHz short-range radio frequency bandwidth. Bluetooth uses a radio technology called frequency hopping spread spectrum to avoid radio conflict. It chops up the data being sent and transmits chunks of it on up to 79 different frequencies.

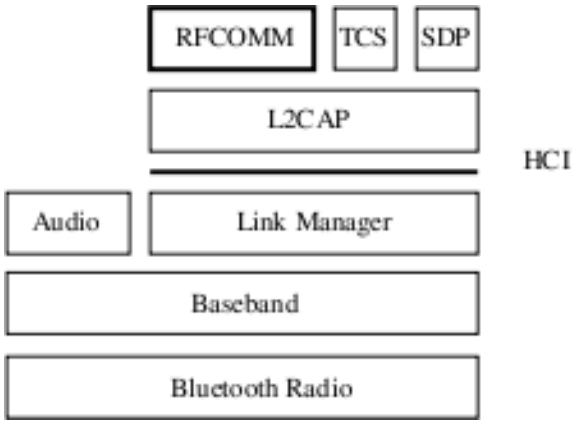

Figure 2 - 1: Bluetooth stack

Bluetooth is primarily designed for low power consumption, so it has a short range from 1 meter to 100 meters based on low-cost transceiver microchips in each device.

Class	Maximum Permitted Power mW(dBm)	Range (approximate)
Class 1	100 mW (20 dBm)	~100 meters
Class 2	2.5 mW (4 dBm)	~10 meters
Class 3	1 mW (0 dBm)	~1 meter

Table 2- 1: Bluetooth range

Currently, most Bluetooth enabled device uses class 2 Bluetooth, but more and more class 1 Bluetooth devices have been manufactured. Class 1 Bluetooth devices significantly increase the radio range of Bluetooth. Moreover, the range of class 2 devices can be extended if they connect to a class 1 transceiver and this is accomplished by the higher sensitivity and transmission power of Class 1 devices. Bluetooth bandwidth is also increasing:

Version	Data Rate
Version 1.2	1 Mbit/s
Version 2.0 + EDR	3 Mbit/s
WiMedia Alliance (proposed)	53 - 480 Mbit/s

Table 2 - 2: Bluetooth Bandwidth

The Bluetooth protocol stack has the following six layers:

1) Bluetooth Radio is the physical layer that defines radio frequency, frequency hopping, and modulation scheme and transmission power. The Bluetooth system operates in the 2.4 GHz ISM band (2400 - 2483.5 MHz), and it has 79 RF channels defined as: f=2402+k MHz, k=0 … 78. Its Modulation is GFSK (Gaussian Frequency Shift Keying).

2) Baseband defines connection establishment within a piconet, addressing, packet

format, timing, and power control. Two or more Bluetooth devices sharing the same physical channel form a piconet. One Bluetooth device acts as the master of the piconet, whereas the other devices act as slaves. Up to seven slaves can be active in the piconet. Additionally, many more slaves can remain connected in a parked state. These parked slaves are not active on the channel, but remain synchronized to the master and can become active without using the connection establishment procedure. Both for active and parked slaves, the channel access is controlled by the master. Piconets that have common devices are called a scatternet. Each piconet only has a single master; however, slaves can participate in different piconets on a time-division multiplex basis. In addition, a master in one piconet can be a slave in other piconets.

3) Link Manager Protocol (LMP) defines link management between Bluetooth devices including security aspects. Every Bluetooth device has a Link Manager which sends and receives LMP messages to Link Managers on other devices. Link Manager interprets and filter out LMP messages and acts on the messages to set up and control the direct link between two devices. LMP messages are not propagated to higher layers on a device.

4) Host Controller Interface (HCI) defines an interface to the base band controller and Link Manager so that the base band capabilities can be accessed with uniform methods for different devices. Standard commands and events are defined to be sent between the host and the Bluetooth controller.

5) Logical Link Control and Adaptation Protocol (L2CAP) supports higher level protocol multiplexing, packet segmentation and reassembly, and the conveying of

quality of service information. L2CAP permits higher level protocols and applications to transmit and receive upper layer data packets (L2CAP Service Data Units, SDU) up to 64 kilobytes in length. L2CAP also permits per-channel flow control and retransmission via the Flow Control and Retransmission Modes. L2CAP provides a multiplexing role allowing many different applications to share the resources of an ACL-U logical link between two devices.

6) Service Discovery Protocol (SDP) defines the discovery of Bluetooth devices and services. The service discovery mechanism provides the means for client applications to discover the existence of services provided by server applications as well as the attributes of those services. The attributes of a service include the type or class of service offered and the mechanism or protocol information needed to utilize the service. The Bluetooth data transportation architecture including physical link and logical link is shown in figure 2-2

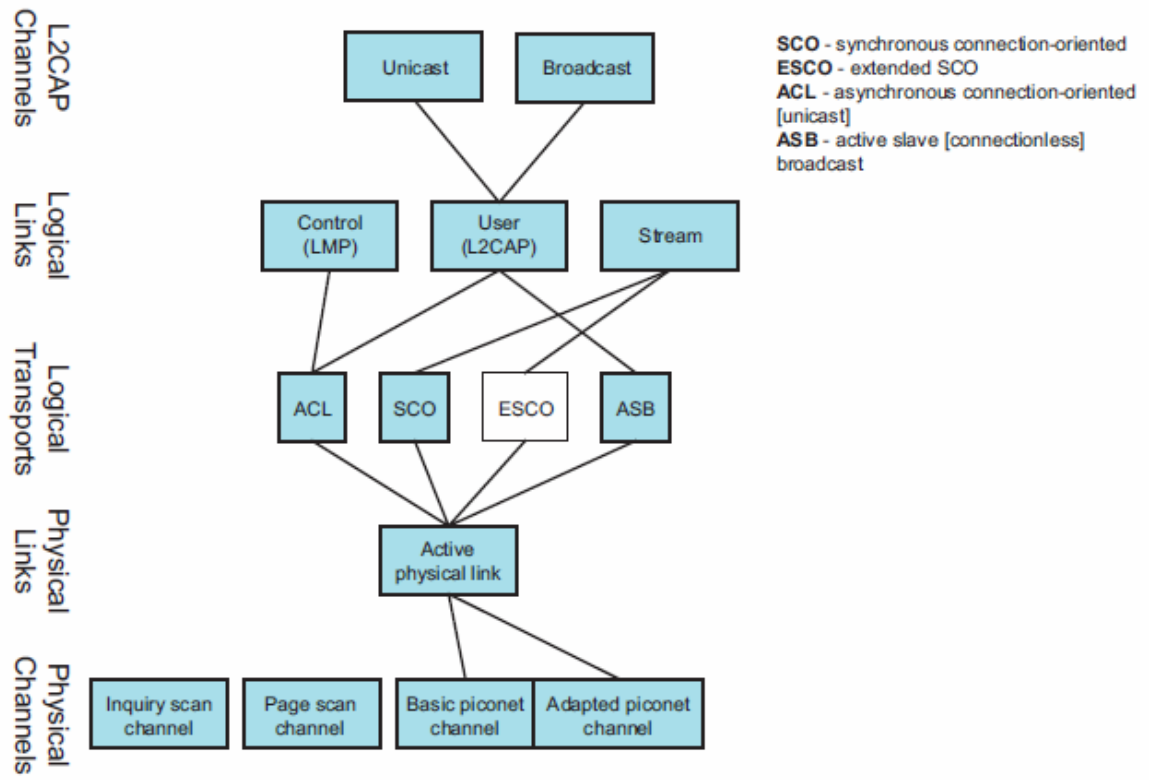

Figure 2 - 2: Overview of transport architecture entities and hierarchy

The general Bluetooth packet structure is shown in figure 2-3

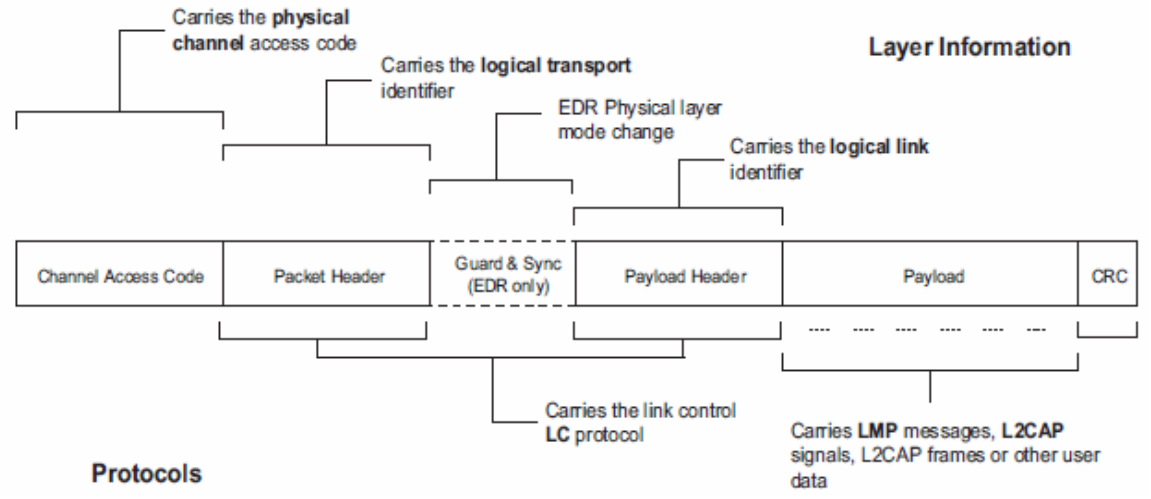

Figure 2 - 3: Bluetooth packet structure

Table 2-3 lists all of the logical transport type v s, the supported logical link types, which type of physical links and physical channels can support them, and a brief description of the purpose of the logical transport. As it will be discussed later that one device can have up to 7 active slave nodes but there is no such limit for parked slave nodes.

Logical transport	Links supported	Supported by	Overview
Asynchronous Connection-Oriented (ACL)	Control (LMP) ACL-C User (L2CAP) ACL-U	Active physical link, basic or adapted physical channel	Reliable or time-bounded, bi-directional, point-to-point.
Synchronous Connection-Oriented (SCO)	Stream (unframed) SCO-S	Active physical link, basic or adapted physical channel	Bi-directional, symmetric, point-to-point, AV channels. Used for 64Kb/s constant rate data.
Extended Synchronous Connection-Oriented (eSCO)	Stream (unframed) eSCO-S	Active physical link, basic or adapted physical channel	Bi-directional, symmetric or asymmetric, point-to-point, general regular data, limited retransmission. Used for constant rate data synchronized to the master Bluetooth clock.
Active slave broadcast (ASB)	User (L2CAP) ASB-U	Active physical link, basic or adapted physical channel.	Unreliable, uni-directional broadcast to any devices synchronized with the physical channel. Used for broadcast L2CAP groups.
Parked slave broadcast (PSB)	Control (LMP) PSB-C, User (L2CAP) PSB-U	Parked physical link, basic or adapted physical channel.	Unreliable, uni-directional broadcast to all piconet devices. Used for LMP and L2CAP traffic to parked devices, and for access requests from parked devices.

