DESIGN AND ANALYSIS OF A MOBILE FILE SHARING SYSTEM FOR OPPORTUNISTIC NETWORKS

by

Gautam Chavan

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

UNIVERSITY OF TEXAS AT ARLINGTON

August 2009

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my family and friends for their support and encouragement that made all this possible.

I would like to offer special thanks to my supervising professor Dr. Yonghe Liu for his continuous help, support, and guidance throughout this endeavor. His patience, experience, and knowledge have been invaluable throughout my research, and I am truly grateful for this. I would also like to thank Dr. Mohan Kumar and Dr. Bob Weems for their encouragement during the course of the project.

I would also like to thank Mr. Mike O’Dell and Dr. Bahram Khalili, my graduate advisors and professors, for their help, advice and guidance throughout my M.S.

July 11, 2009

ABSTRACT

DESIGN AND ANALYSIS OF A MOBILE FILE SHARING SYSTEM FOR OPPORTUNISTIC NETWORKS

Gautam Chavan, M.S.

The University of Texas at Arlington, 2009

Supervising Professors: Dr. Yonghe Liu

In the past several years, wireless mobile devices with advanced computing, sensing, and storing capabilities have been increasingly developed by handset manufacturers, deployed by wireless carriers, and accepted by consumers. A prominent example is the vast success of Apple's iPhone, which witnessed a sale of 1 million units during the first 3 days of sales for its new version iPhone 3GS. As a result of the increasing popularity of these mobile devices, users can carry, utilize, and exchange information embedded therein, ideally wherever they go and whenever they want. A prominent scenario is where information is exchanged among users when they are opportunistically in contact with each other while on the move.

In this thesis, we propose a file sharing system for opportunistic networks, where users equipped with smart mobile devices exchange desired files or data during opportunistic contacts. Our system is based on Bluetooth, the de facto and universal standard for short range communication among mobile devices. Facing the unique challenges of opportunistic networks, namely intermittent connections and fast changing link capacity, we employ the following techniques.

We introduce the concept of priority, where a user of a system can set the priority of the files for downloading. For example a user may mark the audio files as high priority, large video files as low priority, and wallpapers or images as medium priority. These files will then be downloaded according to their priorities.

We introduce push mechanism where a user of a system can assign a few files for distribution. On finding peers, the system forwards the files to these peers. For example a researcher can set aside a few of his publications for sharing with other researchers, and on finding of a researcher, the system forwards the publications to the remote users.

We also introduce file compression mechanism where the mobile device compresses the sharing files in order to reduce bandwidth consumption and downloading time. As a result, energy can be conserved and the lifetime of the mobile device can be extended.

We implement the proposed file sharing system on a HP iPAQ smartphone, which is based on Windows Mobile. The connections between devices are established on the fly and file transfer takes place within a short span of time. If an interested file is present on more than one peer devices, simultaneous downloading is implemented to increase the overall data downloading rate.

We have performed extensive experimental study to investigate the system performance in terms downloading speed, energy efficiency, and effectiveness of the techniques mentioned above.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

The speedy proliferation of mobile users over the past few years has given rise to lot of companies investing in mobile devices and technologies that are used by these mobile devices. Wi-Fi and Bluetooth remain two critical technologies as they enable users in the same vicinity to communicate with each other. Most mobile users store certain information on their mobile devices and in some scenarios would like to share this information with other users in the physical proximity.

Information exchange is an important part of our day to day lives and we exchange information in many different ways, be it exchanging information via emails, online chat, by talking over the phone and in other ways. Also, an important part of information exchange is to know where to seek the information from, i.e. the source for the information.

With the coming of App Store [27], Android Market [28] and Ovi Store [29], users can easily download applications from the application stores to their mobile devices. However consider a scenario where the users are able to download applications not just from the online application store but from other users too.

This approach will help both the online application stores as well as the users of these applications. This will help the online application stores as the apps will be much more widely distributed, as the mobile users share their application and also inform other users about applications that can be of interest to them. This approach also helps the millions of users, as most online application store have thousands of applications and it is extremely difficult for one person to go through all the applications and select a few of them, instead it would be nice if the users around him suggest or inform him about applications that might be of interest to the user. If the user seems interested in the application, he may then download the application from the online store or from users in the physical proximity.

There are a few applications in the area of file sharing over Bluetooth such as BlueTella [6], BlueMatch [7], and BlueTorrent [7]. The predominant problem with this approach is that they do not capitalize on the physical proximity of the users.

BlueTella and BlueMatch support file sharing between two mobile devices at any point of time. They also require the user to search for a particular file in the neighboring nodes, i.e. to select a node from one of the nodes in the vicinity and to send a search query to the node and wait for the response. Once the mobile device gets a response the user may decide whether or not to initiate downloading of the file that is present on the remote Bluetooth device. If there is an interruption during the file transfer the entire file transfer is nullified and the partial file is removed from the device. Thus the remote device must stay in the vicinity until the file is completely downloaded. There is no push mechanism where in the user is able to distribute content to other mobile users in the communication range.

BlueTorrent however is a peer-to-peer file sharing system and hence the files are split into pieces. The individual pieces are downloaded from different peers in the vicinity of the mobile device. The system employs a pull only mechanism where a user can only search for a file that he or she is interested in. If the user of a mobile device wishes to distribute his content, he is not allowed to do so. When a query results in a successful match then the user is given the responsibility to decide whether to initiate the file download. BlueTorrent was simulated using UCBT NS-2 simulator and hence performance evaluation is not a realistic. Also, since desktops and laptops were used which have continuous power supply, the impact on the battery life for mobile devices that are battery enabled is impossible to deduce. The number of threads per process is an important factor for the performance of the application on mobile devices and has been ignored. To fully understand the performance evaluation of system that supports peer-to-peer file sharing, implementation on an actual mobile device is extremely important. Also important is testing these mobile devices in real time, i.e. in a realistic environment. To fully understand the impact on the mobile device by other mobile devices in the vicinity, by interference of Wi-Fi signals and signals emitted by other mobile devices. Also, the difference in infrastructure also matters to an extent.

Our objective is to come up with an architecture that matches the requirements of opportunistic networks. The user intervention should be minimal or not present at all. The user would specify the list of files that he is interested in and the application takes care of finding the files from other Bluetooth enabled devices in the neighborhood and also downloading them from the neighboring mobile nodes. The user can also prioritize the files that he wants download so that we can download high priority files first and then the low priority files.

The application must be able to simultaneously download a given file from multiple sources. Hence it needs to have a peer-to-peer support. It should actively search for Bluetooth enabled mobile devices around and also filter out those devices that aren't running the application. The application should be able efficiently search for the files in the remote devices.

Thread handling is an important part of the application as there are server and client threads. The server threads are serving the clients of the remote devices and the client threads are looking up the devices for files. A Bluetooth enabled mobile device can communicate to a maximum of seven devices simultaneously.

Whenever possible, the files must be compressed before transferring to the remote device as this will minimize the time taken to transfer the file which in turn will minimize the bandwidth utilization and also minimize the amount of battery being exhausted. For e.g. a 10 mega byte file on compressing using the Pocket RAR which is a pocket pc freeware reduces to a file of size 59 kilo bytes with compression rate of 99.5%. The time taken to compress and de-compress a file of size 10 mega bytes is in the range of 15 – 20 seconds.

The application should be robust against crashes, mobile device switching off, or remote Bluetooth device going out of range, etc. The application should not be heavy as the main this might reduce battery life of the mobile device. Also, connection requests from or to remote mobile devices shouldn’t be allowed when the amount of battery left on the mobile device goes below a threshold.

Energy efficiency is another critical factor as the application will be running on a mobile device that has a battery constraint. The application should not drain out the battery from the user’s mobile device. The application should be light and must run in the background and not affect the other applications that are running. Also, if a there is a Bluetooth enabled device in the vicinity and the device is not running the application, the user’s mobile device should not try to connect to it, as the connection is bound to fail as the remote device does not run the service.

We were able to implement the software system on a HP iPAQ Business Manager and verify the working of a peer-to-peer file sharing system. The file downloading works just as mentioned and if a file is shared at more than one source then the file is downloaded from all the sources where the file is available.

We also verify the push mechanism where in if a file is supposed to be distributed to other users around, the file is pushed to the neighboring Bluetooth enabled devices running the application.

We also compress files using Pocket RAR and look at the pros and cons of compressing files before sharing them. We also discuss the compression behavior for different type of files.

For instance if the doctors and staff want to share some critical information about a patient the doctor need not be in front of the staff to do the same, if the doctor is doing his rounds and comes within the range of the staff he may download that information from the staff and then decide the necessary action to take. At work the employees can download documents for the products that they are currently working on without coming in contact but just being the same floor or meeting room. The cops can use this to share information about a convict who has escaped from prison or a convict they are looking for.

Information exchange is important even in the education system, think of a college where a professor need not upload home works or assignments to his website, he may instead keep a copy of the home work or assignment in his mobile device and later distribute it to the students during the class.

Information exchange is critical in the media and entertainment industry too. The producers of a movie can send a trailer to a few Bluetooth enabled devices which can then distribute this trailer to the mobile users. The mobile users do not have to go to movie theaters to watch upcoming movie trailers; neither do they need to visit iTunes [27], MSN [30] or Yahoo [31] to watch the trailers.

CHAPTER 2

BACKGROUND

In the following sections we will discuss Opportunistic Networks 2.1. , Bluetooth 2.2. , Peer-to-Peer File sharing 2.3. and understand the workings of the same.

2.1. Opportunistic Networks

Consider a scenario where a researcher goes on a 2 – 3 day long technology conference, researchers from all around the world come together and share their research. It is practically impossible for a researcher to find out about all the other researchers and choose the researchers that he or she would like to meet up with, as their areas of research overlap. This is where opportunistic network comes into the picture. Every researcher can carry an opportunistic network node along, which contains his or her areas of research and probably a set of papers published by the researcher. At the conference, the node can start the node discovery procedure and find the researchers whose areas of research overlap with that of itself. The node can make use of Bluetooth or Wi-Fi or any other short-range wireless communication technology to get in touch with the researcher and probably schedule a meeting with him. Also, the researcher may be able to share a few of his publications for other researchers in the same area to read.

An Opportunistic Network is a network of nodes that are connected wirelessly. Opportunistic Networks support spontaneous interaction between mobile users carrying wireless mobile devices. Opportunistic network nodes leverage the fact that they can communicate with one another whenever they come within each other's range. The nodes are either mobile or fixed. The distance between such nodes is very small in the range of 100 – 300 meters. The nodes provide the following functionality at the minimum:

a) Node discovery

   A node is able to discover other nodes within a given range.

b) One-hop message exchange

   A node is able to send or receive data from other nodes in the vicinity.

Opportunistic Networks grow opportunistically. They leverage the wealth of pervasive resources and capabilities that are in reach. They have a high interoperability and integrate different communication, computation, sensing, and storage systems within reach.

An Opportunistic Network Node is a node with capabilities of short-range communication. The node runs an Opportunistic Network application, which helps to find other nodes and to exchange data between them.

A Mobile Node in an Opportunistic Network is a mobile device carried along by a user that acts as an Opportunistic Network node.

An Information Sprinkler is a fixed Opportunistic Network node that is fixed at a particular location, acting as any other Opportunistic Network node. The Information Sprinkler can act in two modes, either in the Sprinkler only mode where it only distributes data or information, in a Sink mode where it only collects data or information or in both modes at the same time.

An Opportunistic Network node contains a data structure that describes the nodes profile. The profile is used to inform other users about the individual carrying the node. The profile consists of two lists, one list mentioning the data or information that user has and willing to share and another list mentioning the information the user is interested in. Before going ahead let us look at an example to fully understand the importance of opportunistic networks.

E.g.: A user goes to shopping mall along with his spouse or with a set of friends. Once they are done with shopping suppose they feel like watching a movie and they are interested in knowing the movies that are currently playing in the theater and some more information about the movie like say, they want to watch a trailer or read a review. If there are Information Sprinklers around the mall and they are continuously broadcasting the list of movies that they are currently playing, their trailers, reviews and ratings, the user can receive that information on his opportunistic network node and then decide whether he they are willing to go for any of the movies or not.