Table 2 - 3: Logical transport types

Bluetooth also has two application level protocols:

1) Radio frequency communications (RFCOMM) protocol is used to create a virtual serial port to transport data like a cable. RFCOMM is also called cable replacement protocol. RFCOMM emulates EIA-232 (formerly RS-232) and is on top of L2CAP layer.

2) Telephony control protocol-binary (TCS BIN) defines the establishment of voice and data calls between Bluetooth devices. It is also on top of L2CAP layer.

From the Bluetooth protocol layers, we can see L2CAP, comparable to IP in wired networking, is the right choice for networking. L2CAP gives us enough device abstraction and at the same time support critical application protocols such as SDP, RFCOMM and TCS BIN.

2.2 IP Network

Internet Protocol has long been established on wired networking especially on the Internet. IP packets are routed among routers from network to network to get to its destination. Some IP principles are good for all networking, and should be followed in Bluetooth networking as well, but because of the significant difference between Bluetooth networking and IP networking, they have to be adjusted. Here we will study two mostly used IP routing protocols to see how we can use them and what changes we should make.

1) Distance vector routing protocol

In distance vector routing protocol, every router maintains its own routing table and inform any topology changes to its neighbors periodically. Every neighbor that receives the information will in turn amend its own routing table and inform changes to its own neighbors until the process converge which means every node in the network knows the changes. Examples of distance vector routing protocol include RIP, IGRP, EGP and BGP. Since a router only advertises changes to its neighbors, distance vector routing protocol has less computational complexity and message overhead. Looping, also known as count-to-infinity problem is a serious limitation to the distance vector protocol. Count-to-infinity will happen when two nodes get from each other the routing information such as of number of hops toward a destination. One way to deal with the count-to-infinity problem is to use a maximum number of hops, but it will limit the number of hops when there are not loops. We partially use distance vector routing protocol in our Bluetooth routing protocol because of its less message overhead and computational complexity, since the Bluetooth devices usually have small computation powers or limited battery time.

We set a maximum number of hops to avoid looping, but this won't be a significant limitation because in the unstable and dynamic Bluetooth network, a route contains many hops is not a robust one and the maximum number of hops should be set even when there is no looping. To avoid message overhead, we further set a maximum number of hops to advertise routing changes. After the maximum number is reached, a receiving node will update its own routing table but will not continue to advertise the information. Here we give up on converging, but since we also use dynamic routing protocol in our protocol, the change can be discovered later by the nodes outside of the

maximum distance. Overall, we use distance vector routing protocol in our protocol and greatly adapt it to suit a Bluetooth network of small mobile devices.

2) Link state routing protocol

The basic concept of link-state routing is that every node constructs a map of the connectivity of the network, in the form of a graph showing which nodes are connected to which other nodes. Each node independently runs an algorithm over the map to determine the shortest path from itself to every other node in the network; generally some variant of Dijkstra's algorithm is used. Examples of link-state routing protocols include OSPF and IS-IS. While link-state routing such as OSPF is more often used in the Internet routing, it is not appropriate for Bluetooth networking because every router is a small device which does not have either the computation power or the storage for knowledge about the whole network, and again since the network is volatile, there is a big overhead to maintain a consistent link state of the whole network. In our protocol, we will keep as much global knowledge as possible, for example the routes discovered in routing discovering processes, but it will not try to get the complete knowledge about the whole network. Another related idea is to use the existing global network topology knowledge as sample data and use machine learning technology to predict the global network connectivity.

2.3 Mobile Adhoc Network

Mobile Adhoc Network (MANET) usually refers to wireless network with mobile nodes connect with each other through WIFI (802.11) technology. MANET has the following characteristics:

1) Dynamic topologies: Nodes are free to move arbitrarily; thus, the network topology--which is typically multihop--may change randomly and rapidly at unpredictable times, and may consist of both bidirectional and unidirectional links.

2) Bandwidth-constrained, variable capacity links: Wireless links will continue to have significantly lower capacity than their hardwired counterparts. In addition, the realized throughput of wireless communications--after accounting for the effects of multiple access, fading, noise, and interference conditions, etc.--is often much less than a radio's maximum transmission rate. One effect of the relatively low to moderate link capacities is that congestion is typically the norm rather than the exception, i.e. aggregate application demand will likely approach or exceed network capacity frequently. As the mobile network is often simply an extension of the fixed network infrastructure, mobile ad hoc users will demand similar services. These demands will continue to increase as multimedia computing and collaborative networking applications rise.

3) Energy-constrained operation: Some or all of the nodes in a MANET may rely on batteries or other exhaustible means for their energy. For these nodes, the most important system design criteria for optimization may be energy conservation.

4) Limited physical security: Mobile wireless networks are generally more prone to physical security threats than are fixed-cable nets. The increased possibility of eavesdropping, spoofing, and denial-of-service attacks should be carefully considered. Existing link security techniques are often applied within wireless networks to reduce security threats. As a benefit, the decentralized nature of

network control in MANETs provides additional robustness against the single points of failure of more centralized approaches.

5) The need for scalability is not unique to MANETS. However, in light of the preceding characteristics, the mechanisms required to achieve scalability likely are.

These characteristics create a set of underlying assumptions and performance concerns for protocol design which extend beyond those guiding the design of topology of the traditional fixed Internet. Bluetooth network has similar characters as MANET, so a protocol for Bluetooth network will have similar assumptions and performance concerns. However MANET protocols cannot be used directly in Bluetooth adhoc network, because Bluetooth adhoc network has some special characteristics:

1) Bluetooth device has topology limitations. One device cannot connect to too many devices directly and the direct connection distance is limited. While these limitations are balanced with battery saving and free licensing, it is more challenging to design an efficient network protocol for Bluetooth network.

2) Bluetooth adhoc network is more dynamic than MANET because usage of Bluetooth network is more sporadic and Bluetooth devices such as cell phones and PDAs are smaller than laptop computers (which can support Bluetooth too) and easier to be carried around. The protocol must handle the case that a device connect to the network briefly, download some data or perform a transaction and leave the network.

3) Bluetooth has smaller bandwidth than WIFI and most Bluetooth devices have

very limited computation power and storage than WIFI devices. Bluetooth consume much less energy than WIFI, but its bandwidth is smaller than that for WIFI. Bandwidth of class 2 Bluetooth devices is about 1mbps while Bandwidth of WIFI devices is up to 100 mbps. One effect of the bandwidth limitation is that a network bottleneck is more likely formed in a Bluetooth network and it will significantly affect the efficiency of the network.

In conclusion, Bluetooth adhoc network is a MANET but with its special characters. When we design a Bluetooth adhoc network protocol, MANET network protocols are very good references and Bluetooth specialty should also be considered.

There are two major MANET protocols:

1) Ad hoc On-Demand Distance Vector (AODV) Routing

AODV enables dynamic, self-starting, multihop routing between participating mobile nodes wishing to establish and maintain an ad hoc network. AODV allows mobile nodes to obtain routes quickly for new destinations, and does not require nodes to maintain routes to destinations that are not in active communication. These characteristics distinguish AODV from Distance Vector algorithm from fixed IP networking, which requires every router to make routing advertisement and maintains a routing table based on the advertisement until it converge. AODV allows mobile nodes to respond to link breakages and changes in network topology in a timely manner.

Three types of message are used in AODV: Route Request (RREQ), Route Response (RREP) and Route Error (RERR). The node that is looking for a route to a destination will send a RREQ to its known nodes, the nodes will continue to broadcast

the message until either the destination of a node having *fresh* information of the destination is hit. Then a RREP will be sent back to the original requesting node, and the routing table in the nodes along the path will all get updated. Every node in AODV will watch its neighbor and broadcast an RERR if it detects any failed node. A very important parameter in AODV is the sequence number in the routing table, which avoid message looping and also helps to determine the *freshness* of the information about the destination on a node. Every node will keep a sequence number for every destination in its routing table, and maintain this destination sequence number upon receiving a networking message RREQ, RREP or RERR. When a node sends RREQ it increases the sequence number and when the destination node sends back a RREP, it uses the larger number between its own sequence number and the one in the RREQ. A larger sequence number is fresher than the smaller one. A number trick is used to rollover the sequence number: the number is increased as an unsigned number and used as a singed number.

AODV is a reactive wireless protocol which is used in the Bluetooth network protocol we suggest because it provides an efficient way for route discovery and keeps routing messages to the minimum by fully utilize the information contained in the messages. Many valid routes are formed in the process of discovering a single route.

2) Optimized Link State Routing Protocol (OLSR)

OLSR is an optimization of the classical link state algorithm tailored to the requirements of a mobile wireless LAN. The key concept used in the protocol is that of multi point relays (MPR). MPRs are selected nodes which forward broadcast messages during the flooding process. This technique substantially reduces the message overhead as compared to a classical flooding mechanism, where every node retransmits each

message when it receives the first copy of the message. In OLSR, link state information is generated only by nodes elected as MPRs. Thus, a second optimization is achieved by minimizing the number of control messages flooded in the network. As a third optimization, an MPR node may chose to report only links between itself and its MPR selectors. Hence, as contrary to the classic link state algorithm, partial link state information is distributed in the network. This information is then used for route calculation. OLSR provides optimal routes (in terms of number of hops). The protocol is particularly suitable for large and dense networks as the technique of MPRs works well in this context.

The concept of MPR is useful to Bluetooth network too because the optimizations are necessary for an efficient Bluetooth network. Devices in a Bluetooth network are quite different: some nodes have lots of bandwidth and computation power while some have little; some nodes are static while some are dynamic; some nodes are connected to the Internet while some aren't. The nodes with higher capability should be chosen as MPR in a Bluetooth network. Another point is that dense networks are common because of the limited connection range among Bluetooth devices and the technique of MPR will be very helpful.

There are some other MANET protocols such as Dynamic Source Routing (DSR), Open Shortest Path First Routing (OSPF), Destination-Sequenced Distance-Vector Routing (DSDV) and Predictive Wireless Routing Protocol (PWRP). Most of these protocols are similar with the above two major protocols, but with their own optimizations. All these protocols are initially designed for WI-FI mesh networking on 802.11, but they also can be used in Bluetooth adhoc network, and Bluetooth adhoc

network is essentially a Bluetooth mesh network. Bluetooth adhoc network can also connect to the Internet or a WI-FI mesh network via gateways.

2.4 Sensor Network

With its low energy consumption and low cost, Bluetooth is a good choice of communication technology for sensor network. Bluetooth enabled sensors can form adhoc networks to transfer environment data. The Bluetooth adhoc network protocol we suggest can support sensor networks efficiently. Since Bluetooth is supported on many devices such as laptops, cell phones or printers, it is convenient to collect data or administrate a Bluetooth enabled sensor network using these devices. The Bluetooth enabled sensor network can use various Bluetooth enabled device as a gateway to connect to other networks.

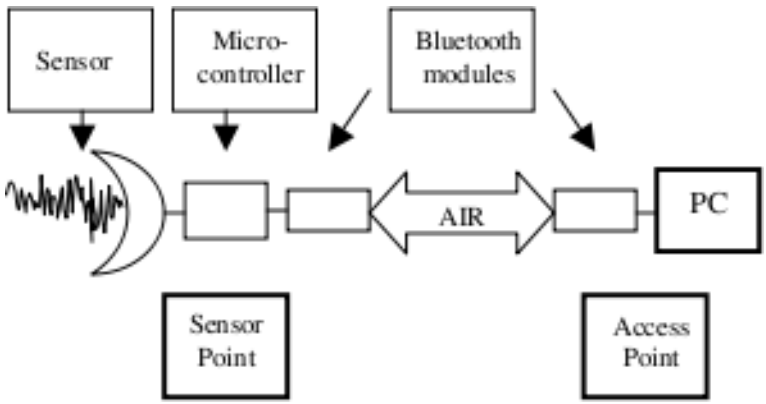

Figure 2 - 4: Bluetooth Sensor Network

Chapter 3: Bluetooth Network

3.1 Piconet and Scatternet

Direct Bluetooth link is formed through a piconet. A piconet consists of two and up to seven devices that are synchronized to a common clock and hopping sequence, which is also called a channel. The common (piconet) clock is identical to the Bluetooth clock of one of the devices in the piconet, and this device is called the master node of the piconet, and the hopping sequence is derived from the master's clock and the master's Bluetooth device address. All other synchronized devices in the piconet are referred to as slave nodes. A number of independent piconets may be interconnected to form a network called scatternet. The piconets can be connected via shared nodes. The shared node participates concurrently in two or more piconets on a time-division multiplexing basis. The shared node can be a master node in one piconet and a slave node in other piconets, or a slave node in more than one piconet, but it cannot be the master node on more than one piconet because the piconet is defined by synchronization to the master's Bluetooth clock.

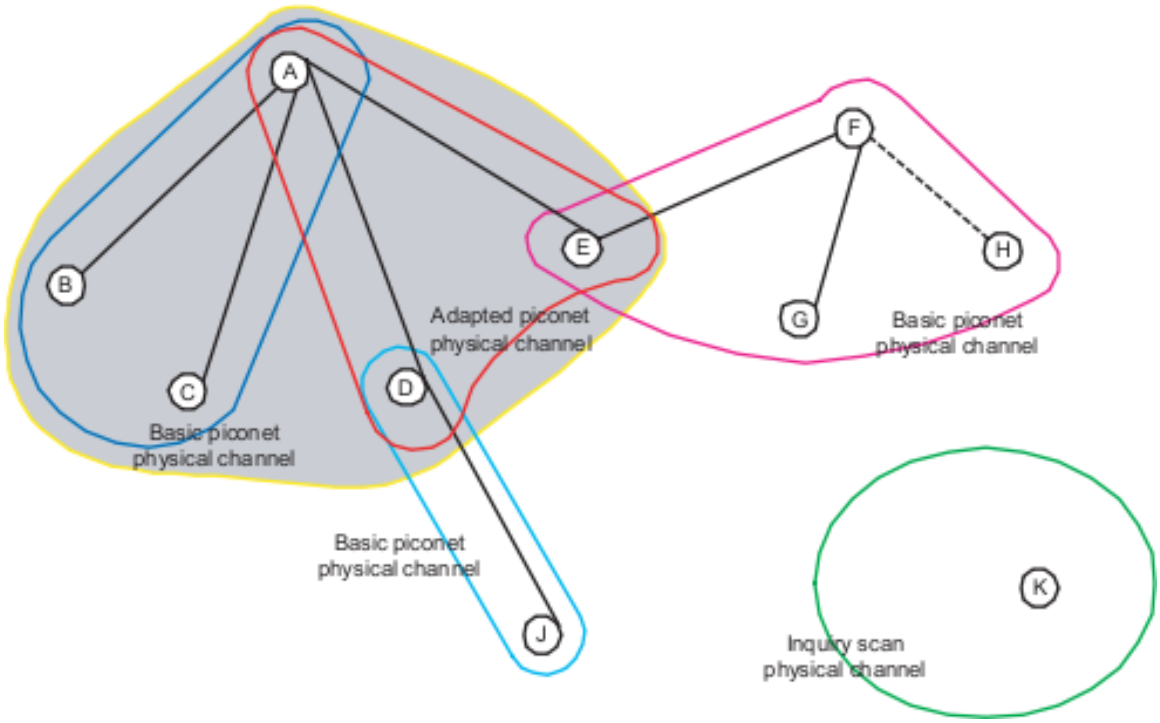

Figure 3 - 1: Bluetooth network: piconet and scatternet

There are two procedures to set up a Bluetooth connection: inquiry (or discovering) procedure and paging (or connecting) procedure. Bluetooth devices use the inquiry procedure to discover nearby devices, or to be discovered by devices in their locality. The inquiry procedure is asymmetrical. A Bluetooth device that tries to find other nearby devices is known as an inquiring device and actively sends inquiry requests. Bluetooth devices that are available to be found are known as discoverable devices and listen for these inquiry requests and send responses. The inquiry procedure uses a special physical channel for the inquiry requests and responses. Both inquiring and discoverable devices may already be connected to other Bluetooth devices in a piconet. Any time spent inquiring or occupying the inquiry scan physical channel needs to be balanced with the demands of the QoS commitments on existing logical transports. The inquiry procedure

does not make use of any of the architectural layers above the physical channel, although a transient physical link may be considered to be present during the exchange of inquiry and inquiry response information. The procedure for forming connections is asymmetrical and requires that one Bluetooth device carries out the page (connection) procedure while the other Bluetooth device is connectable (page scanning.) The procedure is targeted, so that the page procedure is only responded to by one specified Bluetooth device. The connectable device uses a special physical channel to listen for connection request packets from the paging (connecting) device. This physical channel has attributes that are specific to the connectable device, hence only a paging device with knowledge of the connectable device is able to communicate on this channel. Both paging and connectable devices may already be connected to other Bluetooth devices in a piconet. Any time spent paging or occupying the page scan physical channel needs to be balanced with the demands of the QoS commitments on existing logical transports.

During or after the two procedures, the device can be in the following mode and state:

- Connected mode. In this mode, the two devices are physically connected within a piconet.
- Hold mode. In this mode, the physical link is only active during slots that are reserved for the operation of the synchronous link types SCO and eSCO.
- Sniff mode. In this mode the availability of ACL logical transports is modified by defining a duty cycle consisting of periods of presence and absence. Devices that have their default ACL logical transports in sniff mode may use the absent periods to engage in activity on another physical channel, or to enter reduced power mode.

- Parked state. A slave device may remain connected to a piconet but have its physical link in the parked state. In this state the device cannot support any logical links to the master with the exception of the PSB-C and PSB-U logical links that are used for all communication between the piconet master and the parked slave. When the physical link to a slave device is parked this means that there are restrictions on when the master and slave may communicate, defined by the PSB logical transport parameters. During times when the PSB logical transport is inactive (or absent) then the devices may engage in activity on other physical channels, or enter reduced power mode.

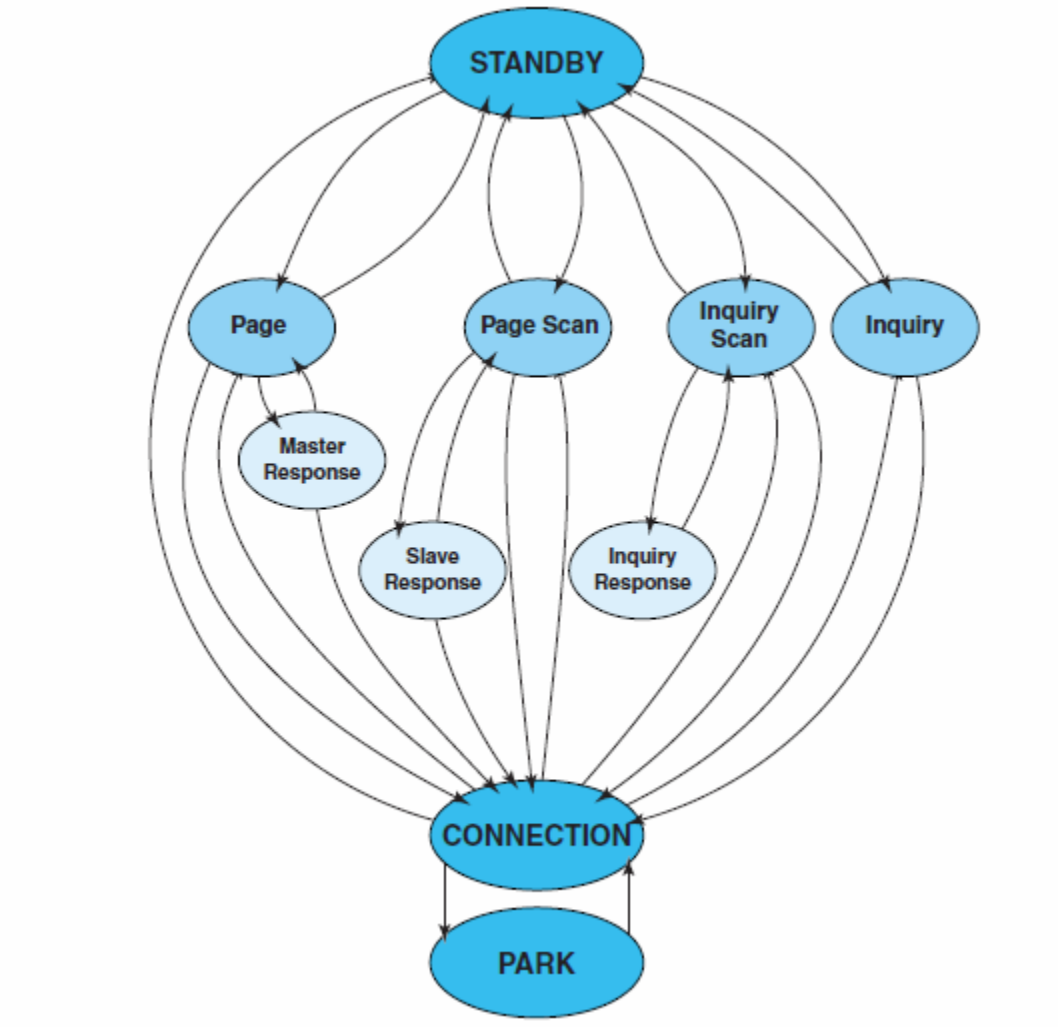

Figure 3 - 2: State transition of link controller

3.2 Scatternet Formation Algorithms

The Bluetooth specification does not define the routing capability of the shared node, and Bluetooth core stack of the shared Bluetooth device does not support routing. Instead the Bluetooth specification considers routing the responsibility of higher level protocols. There are many researches have been done on scatternet formation focusing on routing performance, scalability and stability. There are two basic measures for the

complexity of a scatternet formation algorithm [1]:

- Time complexity – amount of time needed to form a scatternet
- Message complexity – number of messages sent among devices to form a scatternet

Also, because of the mobility nature of the devices, the scatternet is very dynamic. So it is critical to evaluate the maintenance complexity, which includes the amount of time and the number of messages needed when new node joins or an existing node leaves or two existing scatternets merge into one new scatternet. A good algorithm should keep the balance of formation and maintenance complexity and network performance.