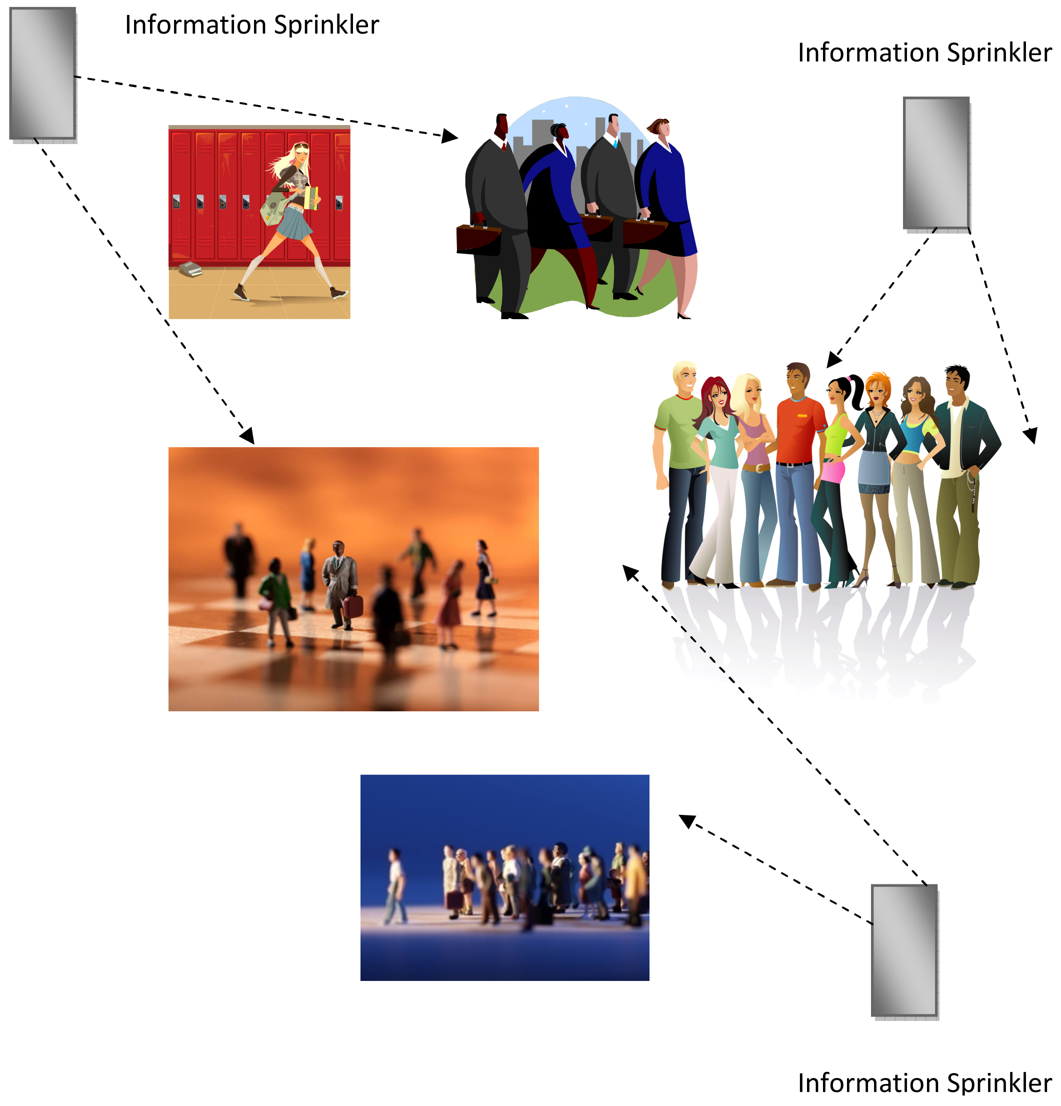

Figure 1: Information Sprinklers in a shopping mall

2.1.1. Ideas and Concepts

A few ideas and concepts that are predominant in opportunistic networks are explained below:

2.1.1.1. User vicinity exploitation

A necessity for short range communication devices is to locate wireless devices and the users of those devices when they are in communication range of the wireless device. This increases the opportunity for the users to meet face to face.

2.1.1.2. Profile based user interest expression

Once the devices have discovered each other, there needs to be some logic to calculate if further communication between the devices will benefit either of the users. This is achieved by keeping track of a users profile which contains users likes and dislikes, etc.

2.1.1.3. Data dissemination

When device A's knowledge of the data stored on the device matches with the interest of user B, this knowledge must be transferred from device A to B.

2.1.1.4. Open and unrelated user group

Opportunistic Networks do not make assumptions about the participating users in a network. Hence, the users most likely pursue their personal interests.

2.1.1.5. Unpredictable communication pattern

Communication and information exchange happens between devices due to the sole reason that they were at the same place at the same time. A user should not rely on Opportunistic Networks to satisfy all his interests.

2.1.2. Opportunistic Network Applications

There is wide range of applications for opportunistic networks. Opportunistic networks find applications in the industry, where users want to share information about the things on which they work together. It has wide range of applications in the education system, media and entertainment industry. Also, such networks have are of interest in particular in social networking. Hence, they are broadly classified into Active and Passive Collaboration. Below we briefly discuss about the classifications and also provide existing applications for the same.

2.1.2.1. Active Collaboration

Active collaboration [2] exploits the physical proximity of the users. In addition to exchanging information with other users, this may inform the user of the presence of other users in the network, which may in turn lead to a face to face talk to accomplish some common goal. Active collaboration does not require the user to store on the device a detailed knowledge; instead a short description would suffice. Once the communication is established the users can exchange the detailed information by other means.

Lovegety [10] is a small mobile device to introduce people to each other that happen to be within the given communication range.

SpotMe [11] is collaboration system and tool that is used for conferences, symposia, corporate meetings, etc. Using a special purpose hand held wireless mobile device, users are able to search for interesting conversations nearby. A SpotMe device is personalized upon registration, like for instance taking a photo and uploading it onto the device. The information is stored on the device itself. On the spot, the users can specify special interest in another user. The device will then give a notification when the other user is in communication range.

Nokia Sensor [12] is a Bluetooth based application for Nokia cellular phones. A user specifies a profile on the device, for example taste in music, and allows other Nokia Sensor users to search for their devices. This allows users to form spontaneous social networks or communities. In addition it allows users to share files.

2.1.2.2. Passive Collaboration

Passive collaboration [2] accumulated and forwards information from and to the users within communication range. This happens in the background without the user being interrupted. This leads to autonomous information dissemination. Incentive schemes might be crucial to motivate users to share information.

Data Dissemination in iClouds [2] is an Opportunistic Network reference architecture and prototype developed at the Telecooperation Group, Computer Science Department, at Technical University of Darmstadt. iClouds [2] support both active and passive collaboration applications. Let us take an in depth look at the profile based data dissemination mechanisms proposed in iClouds. iClouds contain the so-called information lists, iHave-list and the iWish-list. The iHave-list contains the information that the node wants to contribute to other nodes in the network. The iWish-list contains the information that the user is interested in.

2.1.3. Privacy Issues

The mobile devices in Opportunistic Networks are carried by us, humans. The communication occurs in the users proximity, the fact that information passes from his device or to his device might conflict with the user’s privacy. Privacy is the ability of a user to prevent information about him or her from becoming known to other users. In the context of Opportunistic Network this means that the information needed by the user and the information shared by the user should not link it back to the user.

The user’s identifiability is grouped into three categories:

a) Identity

A user communicates with other users in the networks and reveals information that can be used to clearly pin point the user. Examples are full names, Facebook profiles, etc.

b) Pseudonymity

This category is the ability to prove a consistent identity without revealing a users real identity, using a pseudonym instead. This is extremely common on the internet, for example in chat rooms, electronic mails, etc.

c) Anonymity

Anonymity is ability to remain unidentifiable within a set. This category is employed by the users who do not wish to share their identity with other users in the network.

2.2. Bluetooth

Announced by the Bluetooth Special Interest Group [1] in the early 1998, Bluetooth is a low power, low price, and short-range radio technology for wireless communication between mobile devices. Bluetooth operates in the unlicensed

ISM (Industrial Scientific Medical) band at 2.4GHz. Bluetooth is designed to operate as a background process on a device. Each Bluetooth device is assigned a unique 12-byte address, and a connection between two devices requires the knowledge of the target address. Bluetooth also supports authentication of devices and encryption of data transferred between Bluetooth devices.

2.2.1. Master-Slave Relationship

The communication between Bluetooth nodes is based on Master-Slave relationship. A Bluetooth device that has a service to provide, acts as a slave and the device that uses the service is the Master. The Master node can be connected to a maximum of seven Slave nodes, forming a star-shaped cluster, called a Piconet. The Slaves nodes can communicate only with the Master and not with other Slaves. The Master node, however can communicate with all the Slaves. A Bluetooth device may participate concurrently in two or more piconets. It does this on a time-division multiplexing basis.

A Bluetooth device can never be a Master of more than one piconet. However, a Bluetooth device may be a slave in many independent piconets. A Bluetooth device, that is a part of more than one piconet is said to be a part of a scatternet. For two Piconets to be connected, a Slave node in one Piconet will act as a Master in another Piconet.

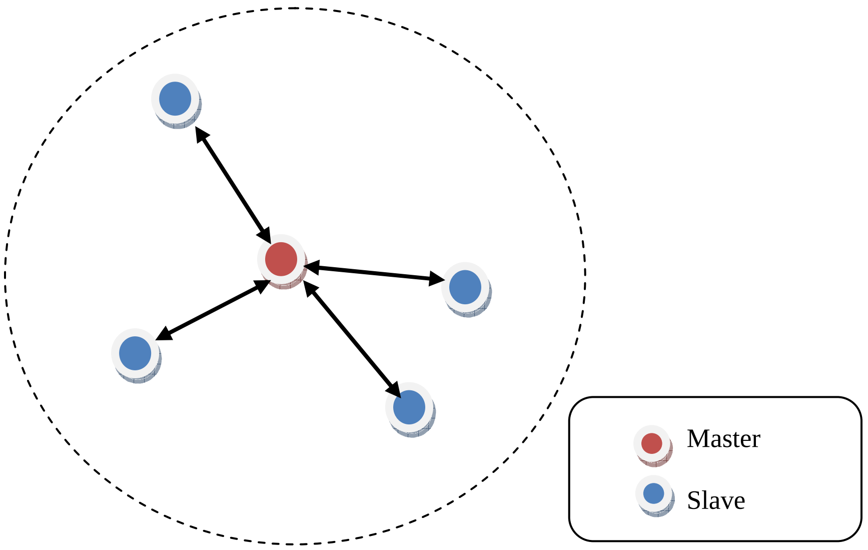

Figure 2: Bluetooth Nodes in a Piconet

In the above figure, a piconet is shown with, the red colored node acting as the master and all the blue colored nodes acting as the slaves.