As in all networks, bandwidth or throughput, packet lost, packet delay (RTT) and reachability can be checked to see the performance of a scatternet that is formed using a specific algorithm. Battery life time of the devices can also be checked to see the over all energy consumption. Statistical study concludes that the amount of bridging overhead and the number of links have a major impact on the performance of a scatternet [2]. From *topology* perspective, the following measures are good indicators of the performance [1]:

- Number of piconets – all piconets shares the 79 physical channels, so there will be more collisions with more piconets. As shown in [3], the burst failure rate increases with the number of piconets. More piconets also mean more hops for network packets.
- Degree of the devices – a device shared by two or more piconets will switch among different channels which greatly decrease the performance of the device and will become the bottleneck for the scatternet. Both maximum degree of one of

the devices and the sum degree of all devices can be used as the measure.

- Network diameter – network diameter indicate number of hops of packets and the latency.

The above measures should be minimized. One simple rule is to avoid create new piconet whenever possible and to connect to the node with lower degrees.

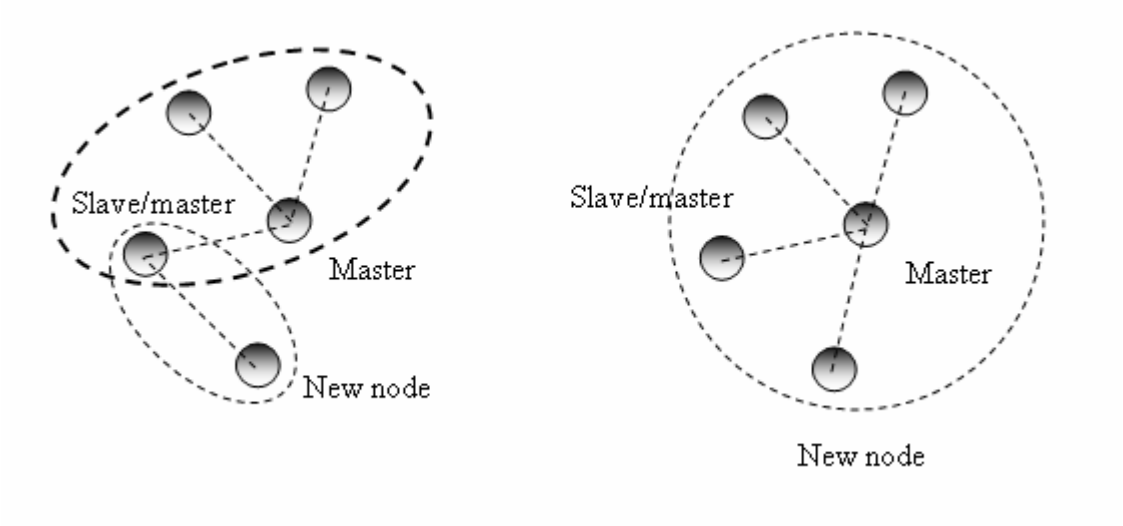

Figure 3 - 3: Two different ways to form scatternet

There are some other factors also needed to be considered when a new node is choosing an exiting node as a bridge to join the network. Firstly [5][6], computation capabilities are dramatically different among Bluetooth devices, such as those of a laptop and a cell phone, so a good algorithm should weigh the capability and give bias to the node with higher capability. Secondly [5], Bluetooth network usually is a sub network which can connect to other networks, such as WIFI mesh network or the Internet, via access point. If a large amount of network traffic is targeted to the external network, the new node should connect to an existing node with the least distance to an access point.

Thirdly, stability or mobility of an existing node should be a big concern. It is very inefficient to connect to a node that may disappear very soon. The existing node should suggest its life time so the new node can make decision based on that.

The existing scatternet formation algorithms fall into two categories: in the first category, which includes the algorithms proposed in the following research papers [1] [2] [7] [8] [9] [10] [11], it is assumed that all Bluetooth devices are within transmission range of each other, so any two devices are able to connect directly with each other with Bluetooth. We call this assumption single-scope assumption. The second category are the algorithms without the single-scope assumption, which includes the algorithms in [12] [13] [14] [15] and this thesis.

The single-scope assumption is too strict and gives up one of the main purposes of scatternet, which is to increase network reachability and enlarge network scope. Especially, the algorithms with such assumption may not work effectively for the class 2 Bluetooth devices, which are the most common devices and only have a transmission scope of 10 meters. The assumption is reasonable for class 1 Bluetooth devices which have 100 meters transmission scope, but with such devices, we may expect a network with much greater reachability, such as a wireless mesh network. However, the algorithms in this category focus on the single scope and are good references for solving the problems in one part of a large network.

Aggarwal et al. [8] introduce a two-phase scatternet clustering algorithm that partitions the network into independent piconets. A super-master, which knows about all the nodes, is elected to organize the formation of full piconets and minimize the number of piconets. A reorganization process is then needed to interconnect the piconets. We

think an algorithm that requires a super-master violates the distribution principle of adhoc networking, and it is time consuming and non-practical to elect a super-master. It is efficient to minimize the number of piconets

Salonidis et al. [9] introduce a randomized point-to-point symmetric protocol that yields link establishment delay with predictable statistical properties, and then propose the Bluetooth Topology Construction Protocol (BTCP) which extends the symmetric mechanism to the case of several nodes. BTCP requires a coordinator or super-master to be elected and the coordinator determines the role, either master or slave, of each node in the scatternet. The result is a fully connected scatternet with the maximum number of devices limited to 36.

Tan et al. [10] propose a Tree Scatternet Formation (TSF) algorithm that builds scatternets by connecting nodes into a tree structure. Master/slave roles are dynamically assigned to each node. However, tree structure is not good for networking because it is not efficient and error prone. When a node fails or leaves the scatternet, its subtrees will get isolated from the network and have to change the topology to re-connect.

Law et al [1] propose to partition devices into components. Every component can be a single device, a piconet or a scatternet. Each component has one leader, and as the component expands, the old leader will retire when it is saturated and a new leader will be assigned and the component will continue to expand. Piconets are merged to get minimum number of piconet, and nodes are moved to other piconet to retire its leader. The result scatternet is optimized, but the merging and moving of node among piconets are expensive and will cause disruption.

The algorithms in the second category, which do not require single-scope, are more practical and valuable. These algorithms are distributed, which is the key feature of adhoc networking.

Basagni et al. [14] propose the BlueTrees protocol. The BlueTrees protocol designates a node called the *blueroot* that initiates the protocol and builds a treelike scatternet rooted at itself. Then, a re-configuration procedure bounds the number of slaves per master such that, at the end of this procedure, each master has no more than seven slaves. This tree-like topology limits the robustness of the scatternet generated by this protocol. In addition, BlueTrees depends on the blueroot to initiate the protocol; this creates a single point of failure, and if the network is not connected after the device discovery phase then this solution will not work. A tree structure is not efficient for network communication and it will increase the average network distance (i.e. number of hops) and the blueroot is also a bottleneck of network traffic.

Basagni et al. [12] also propose the Bluestars protocol. The BlueStars protocol has three phases. In the first phase, devices discover each other. In the second phase, piconet formation is initiated by all nodes in a distributed manner wherein each node decides to be a master or a slave based on a locally computed weight. The third phase generates the scatternet by interconnecting the piconets through gateway nodes selected by each piconet.

Zhen et al. [15] present a two-phase scatternet formation algorithm: in the first phase, all the nodes are self-organized into “bluestar islands” that consist of piconets interconnected by a joint slave node. In the second phase, the “bluestar islands” are bridged by means of a routing trigger to form a fully connected network. The routing

trigger is implemented by means of route request messages, that have been used in on-demand routing protocols.

Stojmenovic [13] proposed a dominating-set based three-phase scatternet formation protocol for Bluetooth networks. In the first phase, all neighbors within transmission range are discovered, the unit graph is constructed and a localized sparse subgraph is constructed. In the second phase, the subgraph construct is applied simultaneously on all nodes with excessive degree, to limit the degree of each node to 7. In the third phase, master-slave roles are assigned based on dominating set membership.

3.3 Our suggestions on scatternet formation

It is obvious that the piconet topology of Bluetooth is an additional feature for forming network, but *not* a limitation. The performance of Bluetooth adhoc network is not solely determined by the topology, or an ideal topology is not necessary for a sound Bluetooth adhoc network.

To achieve ideal topology, all the existing protocol listed in 2.5.2, make several assumptions which are not realistic, such as one-scope or single hop, or a super-master, or certain deterministic knowledge that is not achievable. Besides these unrealistic assumptions, all the algorithms also assume the Bluetooth chip has been designed to function in the way the algorithms require. Given the versatility of Bluetooth devices, which can be an ear phone, a printer or a computer, it is impractical for all the Bluetooth chips to have the same behaviors and functionalities. Therefore, it is not practical and contra productive to construct an ideal topology. The mobility of Bluetooth devices makes an idea topology even more unnecessary and impractical.

Since it is unnecessary and impractical to construct an ideal scatternet topology, we build our solution to Bluetooth adhoc network on a layer above the scatternet formation or the topology construction layer. However, this does not mean that the scatternet topology is not important. A sound topology will still make the network more efficient, but simple random direct link topology should still be tolerated and traffic should be adjusted correspondingly.

Our suggestions on scatternet formation are:

1) The scatternet formation should be distributed and should not need any global information or management. The temporary local topology is more important than the optimized global structure.
2) The scatternet formation should be a lazy process such as the one shown in [15], which is the only one in the researches we list above. The reason is that a heavy initial scatternet formation process will cause a long delay, and such an initial process is usually heavily depended on unrealistic assumptions such as some global information. The result network of such initial process may only exist for a short period of time because the mobility of the devices.
3) The Bluetooth protocol should define HCI command for host to inquire a local controller or a remote device about the scatternet topology of the device. Currently, there is no such command. The host can analyze the topologies from multiple nodes and make a decision which node to connect. This mechanism is necessary to make the algorithms at scatternet level possible. The host is on the high level and has more information to make a decision such as if the node should be a master or slave.

We base the above suggestions on reasoning and our study on existing research papers, and we will not design a specific algorithm or a system to prove our suggestions to be right since this is not our solution for Bluetooth adhoc networking. Our solution, as explained in the following chapters, is on a higher level than the scatternet level. Scatternet formation is an important process for Bluetooth networking, and our solution will take the advantage if a sound scatternet formation algorithm is implemented.

Chapter 4: Bluetooth Adhoc Network

4.1 Networking on the network layer

All the researches listed above on scatternet formation and Bluetooth adhoc networking provide logical analyses, mathematic models and algorithms to solve this issue, but none of them provides a practical solution. Most of the researches used simulation result to prove the correctness of their algorithms, but none of them provides a program that can actually support Bluetooth networking. One reason for this absence of solution is that the current researches focus on scatternet topology, which is in the data link layer and physical layer in the OSI 7-layer model, to tackle Bluetooth networking issue. We think it is much more effective to do networking in the network layer. Another reason is researchers intend to leave IP networking or MANET adhoc networking to solve the issue in the network layer. However, because of the characteristics of Bluetooth networking, as we mentioned when introduce IP networking and MANET, using these networking protocols directly are not efficient for Bluetooth networking.

Therefore, we propose a solution to Bluetooth networking on the network layer, which is L2CAP layer in Bluetooth protocol stack. The solution will consider the underlying Bluetooth piconet or scatternet topology, but mostly use a combination of modified IP networking and MANET to solve Bluetooth networking problem. We also implement our algorithm in open source software, and we test the software in emulators and real devices to prove the efficiency of our algorithm.

4.2 Bluetooth Adhoc Network (BAN)

Our solution, Bluetooth Adhoc Network (BAN) algorithm has the following characteristics:

1) It is on the networking layer, i.e. L2CAP layer. Like IP, L2CAP use data packet to transfer data and so it is natural to route the packet on this layer. L2CAP is above the hardware layer, and thus provide abstraction above the versatile Bluetooth hardware. L2CAP supports other higher layer protocols such as RFCOMM, TCS and SDP so the multi-hop connections are achieved transparently for these protocols if L2CAP packets can be routed through multi-hop in the network.

2) It is adhoc, on-demand. It is lazy and reactive. It will not try to form a complete network; instead, a node sends RREQ when it needs to send a packet to a destination but cannot find a route in its routing table. RREP and RERR packets are used for response and error response. A sequence number is used to avoid routing packet looping.

3) It is fully distributed. All nodes are peers and there is not any central node or super node. Every node should have routing information and do routing based on it. However, the node with higher capability of transportation, computation or storage may take more responsibility on routing.

4) It has an initial proactive stage, in which a new node will learn routing information from its direct linked neighbors. Since it is direct piconet link, it is efficient for the new node to retrieve the information. Because of the topology limit, the new node can only have less than 7 direct linked nodes, so there is not

too much information to be transferred to the node. The information forms the initial routing table for the new node and pre-warms the node. It happens following the discovery stage so it will not add too much overhead since the devices already formed physical links.

5) It keeps redundant routing path information. Because of the mobility of the devices, every node is unpredictable, and a routing path may fail very quickly because one or more nodes on the path leave or fail. In that case, the redundant routing path if any will be used.

6) The routing distance is not just the number of hops. It will be a weighted distance based on the statistic properties of the nodes on the path. The properties include the bandwidth, life time, computation capacity (such as CPU speed and memory size) of a node etc. This takes in consider the versatility of Bluetooth devices.

7) It uses device physical address instead of IP address. The device address can uniquely identify a device. Since every node has its own routing capability, IP address is not necessary for routing. This will eliminate IP address management overhead as well.

The Bluetooth Adhoc Network is formed in the following stages:

1) Device Discovery and Direct Connection

A Bluetooth device will start a discovering process under the following conditions:

- The device is just started.

- The device has been started but it lost direct connections with a certain number of its neighbor devices. For example a device can be configured to start a discovering process when 50% of its neighbor devices have lost connection with it.

The device will try to connect all the neighbor devices it discovered. The number of neighbor devices the device can connect to and the number of piconets the device can participate depend on the capability of the device and the physical scatternet topology. As explained before, a device can be a slave node in multiple piconets but can only be a master node in one piconet. Some devices cannot switching roles between two piconets and thus can involve in one piconet. BAN does not control the formation of scatternet or the topology because for existing devices that can only be controlled in physical layer. We suggest the Bluetooth spec to define higher level interfaces to control scatternet formation. If the control interface is available, BAN should make decision based on the algorithm we suggested above and some other factors such as device capability. Once connected to its neighbor devices, the neighbor devices will record the new device in their routing tables and send their routing information to the new device. Depending on the size of its routing table, a neighbor may only send part of its routing table to the new device, and the part should contain the routes to the closest nodes; for example, the least number of hops or weighted distance. The new device will receive all the routing information and save them in its routing table. Redundant routing information will be queued based on the distance in the routing table of the new device, and the longer redundant routes will be used if the shorter one fails. The redundant route may help to avoid a route discovery process, which needs broadcasting and thus is expensive. It is

efficient for the routing information to be transferred between direct-linked devices. Also, a device cannot connect too many devices directly because of the limit of Bluetooth topology so the initial transferring of routing information won't take a long time. After this process, we say the new device has joined in the network and its routing table will contain the information about all its neighbor devices and all or part of the information contained in the routing table on these devices. When the device needs to send a packet to another device, it checks its routing table and send the packet if it finds a route.

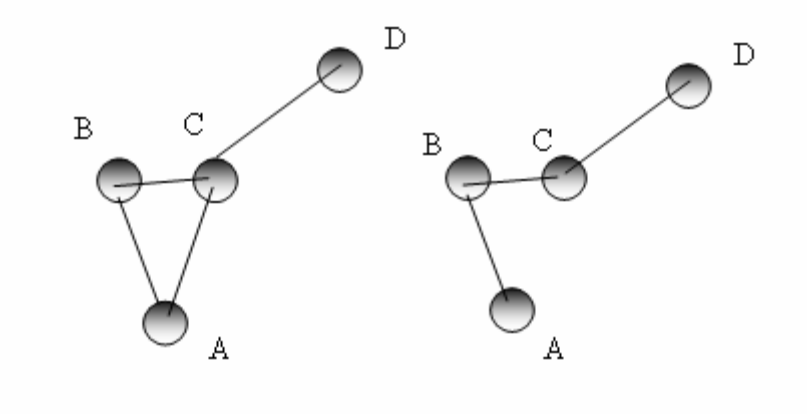

Figure 4 - 1: Redundant routes

Redundant routes make Bluetooth adhoc network more robust. When the link between A and C fails, which happens often in Bluetooth adhoc network, a packet still can go from A to D without a new route discovery process.

2) On-Demand Route Discovery

If the device cannot find a route in its routing table for a destination, or errors have happened for all known routes to the destination, the device will start a route discovery process. Failed route entry will be cleaned from the routing table. The device

will broadcast a Route Request (RREQ) packet, which contains the following data:

- Type: RREQ
- Hop Count: The number of hops from the originator device to the node handling the request.
- RREQ ID: A sequence number uniquely identifying the particular RREQ when taken in conjunction with the originating node's Bluetooth address.
- Destination Bluetooth address: The Bluetooth address of the destination for which a route is desired.
- Destination Sequence Number: The latest sequence number received in the past by the originator for any route towards the destination.
- Originator Bluetooth address: The Bluetooth address of the node which originated the Route Request.
- Originator Sequence Number: The current sequence number to be used in the route entry pointing towards the originator of the route request.
- TTL: the number of hops the RREQ should be kept alive.

The RREQ uses the 48 bits Bluetooth Address instead of the 32 bit IP address. The reason is that the management of IP address is not a distributed process. A large number of Bluetooth devices do not have fixed IP address, and protocol such as DHCP is an overhead and also not available at all before a connection is set up. On the other hand, in adhoc networking context, the IP address format is useless, and the fact that IP is not unique for a device adds further more overhead to networking; e.g. a device in IP networking may have a different IP address after restarting.

The sequence number has a critical role in this protocol. It indicates the "freshness" of a route toward a node. Every node has a sequence number and also for every route in its routing table, there is a sequence number for every destination; therefore it is called destination sequence number.

The original node will increase its own sequence number, and set the destination sequence number to be the largest one in its routing table or unknown if the node is unknown. When broadcasts RREQ, a node will cache originator Bluetooth address and the request ID, which two combined uniquely identifies a RREQ. If the node receives the same RREQ again, it will *discard* the packet. To prevent unnecessary network-wide broadcasting of RREQ, the originating node uses an *expanding ring* search technique. The originating node should set the TTL of the RREQ to the last known number of hops to the node plus an initial search ring distance. If the RREQ failed, the node will increase the search ring distance by setting a larger TTL gradually.

When a node receives a RREQ, it will update its routing table related to the original Bluetooth device of the RREQ and the previous hop because obviously the node needs to be connected with the original node and the previous hop to receive the packet. The node should create a new route if there is not one; otherwise, the receiving node should compare the sequence numbers and use the larger one for the route. The receiving node will keep the RREQ ID and the originator address for a certain time, so that it will not process the same RREQ again. The time should be approximate to the amount that is needed to finish the route discovery process.

Next, the receiving node will check the destination Bluetooth address and if the address is the same as its own address, then the route is found and the node should

increase its sequence number; otherwise, the receiving node look the destination in its routing table. If the sequence number for the destination in the routing table is larger than the one in RREQ, the route is found too because it is fresh. If the route cannot be found in either case, the RREQ will be broadcast again to the next hop nodes if TTL is still larger than zero. If the route is found, the node will send a response, i.e. RREP packet, to the original node. The RREP packet contains the following data:

- Type: RREP
- Distance: The number of hops or weighted distance from the originator Bluetooth address to the destination Bluetooth address.
- Destination Bluetooth Address: The Bluetooth address of the destination for which a route is supplied.
- Destination Sequence Number: The destination sequence number associated to the route.
- Originator Bluetooth Address: The Bluetooth address of the node which originated the RREQ for which the route is supplied.
- TTL: the number of hops the RREP should be kept alive.

If the RREP is generated by the destination node, the sequence number is set with the sequence number of the node. If the RREP is generated by an intermediate node that has a fresh route to the destination node, the sequence number is set with the sequence number of the destination route in the routing table of the intermediate node.

When a node received a RREP packet, again, it will update its routing table for the previous hop. The node will also check its routing table to update the route to the destination node. If the route is a new route, it will be generated, and if it exists, its

sequence number is updated with the sequence number in the packet if the sequence number in the packet is higher. The packet is then forward toward the originator Bluetooth Address with the Distance increased by one hop.

In the end, the originator Bluetooth node receives the packet and adds the route in its routing table and the whole route discovery process is finished. The discovery process is time consuming and should be avoided when possible. For example, all the redundant routes to the same destination should be tried before starting a route discovery process.

Optionally, a node may send renew packets, RREN's, to its direct-linked neighbors periodically to inform its neighbors about any route update. There are several purposes for sending RREN:

- To keep the connections alive with its direct-linked neighbors. The direct connection may get lost if the two connected devices have not communicated for too long. This happens when either device time out the connection to save energy or to use the resource to connect to other devices.
- To detect failed node. If a node cannot send RREN to a neighbor or it has not received RREN from the neighbor, it can consider the neighbor is not reachable to the node. The node can then update its routing table and inform its other neighbors about the update.
- To send any update during the update period to direct-link neighbors. Bluetooth network is very unstable, so changes happen all the time. Using RREN, a node can inform the changes to its neighbors in time.

RREN is optional because under some conditions, battery consumption may have

higher priority and RREN is considered to be a waste of battery. A node can simply use the routes it has and start discovery process if the routes fail because they are obsolete. For example, for a Bluetooth sensor network, RREN may not be used since battery life time is one of primary concerns.

RREN contains the following information:

- Type: RREN
- Number of updated routes: The number of routes that are created or updated in the last period of time.
- Destination Bluetooth address and distance of the routes. If the distance is a negative number that means the destination is not reachable from the node anymore.