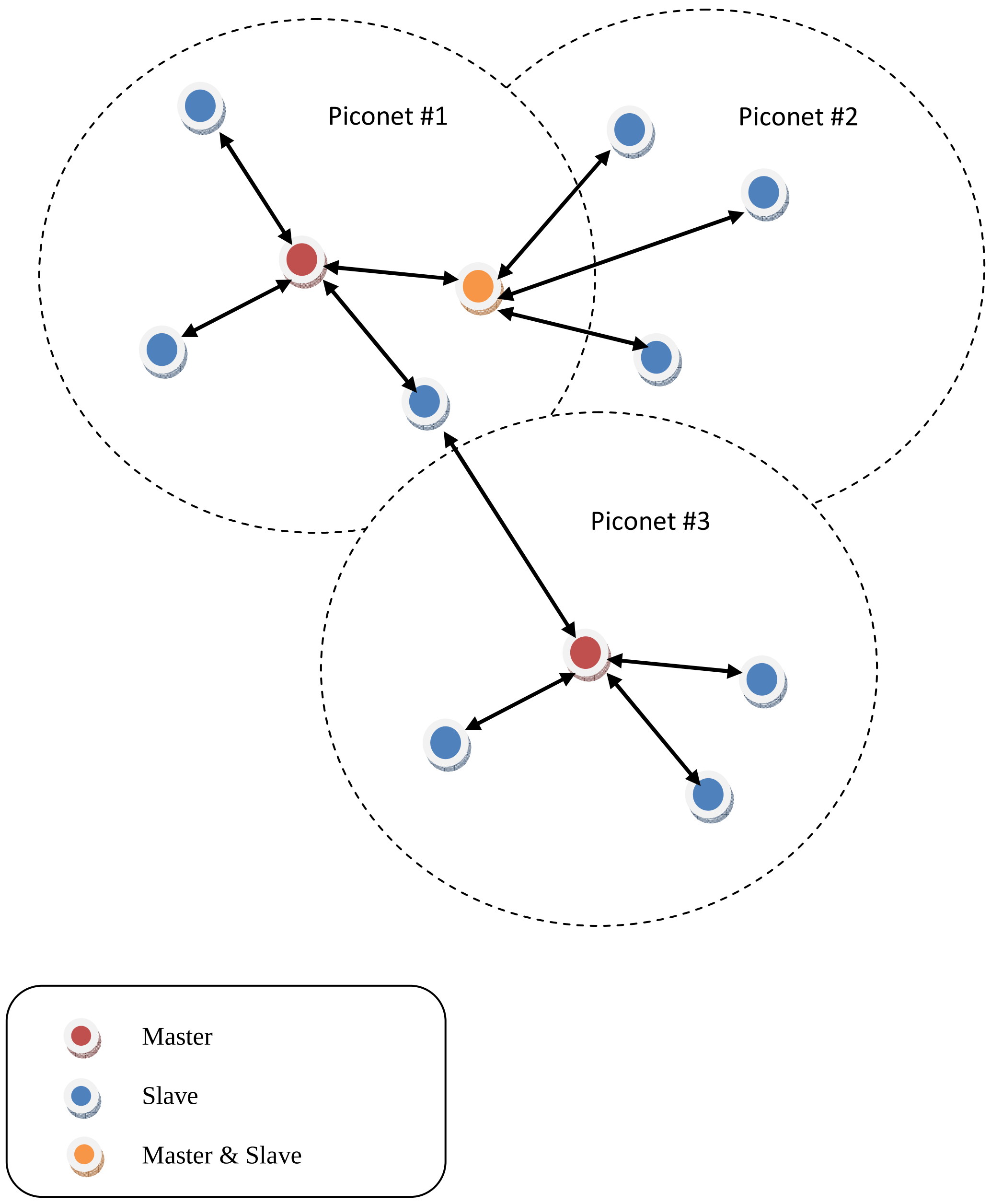

Figure 3: Three Piconets connected to form a Scatternet

In the above figure, two piconets are shown with, the red colored node acting as the master in the first piconet and all the blue colored nodes within the circle acting as the slaves. The orange colored node acts as a slave in piconet one and acts as a master in piconet two. For piconet two, the orange colored node is the master and the remaining blue colored nodes are the slaves. In piconet three, the slave node in piconet one is also a slave node in piconet three.

Bluetooth uses a Frequency Hopping Spread Spectrum (FHSS), to avoid interference with other Bluetooth Devices or Piconets. The entire bandwidth is divided into 79 channels of 1 MHz each, starting at 2.402 GHz and ending at 2.480 GHz. Frequency Hopping (FH) is employed to reduce the effect of Signal Interference and Fading.

Frequency Hopping (FH) is achieved by jumping between channels in a pseudorandom sequence. The two devices should share the same Hopping sequence, in order to communicate with each other, thus forming a Piconet. The hopping rate is 1600 hops per second and hence each channel is occupied for 625 micro seconds. As, there can be up to seven slave nodes that can communicate with the Master, Bluetooth uses a Time Division Multiple Access (TDMA) scheme for communication with the Slaves.

2.2.2. Power Requirements

Bluetooth specification classifies different devices according to the three different power classes:

Table 1: Bluetooth power classes

Power Class	**Maximum Output Power**	**Range**
Class 1	100 mW (20 dBm)	~ 100 meters
Class 2	2.5 mW (4 dBm)	~ 10 meters
Class 3	1.0 mW (0 dBm)	~ 1 meter

2.2.3. Operational Modes

The different operational modes of Bluetooth are explained below:

a) Inquiry (Discover)

Bluetooth devices use the inquiry procedure to either discover nearby devices or be discovered by devices in the locality. A Bluetooth device that tries to discover devices is known as the inquiring device and continuously sends inquiry requests. The Bluetooth devices in the vicinity that are available to be discovered are known as discoverable devices, who listen to the inquiry requests and sends the response.

b) Paging (Connecting)

The paging procedure is used to connect to the discovered device. The device that has been discovered is listening on a particular channel for connection request packets from the paging device.

c) Connected

After a successful connection procedure, the devices are physically connected to each other in a piconet. There is a piconet physical channel to which these devices are connected.

d) Hold

In this mode the physical link is active only during slots that are reserved for the operation of synchronous link types.

e) Sniff

This is a low consumption mode. The Bluetooth device in this mode listens to the piconet at regular intervals for a short instant. This helps it to stay in sync with the piconet and also to send and receive data.

f) Parked

A Slave node can be connected to a piconet and also be in a parked state. In this state the device does not support any logical links to the master except PSB-C and PSB-U that are employed for communication between master and parked slaves.

2.2.4. Bluetooth protocols stack

The Bluetooth protocol stack is an integral part of Bluetooth. Below mentioned are the layers:

A. LMP (Link Management Protocol)

This is used for controlling the radio link between the devices. It is usually implemented in the Bluetooth controller.

B. L2CAP (Logical Link Control & Adaptation Protocol)

Used for multiplexing multiple logical connections between two devices. It provides segmentation and reassembly of packets. It operates in the following modes:

Basic Mode - Provides packets with payload configurable up to 64KB, with 672 bytes as the default MTU.

Retransmission and Flow Control Mode - Can be configured for reliable data transfer by means of retransmissions and CRC checks.

C. <u>SDP (Service Discovery Protocol)</u>

This is used to allow devices to find out what services are being offered by other devices around. For example, when connecting a mobile phone to a Bluetooth headset, SDP will be used to determine which Bluetooth profiles are supported by the headset.

D. <u>HCI (Host/Controller Interface)</u>

The HCI standardizes communication between the host stack and the controller. There are several HCI standards each using different hardware interface to transfer the same command.

E. <u>RFCOMM (Radio Frequency Communication)</u>

RFCOMM is the cable replacement protocol used to create a virtual serial data stream. RFCOMM provides binary data transfer and emulates RS-232 control signals over baseband layer.

2.3. Peer-to-Peer File Sharing

Peer-to-Peer systems have been extremely popular on the internet. They are not limited to internet applications alone, but have also made a significance breakthrough in ubiquitous computing. Both peer-to-peer systems and ubiquitous computing are based on collaboration between independent, autonomous entities.

A lot of attention paid to peer-to-peer systems is due to the peer-to-peer file sharing systems.

Traditionally all the communication model between two devices has been a client-server model. A client requests a service or data and a server satisfies the request. A few examples are the World Wide Web (WWW), File Transfer Protocol (FTP), and Web Services. However, there are problems that are bound with this model. Firstly, such a system is hard to scale. Secondly, it presents a single point of failure – the server, and hence requires administration. Thirdly, it does not fully utilize the available resources.

Peer-to-Peer can be seen as an alternative to the client-server model, which was proposed to overcome the limitations of the client-server model. Peer-to-Peer encourages sharing of resources or data through direct exchanges between peers. A few examples of the resources that can be shared are data, collaborative work, storage space, and network bandwidth.

All the nodes in the system are peers. All the peers are potential users of a service and potential providers of service. Peers act as servers, clients as well as routers. Every single peer is independent and hence avoids the need for administration. Peers are dynamic; they join and leave the network at any point of their choosing. The resources are geographically distributed and hence such a network is easy to scale.

2.3.1. Resources in p2p systems

Peer-to-peer systems by definition are based on resource sharing model. The resources are typically computing resources, and these can be CPU cycles, memory storage, and bandwidth. Based on the resources shared, p2p systems can be classified into three classes:

2.3.1.1. Data Sharing

Data sharing systems are those where in the main resource being shared are storage space and bandwidth. A few examples of such class of p2p systems are file sharing systems such as eDonkey and BitTorrent. In such systems the nodes allow the network to store data and also to the bandwidth to fetch the data.

2.3.1.2. CPU Sharing

In some p2p systems, the main resource shared is the computation capability. A few examples of such systems are SETI@HOME and BOINC (Berkeley Open Infrastructure for Network Computing, 2007). In such systems a central authority divides the computational requirements among the peer nodes.

2.3.1.3. Presence Sharing

These systems instead of sharing resources are based on sharing human presence. A Few example of this category of applications are all instant messaging systems, peer-to-peer telephony applications such as Skype.

2.3.2. Classification of Peer-To-Peer systems

Peer-to-Peer systems are typically classified based on how the resources in the system are placed and located. The classification of p2p systems fall under either unstructured p2p system or the structured p2p system.

2.3.2.1. Unstructured peer-to-peer (p2p) systems

Most of the widely deployed peer-to-peer systems are unstructured. The main focus in unstructured peer-to-peer systems is how to locate peers with the desired resources. Unstructured peer-to-peer systems can be sub-divided into three classes:

I. Centralized

A centralized p2p system is based on a centralized indexing server. An example of such a system is shown below:

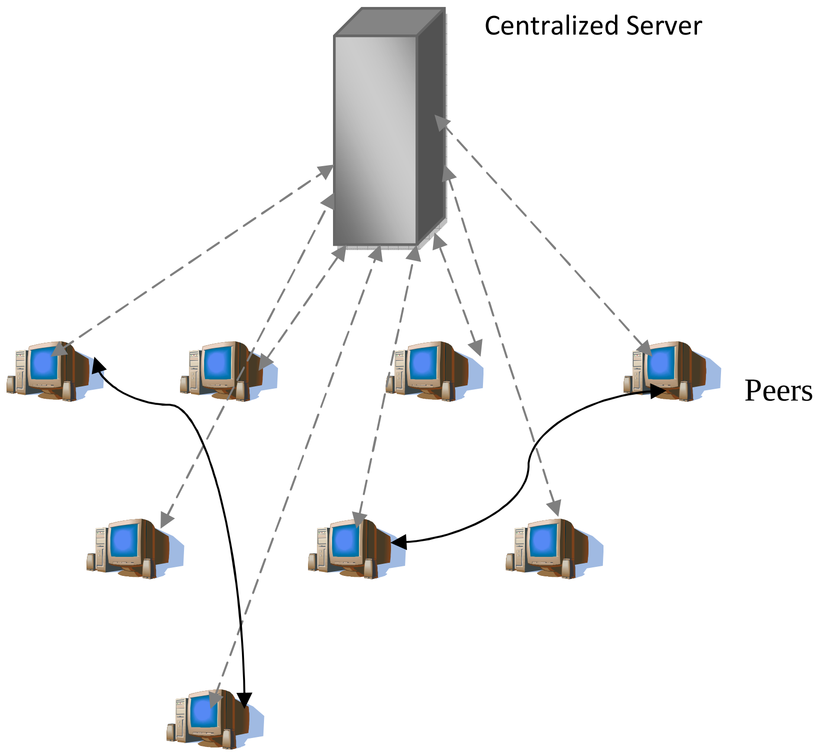

Figure 4: Centralized peer-to-peer network

The dashed lines represent overlay maintenance traffic and queries, whereas the solid lines refer to the actual file transfers. The initial Napster system was based on the architecture of a centralized peer-to-peer system. An advantage of the centralized peer-to-peer system is that the search for queries is efficient and results can be guaranteed to be correct. Since, the indexed centralized server knows about all the files in the network it can give successful results to almost all the searches.

II. <u>Distributed</u>

In a Distributed peer-to-peer system all the peers are considered equal. An example of such a system is shown below:

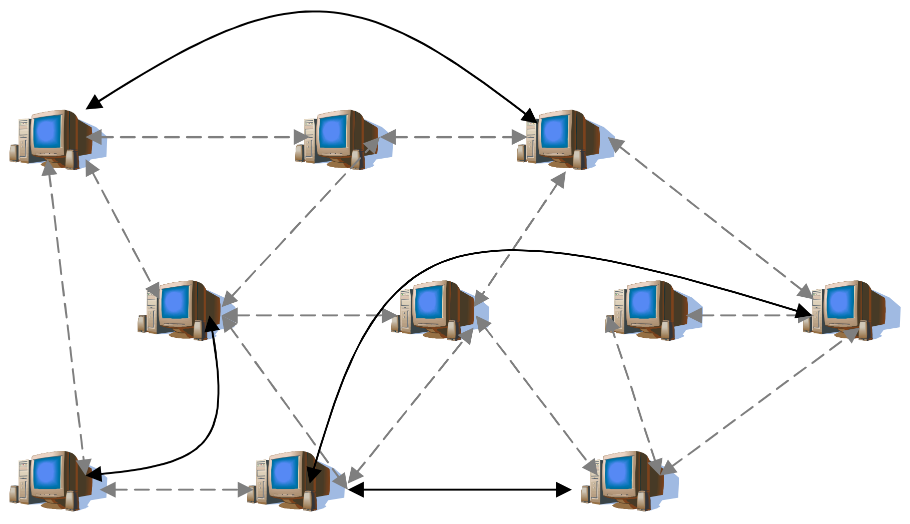

Figure 5: Distributed peer-to-peer network

The Gnutella network is based on a fully distributed architecture. The main disadvantage of a distributed peer-to-peer system is that the performance of query search is very slow. Searches are usually flooded within a certain range and only files in that range are discovered. The main advantage of such a system is the robustness as the system will never fail because no peer has a special role.

III. Hybrid

A hybrid peer-to-peer system combines both the centralized and the distributed systems. Some peers in the system have special roles and are called super-peers, which are assigned the task of managing a few other normal peers in the network and act as a centralized for them. The super-peers communicate among themselves using a distributed network similar to Gnutella. An example of such a system is described below:

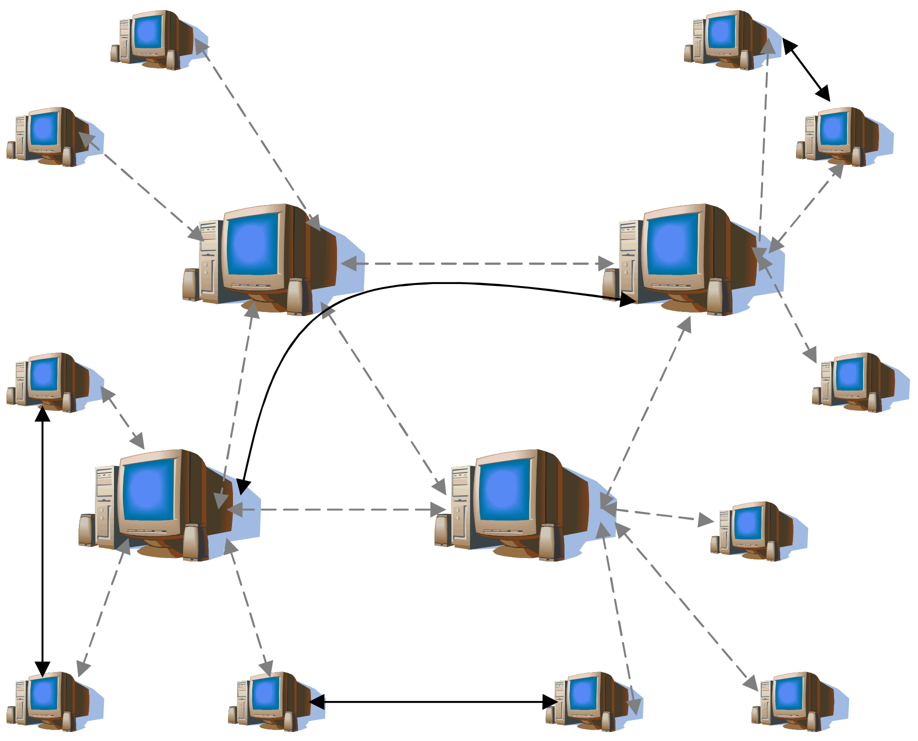

Figure 6: Hybrid peer-to-peer network

When a node wishes to search for a piece of information, he forwards the query to the super-peer. The super-peer knows which children of it have the resource also it can forward the query to other super-peers who answer for their respective children.

2.3.2.2. Structured peer-to-peer (p2p) systems

Structured peer-to-peer systems assign clear responsibilities to the peers in terms of the resources they need to serve. In contrast, in an unstructured peer-to-peer system a node is free to offer any piece of data. Files would be placed on only selected peer nodes and the request for the files would be directed to these nodes.

CHAPTER 3

RELATED WORK

Information exchange is most commonly used and extremely vital part of our day to day life. We exchange information with our customers, colleagues, and loved ones on a daily basis, be it sharing product catalogs or user guide with customers, sharing documents with colleagues at office or instant messaging your loved ones about things to do over the weekend, etc.

File sharing is the most common type of information exchange. This includes sharing product documents with your colleagues, sharing statistics about a player by the coach or the team, sharing term papers with your students, sharing project reports with your professor, sharing mp3's with your buddies in school, sharing movies with your friends in the neighborhood, etc.

We can broadly classify file sharing systems into two categories, one which uses the internet as the medium to share files and another where file sharing occurs between mobile devices within communication range.

The systems that employ internet to share files are very common, like the outlook mailbox, shared drives, internet servers distributing mp3's, or peer-to-peer applications. The systems that share files with other devices in the vicinity using a communication technology are typically mobile devices such as, cellular phones, smart phones, pocket pc, notebooks and netbooks.

Below we take a look at few of the file sharing applications for both the internet and mobile devices.

3.1. File sharing on the Internet

File sharing for the Internet has been there since a long time ago. Typically the traditional file sharing systems had client-server architecture, where the server is a node who has files to share and the client is the node who is interested in downloading of these files. The client would issue a search for a file on the server and if the file was present on the server, a communication channel was setup between the client and the server for file transfer.

The problem with the client-server architecture was that there needed to be a client who had to have these files and even bigger problem was that they needed to be running continuously. If the server were to shutdown the clients wouldn't be able to get the required files. There was no handling of dynamic joining and leaving of the server.

This is when peer-to-peer file sharing 2.3. , started gaining momentum. As in the case of peer-to-peer systems, every system acted as a client and a server if it has files to share. The peers could enter or leave the network at any point of time and this was handled gracefully. Also, in peer - to - peer, a file is broken down into many parts and is simultaneously downloaded from several sources. This greatly reduces the time taken to download a file, especially music and video files which are quite large.

Below we discuss traditional client-server architecture, Napster, Gnutella and BitTorrent architectures for file sharing applications on the internet.

3.1.1. Client Server

Traditionally file sharing over the internet would take place in a client server fashion. A server would upload files that he would like to share and a client would download the file from the server. A major problem with this architecture was that the server needed to be continuously on and hence would require administration.

A shutdown of the server meant that the clients would not be able to download any file. If the connection between the client and the server were interrupted during download then the partially downloaded file would be of no use and needed to be discarded. There was no handling of client joining and leaving the network. Such systems are hard to scale too.

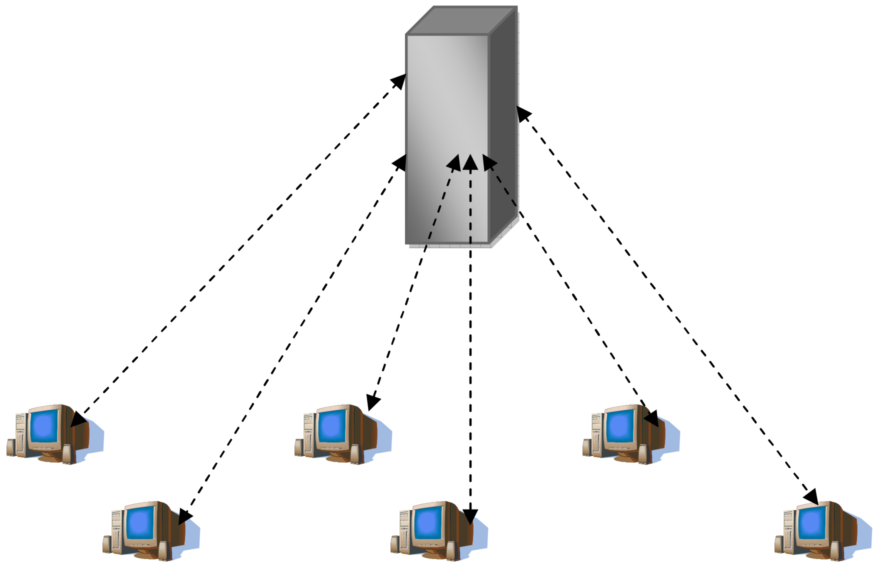

Figure 7: Client Server Systems

3.1.2. Napster

Napster [26] was an online music file sharing service created by Shawn Fanning while he was attending Northeastern University in Boston and operating between June 1999 and July 2001. Napster allowed people to easily share their MP3 files among each other, bypassing the established market for such songs and thus leading to the music industry's accusations of massive copyright violations.

The initial Napster system was based on the architecture of a centralized peer-to-peer system I. Such a system has the advantage that the search for queries is efficient and results can be guaranteed to be correct. However, it has a major drawback that all the participating entities in the network have to depend on the central index server for queries. If the central server goes down, then the entire network becomes useless.

3.1.3. Gnutella

The Gnutella network is based on a fully distributed architecture [II] as opposed to semi-centralized systems such as KaZaA [20] or centralized system such as Napster [26]. In the late 2007, it was the most popular file sharing network on the internet with the estimated market share of more than 40%.

3.1.1.1. Design

When a user in the gnutella network wants to do a search, the client sends a request to every actively connected node, which then forwards it to all their actively connected nodes until the packet reaches a predetermined number of hops. With the coming of version 0.6 gnutella became a composite network made of leaf nodes and ultra nodes (also called ultra peers or super peers). The leaf nodes are connected to a small number of ultra peers, whereas each ultra peer is connected to more than 32 ultra peers.

If a search request finds a result then the node that has the result contacts the searcher. In the traditional gnutella protocol the response message would take the same path backwards as the search request. If the user decided to download the file, they then negotiate the file transfer. If the node that has the result does not have a firewall then the requested node can connect to it directly. However, if the node is firewalled, the client wanting the file will send a so called push request.

3.1.1.2. <u>Protocols</u>

Since a peer-to-peer network like gnutella are dynamic and can grow to a large extent, the protocols employed must take care of different scenarios and must be resilient and robust. They should handle servent leaving and joining the network in a graceful manner. Below mentioned are the protocols employed by a typical gnutella servent.

Ping - This message is used to actively discover hosts on the network. A servent receiving the ping request is expected to respond with a pong message.

Pong - This message is the response to a ping request. It includes the information regarding the address of the connected client and the amount of data that it is making available to the network.

Query - This message is passed when the user searches for a particular file of interest. A servent receiving a query request responds with a query hit if the request matches its local data set.

Query Hit - This message is a response to a query message. This message is sent if the query matches with the local data set.

Push - This mechanism allows a firewall servent to contribute to the network.

One of the biggest advantages of gnutella is that it works all the time. As long as you can get to one other machine that runs the gnutella software, you can reach the entire network. However, gnutella does have few drawbacks. Since queries are flooded throughout the network, a response to a query might take a longer time. The user's machine uses some of the bandwidth to forward queries and responses as the user's systems are also a part of the network.

3.1.4. BitTorrent

BitTorrent [25] is a peer-to-peer file sharing protocol used for distributing large amounts of data. The protocol works when a user makes available a resource to the network. This is called a seed and allows other peers to connect and download the file. Each peer that downloads a part of the files makes it available to other users for download.

The peer distributing a data file treats the file as a number of identically sized pieces, typically between 64 KB and 4 MB each. The peer creates a checksum for each piece, using the SHA1 hashing algorithm, and records it in the torrent file. When another peer later receives a particular piece, the checksum of the piece is compared to the recorded checksum to test that the piece is error-free.

Users browse the web to find a torrent of interest, download it, and open it with a BitTorrent client. The client connects to the tracker(s) specified in the torrent file, from which it receives a list of peers currently transferring pieces of the file(s) specified in the torrent. The client connects to those peers to obtain the various pieces.