There are not originator or destination addresses in RREN because RREN is between directly linked nodes which know the addresses of each other. A node records the following route updates before send them to its neighbors in RREN:

- A new route entry is added to its routing table. For example, when a new node just joined the network, a new route will be created in the routing table.
- An existing route is removed. For example, when a node detect that a neighbor node cannot be reached any more, all the routes via the neighbor node are not valid anymore and will be removed from the routing table.
- An existing route is updated. There are two possible update to an existing route. One is the distance or the number of hops is changed; the other is the destination sequence number is increased which indicates the route gets fresher. Since the

routing table is redundant, the update is recorded only when the shortest distance or the freshest route are updated.

Once the node sends the update message, it will discard the old record and start to record new updates. If a node received a RREN message, it first updates its timer for the sending node and it knows the sending node is still alive. Then the node will go through the update routes: if a route has a negative distance, it is invalid, so the node will remove the route from its routing table; otherwise the node will update its routing table with the new information.

As discussed previously, the routing table keeps redundant routes to compensate the instability of the Bluetooth network. There are two factors to be considered for picking the next node:

- The distance or the number of hops to the destination
- The sequence number or the freshness of the route to the destination

While a fresh route may be longer, but the possibility of failure may be smaller than a less-fresh route. A routing strategy can be decided based on the condition of the Bluetooth network. If a network is very volatile the freshness should be the main factor; otherwise for a stable network, the distance should be the main factor.

4.3 Bluetooth Adhoc Network Implementations

We implemented a program to support Bluetooth Adhoc Network using the algorithm described above. There are two main parts for the implementation: one part is

about device discovery and the routing information update between the directly connected nodes. To implement this part, we need to use the Bluetooth discovery protocol SDP. The other part is about message routing and adhoc route discovery if there is no route. As we mentioned before, the program will route and transfer data packet on L2CAP layer of Bluetooth protocol stack.

Initially we implemented our algorithm on Linux operating system by adding routing routines and a routing table in BlueZ. BlueZ is the official Linux Bluetooth stack, and BlueZ kernel is part of Linux kernel. We modified the BlueZ kernel and change the BlueZ stack into a routing stack. The project is hosted at routingstack.sourceforge.net. We are not satisfied with the implementation because of the following concerns:

- It only supports a device with Linux OS, which may already have access to the Internet or WIFI network
- It needs to modify Linux kernel, so a device must have the kernel with routing stack enabled, which usually is not an easy task.

An ideal solution is a program that can be installed on many types of devices, including small devices with very limited resources such as cell phones, without any modification of the hardware or the operating system of the devices. After research on various Bluetooth stack implementation and interfaces, we found JSR 82 [17], the Bluetooth API for Java Micro Edition, is a very good choice. JSR 82 is *not* an implementation of Bluetooth stack but instead it provides Java interfaces to the underlying Bluetooth stack implemented by the device. JSR 82 provides another abstraction of Bluetooth stack, which means for two JSR 82 enabled devices their Java Bluetooth interfaces are the same even though they have completely different

implementations of Bluetooth stack. If a program can support Bluetooth routing for JSR 82, then it can support many devices that support the JSR 82. Also, the JSR 82 interface is open to Java applications, so there is no need to make any changes to the operating system. We can simply install our program on all the devices that wish to join the adhoc network.

The JSR 82 interfaces allow our program to start a service server which can be discovered by other devices. The program can also start device discovery and service discovery to discover services from other devices. After the devices discovered each other and become connected, L2CAP packets can be transmitted among the devices to transfer data. The program can then route the L2CAP packets as needed. The program is a good example showing routing or adhoc networking can be done at a high level such as the application level. If a few necessary interfaces are provided, then it is possible to do routing and networking by installing an application without changing hardware or operating system. It is much easier to install an application than to change the hardware or operating system. Such an application is also platform and device independent, which is essential to adhoc network with many different types of devices. Also, there will be more tools in the application level to help enhance routing performance; for example, the routing application may use another machine learning application to enhance routing performance. Although our algorithm is language independent, we use Java which makes it easier to support various machine or small devices since Java runs in its own virtual machine on the devices and thus become platform independent. We can call the network supported by our program Java Bluetooth Adhoc Network (JBAN). The following is the layered structure for a device with the JBAN System:

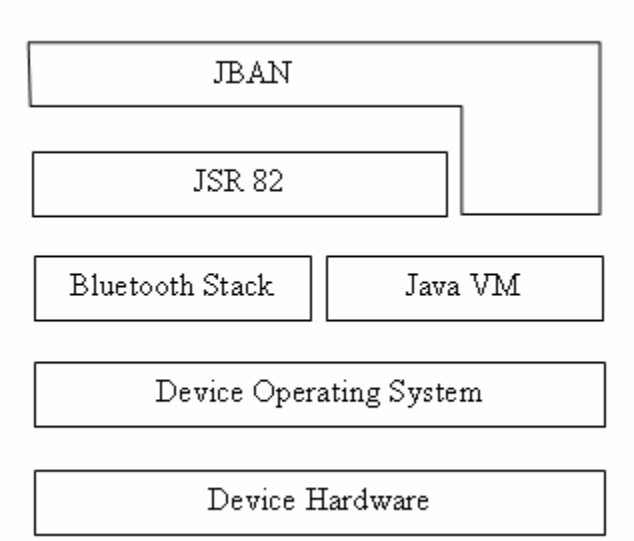

Figure 4 - 2: Structure of a device with JBAN

In some devices, the Bluetooth Stack is implemented in Java VM layer but it can be implemented separately too. JBAN is built on top of JSR 82 and Java VM which are platform and device independent. This contrasts to our routing stack for BlueZ, which has to be implemented in the kernel of the operating system. For most of the current existing Bluetooth networking algorithms we studied previously, new hardware functionalities or properties are needed to make them work and in a very limited way.

Chapter 5: JBAN System Design and Evaluation

5.1 JBAN System Design

The JBAN system has the following structure:

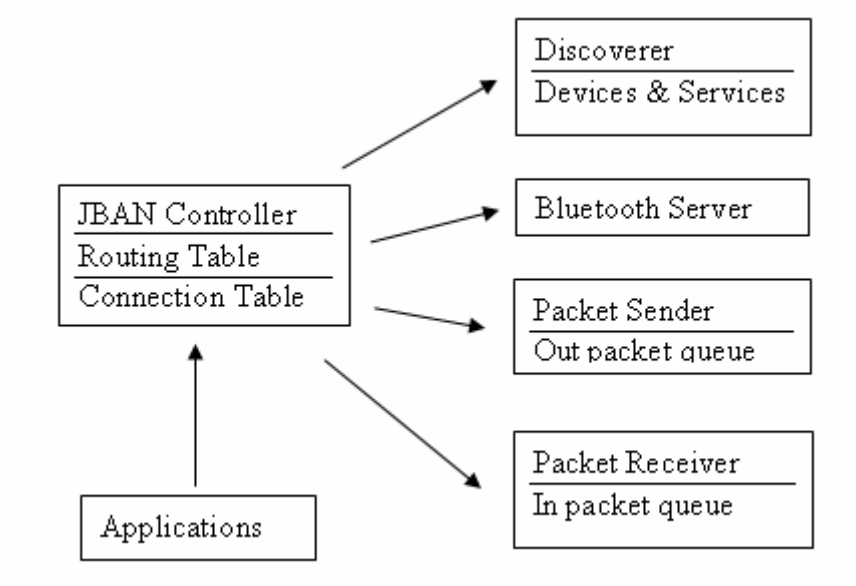

Figure 5 - 1: JBAN System

The main components of the system include a device and JBAN service discoverer, a Bluetooth server, packet sender and receiver and a routing table and routing controller.

Discoverer uses a discovery agent provided by JSR 82 to discover other devices which are also running the JBAN system. There are two steps in discovery, device discovery and service discovery. First the Discoverer searches for devices in its scope and for each device it found the Discoverer searches for JBAN Bluetooth service which is hosted by a JBAN Bluetooth Server. Once a service is found, the system will be able to

open a connection to the device. The JBAN system will try to find and connect to as many as possible devices that are running JBAN system as well. The JBAN system will not do low level, such as physical level, network optimization. The node hosting the Discoverer may be a master node or a slave node or both, which is decided by the manufacture of the Bluetooth chip, but it works in the same way in JBAN. The sample code is as the following:

- To get the Bluetooth discovery agent

DiscoveryAgent agent = LocalDevice.getLocalDevice().getDiscoveryAgent();

- To discover other Bluetooth devices:

agent.startInquiry(DiscoveryAgent.GIAC, this);

- To discover JBAN services on other Bluetooth devices:

agent.searchServices(attrSet, uuidSet, device, this);

Bluetooth Server hosts JBAN service with a unique ID. The server has to be started before the device joins the JBAN network. After started, the server blocks and waits for incoming connection:

String url = "btl2cap://localhost:" + JBAN_UUID.toString() + ";name=JBAN";

L2CAPConnectionNotifier notifier = Connector.open(url);

L2CAPConnection conn = notifier.acceptAndOpen();

The string "*btl2cap://*" indicates the server is using L2CAP protocol. JBAN_UUID is the unique ID for JBAN service. All the devices that want to join in the adhoc network need to run the server with the same UUID. An alternative is to use RFCOMM protocol for data transfer and the URL will be the same except "btl2cap://"

should be replaced with “btspp://”. In L2CAP protocol, data is transferred with packets while in RFCOMM protocol data is transferred as stream. The advantage of using RFCOMM streaming is that data packaging or reassembling on both ends of the connection can be avoided, and once a path is formed data can be streamed through the pipe. In the first version of JBAN we implemented it using RFCOMM protocol. However, there several reasons for us to choose L2CAP in the end:

- Using RFCOMM protocol, every device on the path needs to read data from an input stream into a buffer and write the data in the buffer to an output stream. This is not as efficient as simply forwarding an incoming L2CAP data packet. The efficiency of using L2CAP protocol also comes from the best-effort nature of packet routing.
- It is more robust to use L2CAP protocol since packet can get to the destination through different route. This is especially important when redundant routes are saved in every routing table.
- L2CAP is bellow RFCOMM and some other protocols, so if we can do routing using L2CAP then all the protocols on top of L2CAP will be supported.

Packet sender sends data packet through a list of connections to other directly connected nodes. All the connections are kept open for better performance since it is time consuming to recreate a connection. Also, since there are not more than 7 connections because of the limitation of Bluetooth topology, there is not much overhead to keep all the connections open. Each connection has a queue of outgoing data packets designated to the connection by the controller. Packet sender runs in a dedicated thread and checks all the queues and send the packets in the queues in a round-robin way. Packet receiver

checks the same list of connections and received incoming data packets if any. Packet receiver runs in a dedicated thread too and it passes the data packet to the controller.

JBAN Controller is the core of this program. It starts the JBAN system by starting its components such as discoverer, JBAN server, packet sender thread and packet receiver thread. It also maintains the routing table and route data packet based on the routing table and a routing algorithm. Another major functionality it has is to decompose and package data for sending and reassemble data packet upon receiving them. The JBAN Controller also communicates with applications that are using JBAN. It contains a list of applications that want to receive data via JBAN networks.