A growing number of individuals and organizations are using BitTorrent to distribute their own or licensed material. BitTorrent Inc. has amassed a number of licenses from Hollywood studios for distributing popular content from their websites. In 2008, the CBC became the first public broadcaster in North America to make a full show available for download using BitTorrent.

3.2. File sharing using Bluetooth

One of the major differences between file sharing applications for the internet or desktop and mobile devices is the fact that desktop is a fixed as compared to a mobile device.

File sharing for mobile devices over Bluetooth has been here for quite some time and there are lot of institutions and organizations that have active research in this area and have come up with lot of applications. According to the Bluetooth SIG [9], there are around one billion or more mobile devices that are Bluetooth enabled as of today. Almost every phone has the feature where one Bluetooth enabled device can search for other Bluetooth enabled devices around and pair up with that device. Once the devices are paired up data transfer can take place between these two devices. A user can select a file and select the option of sending it over Bluetooth to the one of the paired devices in the vicinity.

One of the major limitations of these applications is that it requires user interaction. Also not all the file types are supported. Some devices do not support executables to be sent to another mobile device and some do not support transfer of file larger than a threshold size.

As we discussed about opportunistic networks 0we discussed that the opportunistic networks are formed by spontaneous interaction of Bluetooth enabled mobile devices and such interaction will consider user intervention a huge drawback, as we cannot expect the user to select a device every time a new Bluetooth enabled mobile device enters the network.

There are a few existing implementations of file sharing over Bluetooth such as Bluetella [6], BlueMatch [7], BlueTorrent [7]. Below we discuss and take a look at a these few existing architectures for file sharing for Bluetooth enabled mobile devices.

3.2.1. <u>BlueMatch: Personal File Sharing in a Personal Bluetooth Environment</u>

BlueMatch [7] is a Bluetooth file sharing application, implemented as a MIDlet. The features that are already implemented are mentioned below:

Discovery of remote Bluetooth devices

Download the list of files available at the remote devices

Download the individual files from these remote devices

Share file to remote devices

Inform other nodes while shutting down

3.2.1.1. BlueMatch Client

Every instance of BlueMatch starts a client thread. This thread is used to discover other Bluetooth devices in the vicinity and to get the file lists from them. The client can connect to only one Bluetooth device at a time for BlueMatch to work properly.

3.2.1.2. BlueMatch Server

The multi-thread BlueMatch server is opens a service with Universally Unique Identifier (UUID) which is known to other Bluetooth devices running the BlueMatch application. This service can be searched by BlueMatch clients. For every client that tries to connect to this server until the maximum numbers of clients have reached, BlueMatch accepts and opens a connection to the remote client.

3.2.1.3. BlueMatch Operation

The user of BlueMatch starts the application and searches for Bluetooth devices in the vicinity. He then searches for the devices running BlueMatch. Once, he gets the list of devices running BlueMatch, he selects a particular device and sends a “Find Files” request to the device, which requires connection establishment. Once the remote device accepts this request, it sends out the list of files that it is willing to share. The user then selects the file and sends out a “Get File” request along with the file name, which he or she is interested in. The BlueMatch server receives this request and starts’ uploading the file to the remote device, the BlueMatch client then starts downloading that file.

3.2.2. Bluetella: A Java Application for New Mobile Phones

Bluetella [6] is a multi-hop file sharing protocol over an ad-hoc Bluetooth network developed by Ganymed Stanek at the Computer Engineering and Networks Laboratory (TIK) at ETH as his Master's Thesis. Unlike Gnutella, Bluetella is a file sharing protocol in which a user downloads or uploads a file from or to a one other user.

3.2.2.1. Bluetella protocols

As Bluetella is a multi-hop file sharing protocol, it necessitates for two application layer protocols, which are:

3.2.2.1.1. Bluetella source routing protocol

The Bluetella source routing protocol mandates broadcast and unicast message types. When a packet is created for the first time, the hop count field in the message is set to 0. On reaching the next hop the node increments and hop count by 1 and forwards it. If the message was previously received, the message is discarded. If the message was a new one the Bluetooth address of the sender is added to the hoplist. When a client wants to reply to the sender, all it has to do is to convert the broadcast packet into a unicast packet and send it.

3.2.2.1.2. Bluetella file sharing protocol

The Bluetella file sharing protocol consists of four message types. The "searchRequest" message is used to search for files among the Bluetooth devices in the ad-hoc network. The client who contains any file matching the request responds with a "searchResponse" message indicating the file name, size and the number of chunks the file will be broken into. On receiving the "searchResponse" message, the user sends a "packetRequest" message indicating the file id and the part of file, if the part of file number is zero it indicates that the entire file is required. The client, to whom the "packetRequest" message is sent, replies back with a "packetDelivery" message which includes the file part being requested.

3.2.2.2. Bluetella Implementation

The implementation of Bluetella is explained in detail below:

3.2.2.2.1. User Interface

The user interface provides a main menu to the user with the following options:

a) Explore neighborhood

b) Search and download

c) My shared directory

d) Settings

3.2.2.2.2. Threads

Bluetella relies on multiple threads to respond to user interactions. For every new client request Bluetella server creates a new server thread to satisfy the request of the client. For every new search by the user results in a client thread being created to search and download the file from the Bluetella ad-hoc network.

3.2.2.3. Bluetella Development and Test

Bluetella was developed on a Sun ONE Studio development platform. Emulators of Siemens mobile phones were chosen to access the mobiles file system. Rococo Impronto Simulator was used to simulate the JSR-82 API and Bluetooth. Interruption during the file transfer process, neighbors becoming unavailable hasn't been handled gracefully in this version of Bluetella and is left out for future work. Graceful termination of threads again is left out for future work. Another likely scenario is a one of the intermediate hop node disappearing, this situation is currently not handled in Bluetella and is left out for future work.

3.2.3. Bluetella: File Sharing for Bluetooth Enabled Mobile Phones

Bluetella: File Sharing for Bluetooth Enabled Mobile Phones, by Andreas Weibel, Lukas Winterhalter at the Computer Engineering and Networks Laboratory (TIK) at ETH, is based on Bluetella, a Java application that we just described in the previous section by Ganymed Stanek at the Computer Engineering and Networks Laboratory (TIK) at ETH. However, the previous implementation of Bluetella was not tested on an actual Bluetooth device due to lack of resources. In this version, the authors have implemented and tested the Bluetella protocol on a Bluetooth ad-hoc network.

3.2.3.1. Bluetella features

A few of the features present in Bluetella are explained below:

3.2.3.1.1. Searching files

The search procedure allows the user to search for data items by providing as complex a query as possible, because a simple search might get lot of results.

The user can either enter a simple string as a search string or he can enter extra details regarding the search, like the file type, artist, album, year, genre, etc.

3.2.3.1.2. Spreading files

A persistent multi-hop network is not feasible with the given Bluetooth hardware. The connections between the hops take a very long time as the configurations of the piconets have to settle down. Hence, say forwarding a result can take several minutes to reach from the sender to the receiver.

3.2.3.1.3. Downloading

The files could be split up into chunks and then the user could download any chunk of interest or he could download the entire file at once. The disadvantage of chunk download is that if the connection is interrupted during a download, the entire chunk has to be dropped. With downloading the whole file you don't have that problem, the file status can be stored to the last byte received.

3.2.3.1.4. File access

Record stores are used to save the preferences data and the internal list structures containing file and search information. The File Connection API is used to manage the files that are shared or have been downloaded.

3.2.3.1.5. Identifying files

Using the file names alone is not sufficient to identify files uniquely. Hence, for every file accessed by Bluetella is allocated with a unique hash. The hash is called the file id is used to identify and manage files.

3.2.3.1.6. Security

All the applications need to be signed for security purposes. Also, it is not possible for Bluetella to securely authenticate any peers in the system.

3.2.3.2. Bluetella protocols

The protocols described in section 3.2.2.1 are used for this version too.

3.2.3.3. Bluetella implementation

Bluetella implementation details are explained below:

2.2.3.3.1. Server thread

The server thread has to start the service in order to be discovered by other devices. It does so by making itself discoverable and setting the General Inquiry Access Code (GIAC) to a known value. The server thread manages the task of listening to the incoming requests. For every new connection, a new thread is spawned to satisfy the incoming requests.

3.2.3.3.1. Client thread

The client thread searches for clients in the vicinity running Bluetella. It does so by looking for devices running the service specified the Bluetella server. For every new device found, a new search thread is spawned whose job is to search for the required files with the device.

3.2.3.3.2. Indexing files

The FileIndexer class provides methods to search for files in the folders that the user has decided to share. All metadata information regarding the file is extracted. The file names, size, extension, type, id, and the last modified time. Also some special tags such as ID3 tags for audio files are also extracted.

3.2.3.4. Bluetella test results

Bluetella test results are classified into functional and performance tests.

3.2.3.4.1. Functional tests

The peer discovery process does not always accurately retrieve phones running Bluetella due to the limitations of connection setup in piconets. A large number of advanced searches were provided as query and the result were matched against the manual results and they seemed to be in sync. Downloading of files works fine with files of size one hundred kilobytes and less. Multiple file downloads is possible but happens one after the other. Any interruption during file downloading is handled gracefully. With one recharge of the battery, Bluetella was able to run for five hours constantly exchanging data.

3.2.3.4.2. Performance tests

It took about 20 to 40 seconds to discover peer devices and to establish first connection, with an average time of around 29 seconds. The hash calculation of a file of size 4.6 megabytes takes about 6.7 seconds per megabyte. The measured download speeds lay between 39.2 kilo bits per second to 70.2 kilo bits per second. Bluetella allocated between 700 kilobytes to 1500 kilobytes based on the number of connections.

3.2.3.5. Conclusion

The tests showed that Bluetella works pretty well as projected. Noticeably there are some drawbacks. The connection establishment between two peers often took a tiresome amount of time. The used Bluetooth devices were not stable and produced crashes at times. Overall, Bluetella does serve the intended purpose but it does not even come close to the speeds achieved by internet file sharing applications.

3.2.4. BlueTorrent: Cooperative Content Sharing for Bluetooth Users

BlueTorrent [7] is a peer-to-peer (P2P) file sharing application for Bluetooth enabled devices. BlueTorrent is quite similar to BitTorrent where files are split into smaller chunks before being sent to neighboring nodes. A key difference is that BlueTorrent supports only one hop nodes. BlueTorrent shares content by file swarming.

A prerequisite for BlueTorrent to share content is to know where the content is, which requires randomly installed Bluetooth Access Points, which act as Bluetooth servers broadcasting files that they have. A user of BlueTorrent system can now search for the content by providing search tags, such as movie name or song title, etc and can start downloading files. On finding a server, a user sends a query to the server, which then checks if it can satisfy the request or not, if it can it sends the metadata of the file being requested back to the user. The user can then decide if he would like to initiate a download or not.

For the simulation setup, there are two kinds of nodes used one AP nodes that acts as a server and the other being mobile nodes. Initially only the AP nodes can transfer data. Once the mobile nodes receive data they can re-distribute the data in a P2P fashion. From the simulation results it can be concluded that the P2P mode has 3 times better performance than AP mode.

CHAPTER 4

DESIGN AND IMPLEMENTATION

By definition opportunistic networks are formed dynamically when a set of mobile nodes, those have something to share with each other, come within physical proximity. A user’s device should connect to, share information, and gracefully disconnect from these devices seamlessly, even without the user knowing about it. For such a system any kind of user intervention is a huge drawback.

The applications that we had a look at in the previous chapter, such as BlueMatch 3.2.1, BlueTella 3.2.2, and BlueTorrent 3.2.4 are not meant for such dynamic network. These belong to a category of standalone applications where the user searches for a set of devices in the physical proximity and if he finds a device, he then goes ahead and requests for a particular piece of information on that device. If the remote device consists of that information, then the user then decides whether to download it or not.

This lack of a complete end-to-end file sharing system, for opportunistic networks, is a motive for us to come up with such a system. An end-to-end system, where the user can specify a set of files that he is interested in and then the system takes up the responsibility of finding these files and downloading them from one or multiple sources.

As a pre requisite, the user of a system should enter the files or file types that he is interested in, so that this search criterion could be sent to the remote device and appropriate files be found. Also, the user of a system should be able to prioritize the files that he is interested in, so that the system can capitalize on this opportunity to get high priority files before getting low priority ones.

On startup the system should create server thread to accept requests from other Bluetooth enabled devices. Also, the main thread is responsible to look for other Bluetooth enabled mobile devices in the vicinity. For every device found the system should check if a connection could be established to the remote device and if the system was able to so, then the main thread creates a client thread allowing the client thread to handle requesting of files from these devices.

The server thread takes on the responsibility of accepting connection requests from remote mobile devices, check if it can support an additional connection to a remote device without going above the predefined threshold. The server thread checks if the files or file types mentioned in the search criteria matches with the files on the local device. If there are no files that match the search criteria, then the device sends an abort message to the remote device. If the device finds at least one file that matches this criterion, the device sends out the file name and the file size in bytes to the remote device. Before doing so, the server thread creates another server thread to handle further requests from the remote devices regarding getting files.

The client thread reads the files or file types that the user of the system is interested in and forwards this search criterion to the remote device. The search criterion consists of files or file types arranged in the order of their priority, so that the remote device can reply back with the file names with the priority intact. The remote device replies back to the search criterion with an abort message or a response that contains the file names and file sizes. For every file, the client thread checks if the file is already present on the local device or not. If it already does, ignores the file. If the file does not exist, it checks if the file is partially present, if it is then requests for that part of the file that is not present. If the file is totally new, then the thread creates a folder for the file and creates as many number of file pieces based on the size of the file and the predefine size of the piece.

The user of such a system should also be able to not only download files from other users, but also should be allowed to share files to other users, if need be. Hence, the client thread also includes the push mechanism, where a user of a system marks the files to be pushed and whenever the system comes in contact with other mobile devices these files are pushed to the remote device.

The features described above might pose a problem for energy constrained mobile device. This system has to be high performing and at the same time be energy efficient. In order to do so, the files are compressed before transfer at the sender. This reduces the overall time taken to download the file, also reduces the bandwidth utilization thereby improving the lifetime of the mobile device.

Also, we employ other ways to reduce the amount of energy consumed. A predominant example being, if a connection to a remote device was rejected once, due to the fact that the remote device does not support the application, the system should not try to connect to this remote device again in the future until a predetermined point of time. This will avoid unnecessary connection requests messages being sent to the remote device, and thereby increasing the efficiency of the system. The same goes for a remote device from which all the files are successfully downloaded in the past. The addresses of these devices are added to a list to check if a connection to these devices is necessary or not, and are removed from the list after a certain pre defined period of time.

4.1. Mobile Device

The wireless mobile device that we use to implement this system on is HP iPAQ 910 Business Messenger. The HP iPAQ 910 Business Messenger is quad band mobile with support for integrated multi-mode GPS navigation with Google Maps. The phone uses the Windows Mobile 6.1 Professional operating system and runs on a Marvell PXA270 Processor operating at 416 MHz. The device is integrated with Wi-Fi - 802.11b/g with WPA2 security and Bluetooth v2.0 with EDR.

The display is a 2.46-inch transmissive TFT 65,000 color 320 x 240 pixel touch panel with LED back light. It consists of 128 MB SDRAM main memory for running applications and 256 MB flash ROM. The mobile device is equipped with a 1940 mAh Lithium Ion battery.

4.2. Programming Environment

The mobile device runs the Windows Mobile 6.1 Professional operating system and hence we use the Windows Mobile Development Framework that is based on the .NET framework from Microsoft.

The system requirements to begin development for Windows Mobile based Devices are:

Visual Studio 2005 (with Service Pack 1)/ Visual Studio 2008

Windows Mobile Software Development Kit

Active Sync/Windows Mobile Device Center

4.3. User Interface

The user interface has to be simple and intuitive. The user interface displays the currently discovered Bluetooth devices around. Additionally it has to display the current number of uploads and downloads for the user to get a good information about the behavior of the application.

The user interface should also display the file names that are currently being downloaded and the percentage of download completed.

4.4. Message Types

As file sharing requires quite a few message exchanges between the user's local device and the remote devices, there needs to be a set of predefined messages for different operations. The different operations are connecting to a remote mobile device, responding to a request from a remote device, aborting a connection, requesting a file, responding with the appropriate file, and forwarding a file. There are eight message types that we use in our system.

4.4.1. Connection Request

The Connection Request message type is used by the client to request a connection with the Server. The attributes passed in the request are the request type, i.e. PULL or PUSH and the search query, i.e. the files that the client is interested in.

REQUEST TYPE	SEARCH QUERY

Figure 8: Connection Request Message

4.4.2. Connection Response

The server on receiving the Connection Request message checks if the connection can be accepted. If it can, the server creates a server socket and assigns a thread to listen on that socket. The server also gets the port at which the server is listening and adds it to the output buffer. The server then reads the users query and checks for the files that match the query and add the file names and the file sizes to the output buffer.

PORT	FILE NAME	FILE SIZE	. . .	FILE NAME	FILE SIZE

Figure 9: Connection Response Message

4.4.3. <u>File Pull Request</u>

Once the client receives the Connection response, it reads the port number at which it needs to contact the server and creates a thread to sync with that port. The client also hands over the results of the query to the thread. The thread checks if the file is already present on the device, if so it goes to the next file. If the file is partially present on the device the client thread start to get the next needed part of the file. If the file is not present on the device, the client first breaks the file into many pieces of same size and starts downloading from the first piece. The client thread now starts to download the files sequentially from the server. For downloading the file, the client requests the file name, starting byte and the ending byte.

FILE NAME	STARTING BYTE	ENDING BYTE

Figure 10: File Pull Message

4.4.4. <u>File Pull Response</u>

Once, the server thread receives the File Pull Request, the server then sends the response by adding the file name and packet number equal to zero indicating that the server is ready to upload files.

FILE NAME	PACKET NUMBER

Figure 11: File Pull Response Message

4.4.5. File Data Response

Once, the server thread sends a File Pull Response, the server thread calculates the number of packets to be sent and starts sending the packet one by one. The client thread also does this calculation and hence knows how many packets to expect from the server. The file checksum is used to identify that the packet belongs to the current download. The packet number is used as an indicator representing the current packet number. Number of bytes represents the number of data bytes the packet contains; the client uses this to read that much amount of data.

FILE CHECKSUM	PACKET NUMBER	NUM OF BYTES	PACKET DATA

Figure 12: File Data Response Message

4.4.6. File Push Request

The file push request is important when the user has some files that he wants to distribute, like say a researcher wants to distribute one of his publications to other researchers for them to have a look at them or a meeting request for a talk he is holding in a few days time.

REQUEST TYPE	FILE NAME	FILE SIZE	. . .	FILE NAME	FILE SIZE

Figure 13: File Push Request Message

4.4.7. File Push Response

On receiving the file push request the server decides whether he wants to download the file or not. If he does decide to download the file, the server will create a server socket to download the file and pass the port number to the client. Here, the client is pushing the file and the server is pulling the file.

PORT NUMBER

Figure 14: File Push Response Message

4.4.8. Abort

We use the abort message to inform the client that the server is not ready to accept connections.

4.5. Main thread

The main thread is responsible for initializing the user interface and to starting the server thread. The main thread scans for other Bluetooth devices in the vicinity. For every Bluetooth device found it does the following:

Checks if the number of Bluetooth devices to whom active connection is established is greater than a given threshold.

Checks if an active connection is already established between the Bluetooth devices or an active connection was established between the Bluetooth devices in the last say one hour.

Checks if the remote device runs the required service or not. If the remote device does not run the service, then there is no use trying to connect to the device.

Once it passes all these constraints, the Bluetooth device issues a connection request message to the remote Bluetooth device.

If the remote Bluetooth device responds with a connection response message, then the device picks up the port number at which a new server socket has been created and tries to establish a connection to that server socket.

If the connection is successful, the device creates a thread and passes to it the connection socket and the files shared by the remote Bluetooth device.

4.6. Client thread

The client threads objective is to retrieve files from the remote device, gracefully handling errors, Bluetooth communication link breaks, file splitting, indexing and joining.

When the client thread finds a file that is not present on the local device, it splits the file into a number of pieces of pre determined sizes. Also, a metadata file is created which keeps track of these files, their starting and ending bytes and the status of each piece.

Until the entire file is downloaded, the thread finds out the next piece to be downloaded and checks if the server has the piece. Also, the device calculates the number of packets the server will be sending for each piece, so as to make those many number of send or recv calls to the socket. Once, the calculations are done, the device starts downloading the file from the server.

While downloading the file from a remote device, if the devices go out of communication range, then the file is saved midway, so that next time, the same thread or a different one trying to download the file need not download the already downloaded part of the file.

Once the last piece of the file is downloaded, the thread that downloaded the last piece is responsible for merging the pieces to form a single file. This might seem simple, but is actually quite tricky. Let us look at an example, in a piconet a master node can communicate with up to seven active nodes at a time. Let us imagine that the entire seven slave nodes have the same file which the master node is interested in and the master nodes begins downloading the file simultaneously from the slave nodes. The seven threads get the last remaining seven pieces to download. When the last pieces are being downloaded, if one of the pieces is taking long time to download, then the thread downloading the last file should not merge the file pieces, but instead wait for the threads to finish downloading the pieces.

The client thread is also responsible for forwarding files that user wants to be distributed to the remote mobile devices. The client thread takes over the responsibility of the server thread of pushing the file to the remote device. The client thread connects to the remote device and sends a push request, thereby enabling the remote device to request for the file that is being shared by the local device. The client thread then starts forwarding the file to the remote device.

4.7. Server Thread

When the application is started a server thread is created whose main objective is to listen to the incoming requests and to respond to them. As mentioned in the title the server handles PULL and PUSH requests. On startup the server thread is created and the service is registered for other Bluetooth enabled wireless devices to search for this service. The server on receiving a Connection Request, recognizes the request type, i.e. either a PULL or a PUSH request.

4.5.1. PULL Request

The thread checks if the remaining battery on the mobile device is sufficient enough to serve the request, if not it sends an Abort message to the client that requested.

If the available battery on the mobile device is sufficient to serve the request, the server then creates a Bluetooth server socket by allowing the device to select an unused port.

If the creation of server socket is not successful due to any reason, the server sends an Abort message to the client.

If the creation of server socket is successful, then the server adds the port number to the output buffer. It then reads the clients query, i.e., the files that the client is interested in.

The server then looks at the files available on the device to share and if it does not find any, it then sends an Abort message to the client.

If the server finds at least one file that matches the clients search criteria, he then copies the file names and sizes to the output buffer.

Finally, the server creates a thread, which waits for connection from the requested client on the port mentioned in the output buffer. The server thread responds with a Connection Response message.

4.5.2. PUSH Request

The thread checks if the remaining battery on the mobile device is sufficient enough to serve the request, if not it sends an Abort message to the client that requested.

If the available battery on the mobile device is sufficient to serve the request, the server then creates a Bluetooth server socket by allowing the device to select an unused port.

If the creation of server socket is not successful due to any reason, the server sends an Abort message to the client.

If the creation of server socket is successful, then the server adds the port number to the output buffer. It then reads the clients file names, i.e., the files that the client wants to push.

The server creates a thread, which waits for connection from the requested client on the port mentioned in the output buffer and starts requesting files provided by the client.

4.8. Priority based file downloading

The user of the application should be able to set priorities to the files he is interested in. For example the user might want more number of downloads, for e.g. if he is interested in documents, applications, image files, audio and video files, then the user would want the device to download the documents first, followed by image files, then applications and finally audio and video files.

The size of audio files are typically above 5 mega bytes and hence would take a longer time to download as compared to documents or image files. Hence by allowing the user to prioritize his preferences, we allow him control the sequence of downloads. So, the user can probably set documents and image files as high priority, followed by audio files and finally the video files. Thereby increasing the number of file downloads.

4.9. File Compression

If the user of a system has a files of several mega bytes, and if the file is compressed using any of the compression techniques, the file reduces to a very small size. Sharing this file with other users is much more beneficial because of the amount of time saved and communication duration is low, which results in preventing the battery from discharging. Typically, compressing documents, text and image files using Pocket RAR which is a lossy compression technique would result in compression ratio in the range of 80% - 99%.

The file compression can be on the fly or beforehand. If the compression is on the fly, then this would add the time taken to compress files to the final download time. As you will see from the results below, the time taken to compress text, documents or image files of several mega bytes are a few seconds.