The working flow of the JBAN system is as the following:

1) A device needs to install JBAN through a data cable or a web site it can access the Internet. Over the air installation from a web site is supported by Java ME platform. The only thing the user needs to do is to type in a URL or click a hyperlink of the URL from within a message.

2) After installation, the user can start the JBAN application. The discoverer will start to discover other devices in the range and search for JBAN services on these devices. The discoverer will try a certain number of times (2 by default) and after that the controller opens a connection to each service that has been discovered.

3) The controller then starts the receiver and the sender thread and sends a REG packet to register this device to the other devices. After receiving the REG packet, the other devices will update their routing table and also send a selection of routing information to this device, which then update its own routing table based on the information.

4) Applications that want to send packets to other devices can use the interface provided by the controller. Applications that want to receive packets from other devices should register as listeners to the controller, and the controller will inform the applications when data packets are received.

5) When an application send a message via the interface provided by the controller, the controller will check the routing table to find a route for the message. If it cannot find such a route, it will send a RREQ packet to other nodes to discover the route.

6) When receiver received a packet it checks the packet ID and the target address. If it is a route discovery packet including RREQ, RREP, RERR or RREN, then it process the packet data based on the route discovery algorithm described in 3.2. If it is a data packet and the target is this device then the controller will call corresponding application listener. In this case, the packet ID represents a port number for the application. If the target is not this device then the controller will route packet in the same way as described in 5).

5.2 Evaluation

5.2.1 On Device Evaluation

Contrast to other Bluetooth network solutions, JBAN has been tested not only on emulators but also on real cell phones. The devices we used are up to six Nokia 6600 mobile phones. The JBAN program is installed on every device. Two types of experiments have been done on these devices:

1) Test if JBAN network can be formed efficiently. There is not much delay other than the delay caused by the setup of Bluetooth connection itself. Also since we are only using 6 devices for the testing, there are not many routing data to be initialized.

2) Test if the first goal is achieved, which is to have good Bluetooth network performance under multiple hops. The testing program allows the tester to set a filter for routing so that only desired route is chosen. By using the filter we can intentionally to design a long route to test the network performance. According to our testing, there is only less than one second delay for the data packets with 4 hops. Although the packets have been transmitted through a few piconets, the efficiency is not affected since the routes have been listed in the routing tables and the connections are ready in all the nodes.

3) Test if the second goal is achieved, which is to extend the scope of the Bluetooth network. Like most other mobile phones, Nokia 6600 has class 2 Bluetooth support, so its connection scope is about 10 meters. We verified that with JBAN installed, one device can efficiently communicate with other devices that are 20 or 30 meters away from it.

We are satisfied with the test result of the JBAN system and we are convinced JBAN can be used directly to support a Bluetooth network of many devices and extend the connection distance of the network. Ideally, we should use more devices to test JBAN, but not only it is hard to get many devices but also it is hard to collect testing data on many real devices; therefore, we use device emulators to do more testing on JBAN system.

5.2.2 Evaluation with Device Emulator

Device emulator is a program running on desktop computer to simulate a mobile device. Many instances of the emulator can be run simultaneously to simulate multiple devices. The emulator provide the same interfaces as the device so that if a program can run on an emulator then it can run on the real device with little or no modifications. The emulator we used is provided by Sun Java Wireless Toolkit for CLDC, which is a standard Java ME emulator and has support for simulating Bluetooth networking. The program is available in the following page:

http://java.sun.com/products/sjwtoolkit/download.html

The program being evaluated is the current version of JBAN program, which is available at the following JBAN open source project site:

https://jban.dev.java.net/

Source code and binary build can be retrieved from CVS. We performed the following experiments using the emulator to test various functionalities of JBAN network.

5.2.3 Experiment on Registration

In this experiment, we run 30 instances of the emulator to test the time needed for the initial registration process for a network with 30 mobile devices. Every emulator instance runs the JBAN program and forms the Bluetooth network with other instances.

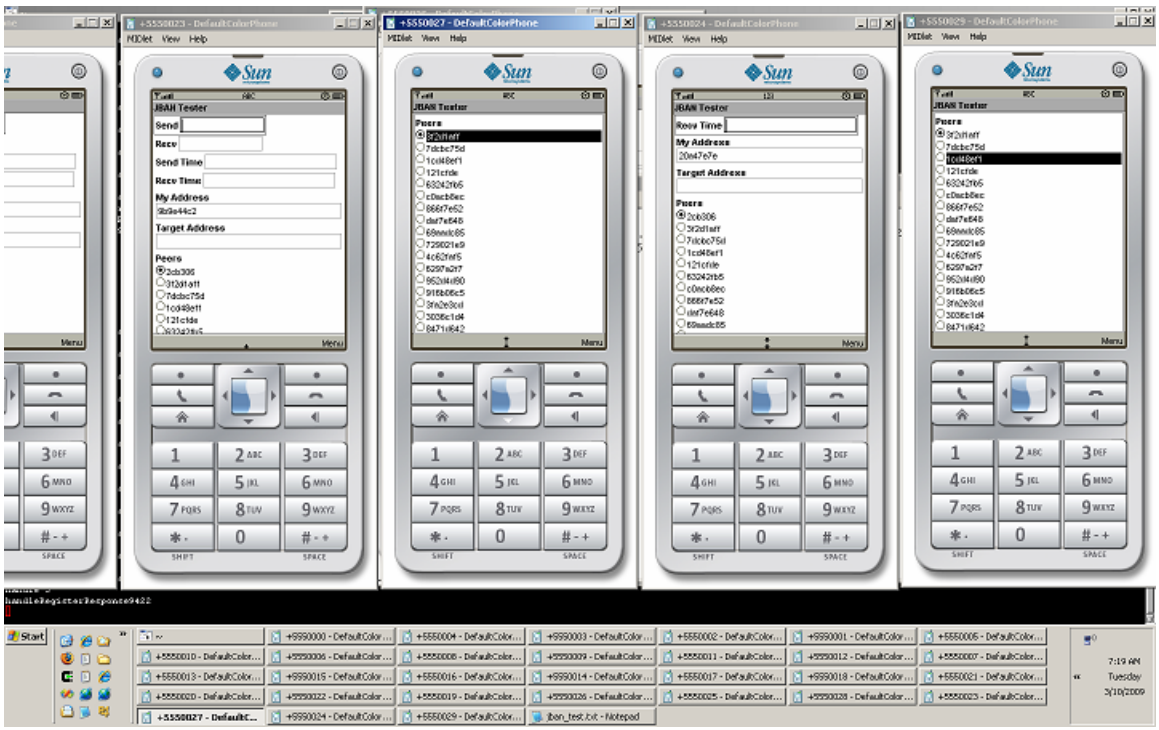

Figure 5 - 2: Running 30 emulator instances to simulate

The experiment is performed with the following steps:

- Start the first device
- Start all the other devices one by one and for each device record the time spend from when the device send a REG message to register to all its neighbor nodes to when the device has received the REGREP message from all its neighbor. After the registration process, the new device will have the most knowledge it can get for the whole Bluetooth network.

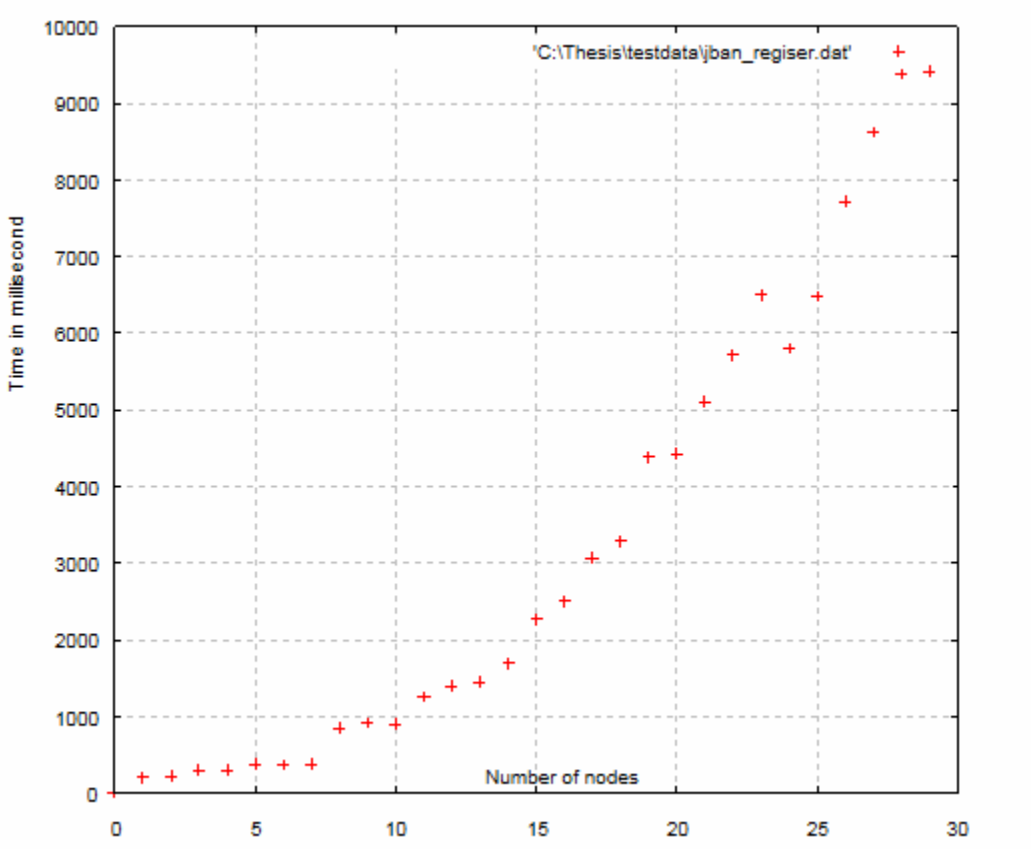

Figure 5 - 3: Time consumed for joining or registry to JBAN network

The result shows the registration is time consuming and the time also increases fast, but since this only happens once when a device joins in the network, it is not a big concern. It takes the last node 10 seconds to finish the registration process but the number will be bigger for real devices. One design optimization is to run the registration asynchronously at the background and allow the new node to start service discovery and communicate with existing nodes once it partially done with the registration, so that it does not need to wait for a long time. Another design option is to let the new node only retrieve local information from its neighbor nodes instead of information about the whole network.

5.2.4 Experiment on messaging without route discovery

In this experiment, we test network speed for nodes that have already been connected which means there is no route discovery needed. This is the scenario if after registration or route discovery, route information is available in the routing table in each node along the route. To do this experiment, we send message over different number of nodes in a network and record the time between sending the message and receiving the message on the other end. We design the network so that every node on the path already knows the next route.

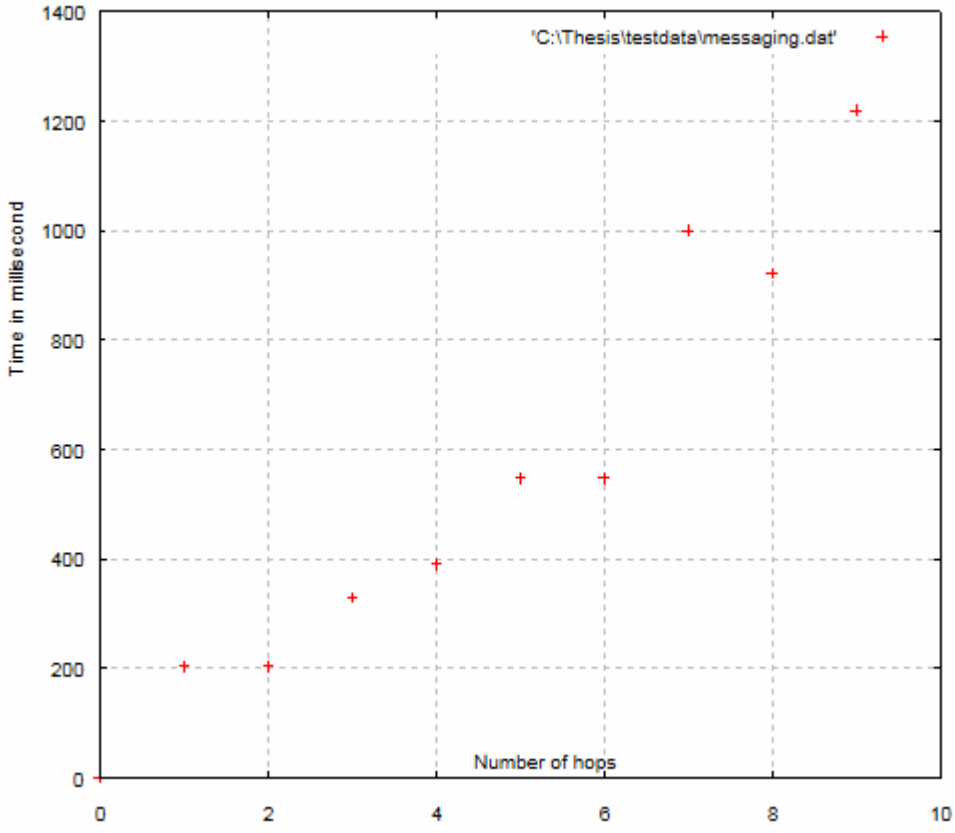

Figure 5 - 4: Time consumed when sending a message across different number of hops

The result shows that transporting message without the need of route discovery is

very efficient. It only takes about 1.2 seconds for transporting the message across 9 hops, and this can show that JBAN is still efficient when the diameter of the network reached 9 hops. Messaging without discovery is common in JBAN network because in the initial registration process, every node obtains much routing information about the whole network, which will help to avoid route discovery.

5.2.5 Experiment on messaging with AODV route discovery

In this experiment, we test the efficiency of route discovery in JBAN network. The experiment is performed with the following steps:

- Start JBAN on the first node
- For every device of the following 9 nodes, it only registers to its previous node. In this way, the n - 1 node does not know a route to the n + 1, 2 …k nodes. So when we send message from n -1 node to n + 1, 2 …k nodes, route discovery has to be performed and the discovery diameter will be 1, 2 … k. The message will then be sent after the route is discovered. We record the time from starting to send the message, discover the route all the way to finally the node on the other end receives the message, so the time include route discovery time and message transporting time.

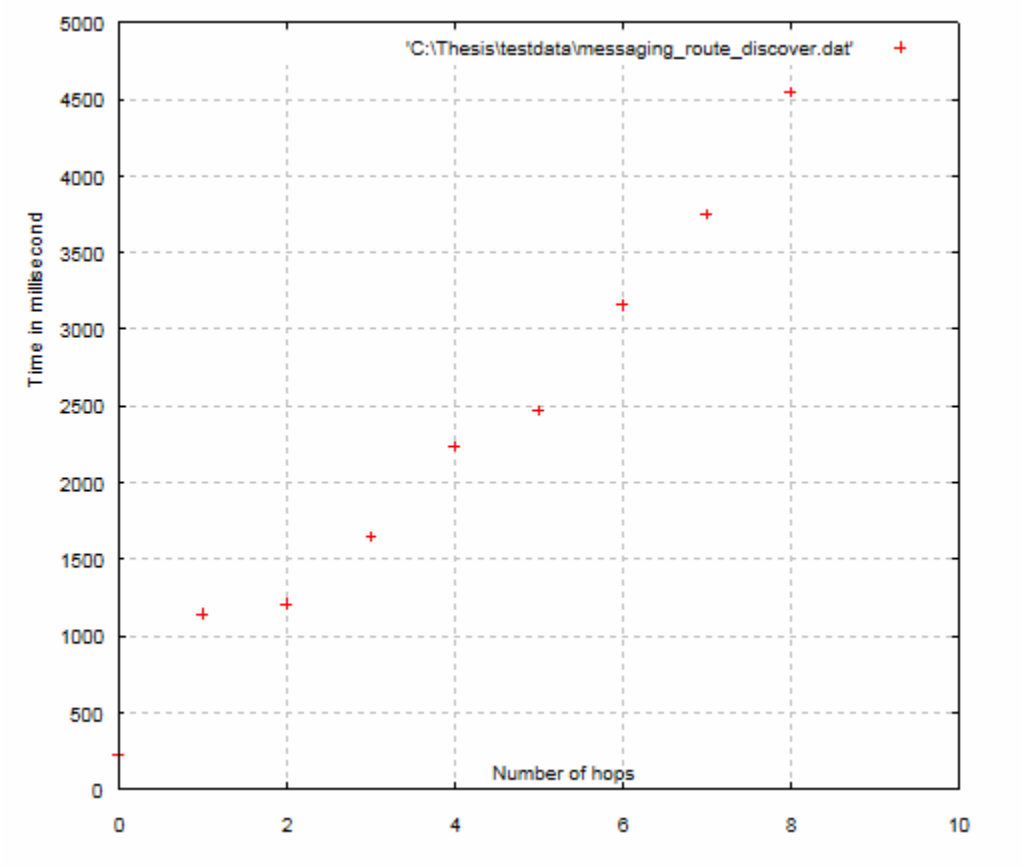

Figure 5 - 5: Time consumed when sending a message with route discovery

The result shows route discovery is time consuming; especially when the discovery diameter is large. However, the advantage of AODV route discovery is that once a route discovery is finished, not only the routing information on the originator and the destination nodes is updated, but also the routing information for all other nodes along the route discovery path is updated as well. We can see that result clearly.

Chapter 6: Conclusions and Future Work

In this thesis, we analyze the special nature of Bluetooth adhoc networking including its topology and instability. After studying related existing technologies and academic researches, we provide our solution for efficient Bluetooth adhoc networking. The solution features a pre-warm registration process, adhoc route discovery and redundant routes. These designs allow many mobile wireless devices to quickly form a robust and efficient network. One contribution of the thesis is a solid implementation, JBAN system. JBAN system is an open source program and has been tested on device emulators and real devices.

Another important feature of JBAN that distinguish it from other networking solutions is that JBAN is implemented in the high level. It does not require hardware or operating system support, and the routing logic is implemented very close to the application level. This allows very different devices to communicate with each other and form an adhoc network. This strategy is not only sound for Bluetooth but also for WIFI or any other protocols. We believe JBAN is a good solution for WIFI network too and it will be fairly easy to modify JBAN to support WIFI or other protocols. Currently JBAN only support packet routing and to make it more useful, an end to end protocol such as TCP should be implemented on top of it. So our future plan includes the following:

- Implement JBAN system on other protocols such as WIFI
- Design an end to end protocol on top of JBAN
- Perform more testing on real devices or emulators using JBAN program to further study the network and improve its performance.

Bibliography

[1] Ching Law, Amar K. Mehta and Kai-Yeung Siu, “Performance of a New Bluetooth Scatternet Formation Protocol”, *In Proceedings of ACM Symposium on Mobile Ad Hoc Networking and Computing,* 2001.

[2] Gy. Miklos, A. Racz, Z. Turanyi, A. Valko and P. Johansson, “Performance aspects of Bluetooth scatternet formation”, *In Proceedings of the FirstAnnual Workshop on Mobile Ad Hoc Networking and Computing, 2000.*

[3] Stefan Zurbes. Considerations on link and system throughput of Bluetooth networks. In Proceedings of the 11th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, volume 2, pages 1315–1319, 2000.

[4] Bluetooth Core Specification Version 2.1, 2007

[5] Satyajit Chakrabarti, “Bluetooth Scatternet Formation and Internetworking with 802.11 and GPRS”, 2002

[6] Jang-Ping Sheu, Jang-Ping Sheu, Kuei-Ping Shih, Shin-Chih Tu and Chao-Hsun Cheng, “A Traffic-Aware Scheduling for Bluetooth Scatternets “, 2006

[7] Hiranmayi Sreenivas and Hesham Ali, “An Evolutionary Bluetooth Scatternet Formation Protocol”, *In Proceedings of the 37th Hawaii International Conference on System Sciences, 2004*

[8] Alok Aggarwal, Manika Kapoor, Lakshmi Ramachandran and Abhinanda Sarkar, “Clustering Algorithms for Wireless Ad Hoc Networks”, *In Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, 2000.*

[9] Theodoros Salonidis, Pravin Bhagwat, Leandros Tassiulas and Richard LaMaire, "Distributed Topology Construction of Bluetooth Personal Area Networks", *In Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communication Societies*, 2001.

[10] G. Tan, Al. Miu, J. Guttag et al, "Forming Scatternets from Bluetooth Personal Area Networks", MIT Technical Report, Oct 2001.

[11] H. Zhang, Jennifer C. Hou and Lui Sha, "Bluetooth Loop Scatternet Formation Algorithm", *IEEE International Conference on Communications*, May 2003.

[12] S. Basagni, C. Petrioli and I. Chlamtac, "Configuring BlueStars: Multihop scatternet formation for Bluetooth networks", *IEEE Transactions on Computers, special issue on Wireless Internet, 2002. in press.*

[13] I. Stojmenovic, "Dominating set based Bluetooth scatternet formation with localized maintenance", *In Proc. Of Workshop on Advances in Parallel and Distributed Computational Models at IEEE PDPS*, 2002.

[14] G. Zaruba, S. Basagni and I. Chlamtac. "BlueTrees – Scatternet formation to enable Bluetooth-based personal area networks", *In Proceedings of the IEEE ICC 2001, Helsinki, Finland*, June 2001.

[15] Bin Zhen, Jonghun Park and Yongsuk Kim, "Scatternet Formation of Bluetooth Ad Hoc Networks", *In Proceedings of the 36th Hawaii International Conference on System Sciences*, 2003.

[16] C. Perkins, E. Belding-Royer, S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing", *RFC 3561, Network Working Group, July 2003.*

[17] Java Community Process, “Java™ APIs for Bluetooth”, *Java Specification Requests, JSR 82, http://jcp.org/en/jsr/detail?id=82.*