CHAPTER 5

TESTS AND RESULTS

The tests are categorized into functional and performance tests and the results are mentioned along with the description of the setup and the parameters used.

5.1. Functional Tests

The functional tests are classified into device discovery, peer-to-peer file sharing, priority based file downloading, and file compression.

5.1.1. Bluetooth device discovery

The Bluetooth device discovery duration is when the Bluetooth enabled device searches for other Bluetooth devices in the vicinity. Typically this time varies from 10 seconds to 17 seconds with an average of 12 seconds. This procedure also depends on the number of mobile devices present in the vicinity.

5.1.2. Peer-to-Peer file sharing

The peer-to-peer file sharing works as mentioned. The Bluetooth enabled device informs the remote devices about the files or file types that the user is interested in. The remote device replies back with a list of files matching the search criteria. The Bluetooth enabled device then requests for the files shared by the remote device one by one. If the file to be downloaded is already being downloaded from another device, then the device requests for the next piece of the file that needs to be downloaded. We used different types of file and complex search criteria's. The file types that were used are executables (.exe), audio files (.mp3, .wav,), video files (.mpg), documents (.doc, .xls, and .txt), and image files (.jpg, .bmp), etc.

5.1.3. File priority handling

The priority based file sharing works as described. The user of a Bluetooth device informs the remote device of the files or file types that he is interested in. The remote device then replies back to the user with the files that match the search criteria and also adheres to file priority. For e.g. if the files that the user is interested in are image files, audio, and video files and in the same priority, the remote device replies back with image files first, followed by audio files and finally the video files. Thereby, enabling the user to download the files in the priority that he would like to.

5.1.4. File compression

File compression also works as described. Compression of text files of size several mega bytes reduce by almost 70% – 99% in size. This way the number of packets required to transfer the file to the remote device is less by a huge margin. Also, this means that the time required to download a compressed text files is much lesser than downloading a non compressed file. This also reduces the amount of battery required for the operation.

5.2. Performance Tests

The performance tests are conducted to measure the performance of the system, their behavior under stress. Performance tests are of great importance for mobile devices as they are energy constrained devices and the software must be rigorously tested in order to ensure that the software does not drain out the battery. We conduct performance tests in various environments, under various stress levels, with different number of devices in the vicinity every time.

5.2.1. Peer-to-Peer file sharing

The performance tests for peer-to-peer file sharing are classified based on the file sizes. These tests were carried out in an indoor environment. The environment consisted of Wi-Fi signals and other Bluetooth enabled devices not running the application in the physical proximity.

Below mentioned are the results for the experiments conducted in an indoor environment. There were about 5 different Wi-Fi signals present and as it was a closed area scattering of signals were evident. The different file sizes used were 500 kilo bytes, 2 mega bytes, 6 mega bytes and 10 mega bytes. The time taken to download files from the peers in the vicinity was measured, and the data rate calculated.

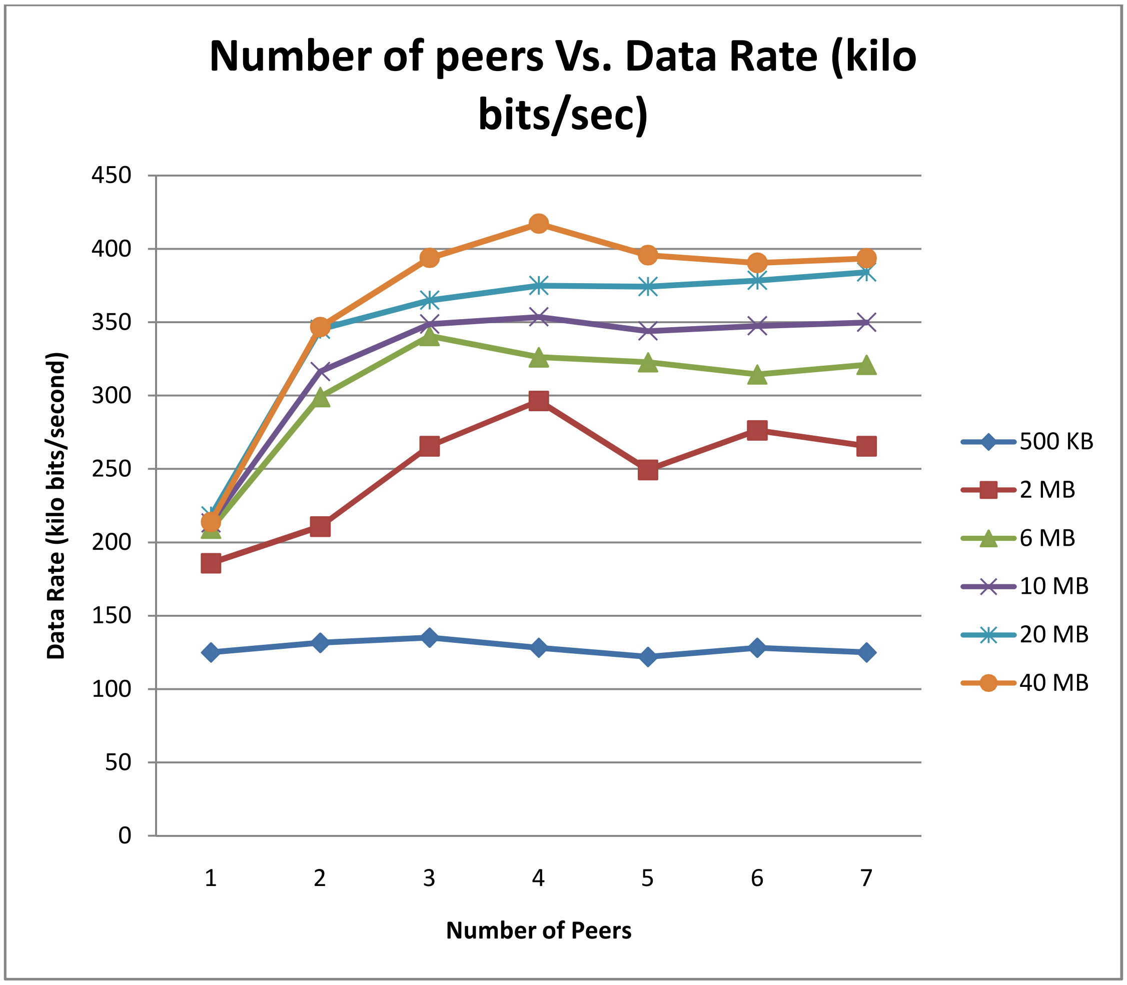

Figure 15: Number of Peers vs. Data Rate (kilo bits/second)

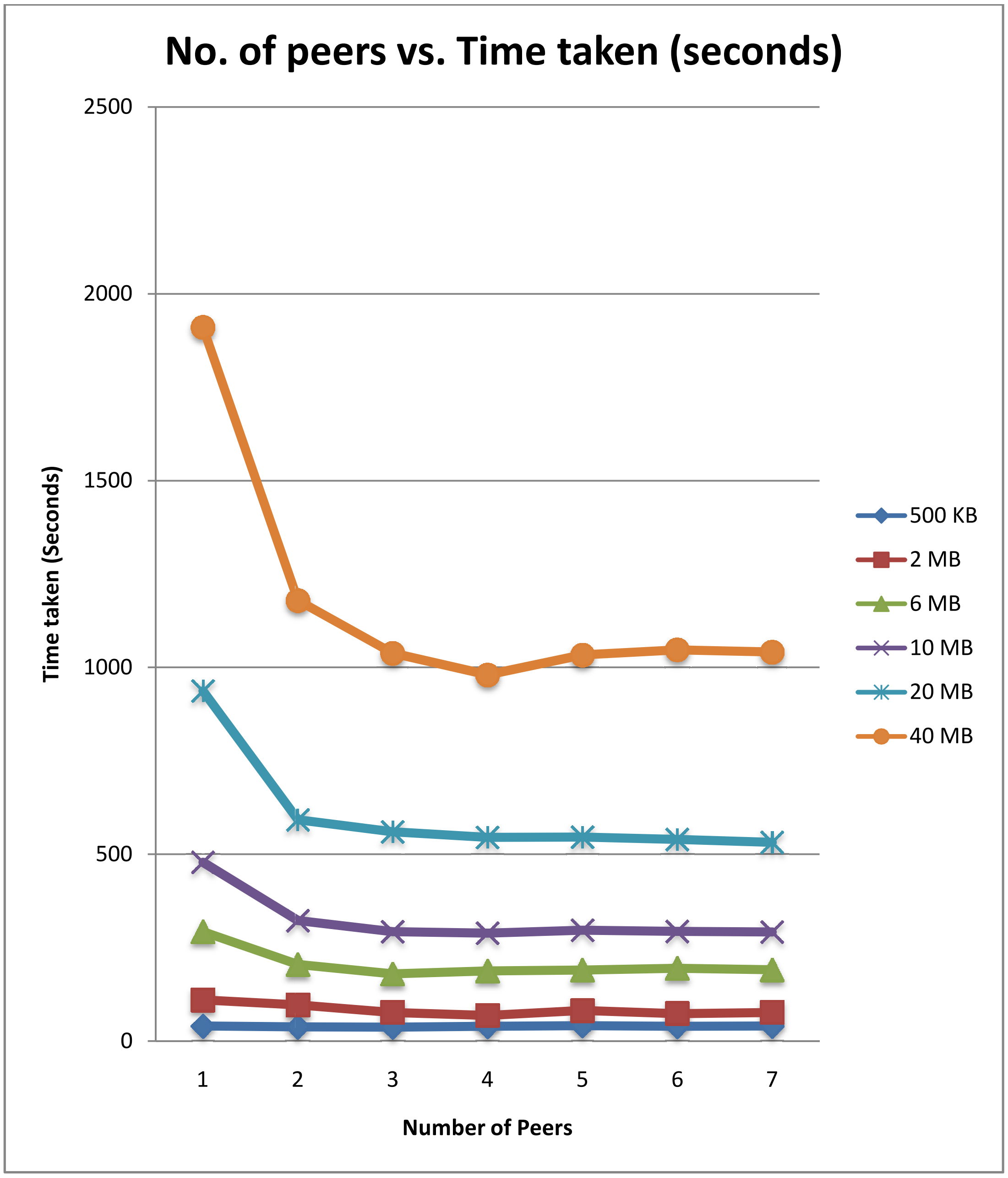

Figure 16: Number of Peers vs. Time taken (seconds)

We can also notice that, as the number of devices from which we are downloading a file increases, the time taken decreases. However, when the number of peers is greater than four, the time taken to download a file remains a constant and does not decrease. This is because the master node has to connect to all the devices using Time Division Multiple Access (TDMA). This stops us from completely capitalizing from the fact that we can simultaneously download from up to seven devices at any point of time. We can limit the number of downloads in one piconet to 3 or 4, thereby maximizing the efficiency of the system and minimizing the load on the system.

5.2.2. Merging files

As the system is a peer-to-peer file sharing, a file has to be split into multiple pieces to simultaneously download different pieces from different peers. Hence, the size of the piece is an important factor and hence measured. The sizes used varied from 10 kilo bytes to 500 kilo bytes and the time taken to break the file into multiple pieces and the time taken to join these individual pieces were measured.

The time taken to create multiple empty pieces of a file took about 1 second on an average irrespective of the size of the file piece. However, the time taken to join these individual pieces varied. Mentioned below are the details for the times taken by the system to merge multiple files to form a file of size 10 mega bytes.

Table 2: Time taken to merge files

File Piece Size	Time (seconds)	Time (seconds)
10 KB	267	1876
50 KB	80	762
100 KB	75	531
200 KB	72	588
300 KB	78	667
400 KB	80	846
500 KB	80	1100

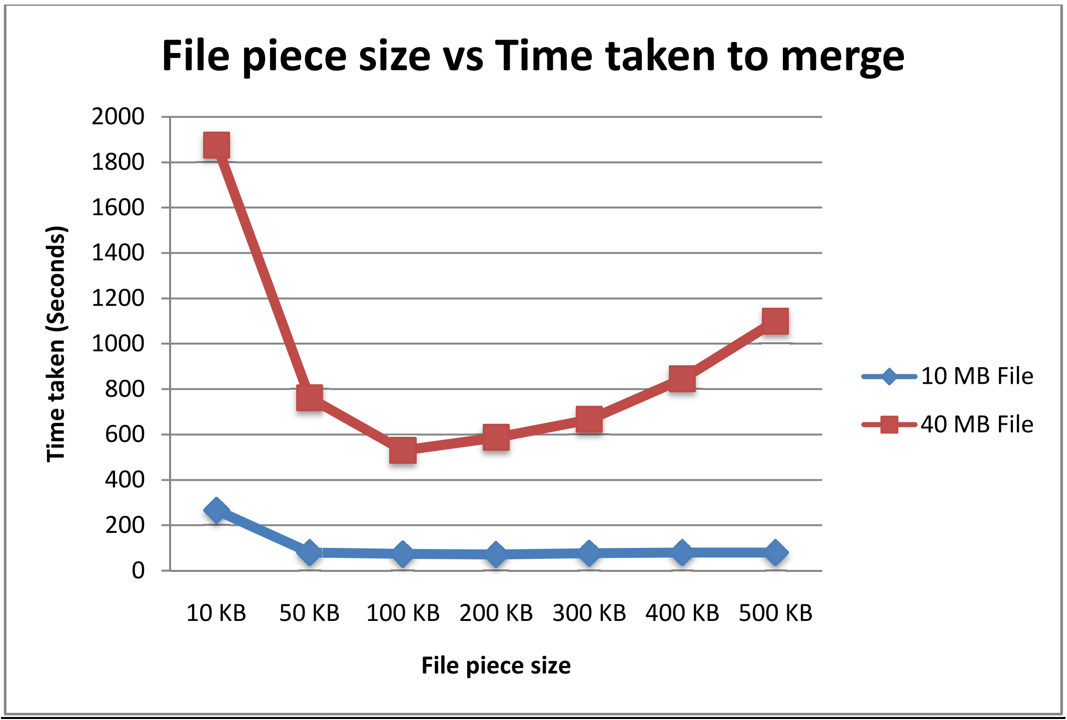

Figure 17: File size vs. Time taken to merge

When the file piece size is 10 KB, the number of pieces is 1024. Hence the time taken to merge these files into 1 file, takes an extremely long time. As the file piece size increase over 50 KB to around 500 KB, the file merge time is almost in the same range, between 70 – 80 seconds.

5.2.3. File compression

File compression helps in reducing the file size, thereby reducing the communication duration, and the battery usage. The compression ratios for text files and image files using Pocket RAR is anywhere between 70% - 95%.

Table 3: File types vs. Compression ratios

File Type	**Compression Ratio**
*.txt, *.pdf, *.xls	70% - 99%
*.doc	< 25%
*.jpg, *.bmp	90% - 99%
*.mp3, *.wav	< 10%
*.mpg	< 10%

For text files like .txt, .pdf, and image files like .jpg and .bmp, the compression ratios are extremely good in the range of 70% – 99%. But for documents like .doc, audio files .mp3, and video files .mpg, the compression ratios are extremely poor or there is hardly any compression. Also it takes long time to compress audio and video files. Hence, we would require some other compression mechanism, with a good compression ratio for multimedia files.

5.2.4. Battery consumption

We also monitored the battery usage of the mobile devices to better understand the impact of the system on the battery constrained mobile device.

We found that when a mobile device is uploading a file to another peer, the battery level reduced by 1 (on a hundred scale), for file sizes from 500 kilo bytes to 10 mega bytes.

Also, we noticed that for a system downloading a file from one or more devices, the battery level reduced by about 3 (on a hundred scale), for file sizes from 6 mega bytes to 10 mega bytes. The battery level reduced by 2 for files below 5 mega bytes.

The reason for this behavior is that the mobile device downloading a file acts as master node in the piconet and hence has to frequency hop and talk to the slave devices in the piconet using TDMA. In contrast, the slave node only has to hop along with the master node and send data when requested.

CHAPTER 6

CONCLUSION AND FUTURE WORK

We have successfully implemented a mobile file sharing system for opportunistic networks on a HP iPAQ 910 Business Messenger.

The mobile file sharing system works as mentioned. For every file, the file is split into number of pieces based on the size of an individual piece and then got from the remote device sharing it. If there are multiple remote devices sharing the file, then the file is downloaded from all these remote devices simultaneously. As shown by the results, this increases the overall efficiency of the system and decreases the time taken to download a file. Once all the pieces are downloaded, all individual file pieces are merged together to form one single file. The system was able to support files of sizes up to 40 mega bytes.

We also noticed that as the number of devices from which we are downloading a file increases, the time taken decreases considerably. However, when the number of peers is greater than four, the time taken to download a file does not decrease by a huge margin or in some cases increases. This is because the master node has to connect to all the devices using Time Division Multiple Access (TDMA).

The push mechanism also works as mentioned. If there are files that a user wanted to distribute to other mobile devices, the system would do so when it finds devices in the physical proximity. The push mechanism was also able to support files of sizes up to 40 mega bytes.

The priority based mechanism works as mentioned. The high priority files are downloaded first from a source and then the low priority. For e.g. if image files and text files are marked with high priority and audio and video files are marked with low priority, the image and text files are downloaded first, as they are of small size and download quickly. If we download the audio or video file, it may take long time to download the file and the remote mobile device may go out of range.

The file compression works great with text files, image files like jpg and bmp with compression ratios of 70% – 99%, but with document files such as .doc and .docx, the compression ratios are in the range of 25%. The file compression ratios in case of audio files are not so good, as there is hardly any compression. Hence we would not use file compression for audio and video files.

The system ensures that the amount of data shared from or to a device that is close by is more than the amount of data share to or from a device that is far off, thereby increasing the efficiency of the system and reducing the time taken to share a file and thereby increasing the lifetime of the mobile device.

The system avoids unnecessary requests to remote devices that do not run the application, thereby increasing the efficiency and the lifetime of the mobile device. Also, requests are not sent to devices from which all the files have been successfully received from the last time they connected. Only after a predefined amount of time, the remote device is considered for requests. This not only avoids getting the file names only to know that they are already present on the local device, but also increases the efficiency of the system and the lifetime of the mobile device.

We would like to test the system with many more mobile devices in the vicinity so as to get a clearer picture of the forming of piconets and scatternets and see their effects on the system with respect to efficiency of the system, battery usage and time taken.

We would like to further continue research in the field of file compression because as we witnessed, the time taken to download files reduces by a great margin when files are compressed. This would further increase the efficiency of the system and increase the lifetime of the mobile device.

We would like to implement this system on a notebook or a netbook and do performance analysis of the system by using files of greater sizes in the range of 100’s of mega bytes, such as movie files, installation files, etc. which are not possible on the mobile devices due to limited memory.

REFERENCES

[1] Bluetooth Special Interest Group. Bluetooth specification version 1.1 and 1.2. *http://www.bluetooth.com*, 2001

[2] Max Mühlhäuser and Iryna Gurevych, “Handbook of Research on Ubiquitous Computing Technology for Real Time Enterprises”, Publisher Information Science Reference, Pages 172 – 211, 2008.

[3] Leszek Lilien, Zille H. Kamal, Ajay Gupta, Vijay Bhuse, and Zijiang Yang, WiSe (Wireless Sensornet) Lab, Department of Computer Science, Western Michigan University, Kalamazoo.

[4] The Gnutella Protocol Specification v0.4, Document Revision 1.2.

[5] Matei Ripeanu, “Peer-to-Peer Architecture Case Study: Gnutella Network”, IEEE First International Conference on Peer-to-Peer Computing, 2001.

[6] Andreas Weibel, Lukas Winterhalter, “Bluetella: File Sharing for Bluetooth Enabled Mobile Phones”, Swiss Federal Institute of Technology Zurich, 2005

[7] Jürgen Nützel and Mario Kubek, “Personal File Sharing in a Personal Bluetooth Environment”.

[8] Sewook Jung, Uichin Lee, Alexander Chang, Dae-Ki Cho, and Mario Gerla. “BlueTorrent: Cooperative Content Sharing for Bluetooth Users,” In *PerComm* 2007, WHITE PLAINS, NY, Mar. 2007.

[9] Homepage of Bluetooth SIG Inc. (2003-2005), Retrieved June 14, 2009, from *http://www.bluetooth.org*

[10] Iwatani, Y. (1998). “Love: Japanese style”, Retrieved June 14, 2009, from *http://www.wired.com/culture/lifestyle/news/1998/06/12899*.

[11] Shockfish SA Switzerland (2003). The SpotMe homepage. Retrieved October 15, 2007, from http://www.spotme.ch.

[12] Nokia (2005). “Nokia sensor”. Retrieved October 14, 2007, from http://www.nokia.com/sensor.

[13] Jehn-Ruey Jiang, Bing-Rong Lin, and Yu-Chee Tseng, “Analysis of Bluetooth Device Discovery and Some Speedup Mechanisms”, National Central University, National Chiao-Tung University, Taiwan.

[14] P. Murphy, E.Welsh, and J. P. Frantz. ”Using Bluetooth for Short-Term Ad-Hoc Connections Between Moving Vehicles: A Fesability Study,” IEEE Vehicular Technology Conference, vol. 1, no. 55, Birmingham, AL, May 2002, pp. 414-418.

[15] A. Busboom, I. Herwono, M. Schuba, and G. Zavagli. ”Unambiguous Device Identification and Fast Connection Setup in Bluetooth,” Proc. of the European Wireless 2002, vol. 0, Florence, Italy, Feb 2002.

[16] K. Cheolgi, M. Joongsoo, L. Joonwon. ”A Random Inquiry Procedure using Bluetooth,” Proc. of International Conference on Communication in Computing (CIC), Las Vegas, USA, Jun 2001.

[17] Kaan Dogan, Guray Gurel, A. Kerim Kamci, and Ibrahim Korpeoglu, “Bluetooth Broadcasting Performance: Reliability and Throughput”, Springer Berlin / Heidelberg, Volume 3992/2006, 2006.

[18] Jürgen Nützel and Mario Kubek, “A Mobile Peer-To-Peer Application for Distributed Recommendation and Re-sale of Music”, 13 Dec. 2006.

[19] Johan Ho, “Bluetooth broadcasting”, Department of Computer and Information Science, Norwegian University of Science and Technology, June 2006.

[20] Jian Liang, Rakesh Kuma, Keith W. Ross, “Understanding KaZaA,” Polytechnic University Brooklyn, NY 11201.

[21] F. Siegemund and M. Rohs. ”Rendezvous Layer Protocols for Bluetooth-Enabled Smart Devices,” Proc. 1st International Conference on Architecture of Computing Systems, vol. 2299, Karlsruhe, Germany, pp. 256-273, Apr. 2002.

[22] R. Woodings, D. Joos, T. Clifton, and C. D. Knutson. "Rapid Heterogeneous Connection Establishment: Accelerating Bluetooth Inquiry Using IrDA," Proc. of the Third Annual IEEE Wireless Communications and Networking Conference (WCNC) 2002, vol. 1, Orlando, Florida, Mar. 2002, pp. 342-349.

[23] I. Maric. "Connection Establishment in the Bluetooth System," Master’s Thesis, the State University of New Jersey, 2000.

[24] S. Basagni, R. Bruno, and C. Petrioli. "Device Discovery in Bluetooth Networks: A Scatternet Perspective," Proc. of the Second IFIP-TC6 Networking Conference, Networking 2002, Pisa, Italy, May 2002, pp. 1087-1092.

[25] *http://www.bittorrent.com/*, BitTorrent Inc.

[26] *http://www.napster.com/*, Napster.

[27] *http://www.apple.com/iphone/appstore/*, App store, Apple Inc.

[28] *http://www.android.com/market/*, Android Market, Google Inc.

[29] *https://store.ovi.com/*, Ovi Store, Nokia Corp.

[30] *http://www.msn.com/*, MSN, Microsoft Corp.

[31] *http://www.yahoo.com/*, Yahoo Corp.

BIOGRAPHICAL INFORMATION

Gautam Ravenrda Chavan was born March 1983, India. He received his Bachelor of Engineering in Computer Science and Engineering from Bangalore University, India in June 2005.

Prior to pursuing his Masters he worked at Siemens Communication Software which later became Nokia Siemens Networks, Bangalore, India in the field of Telecommunications for a period of two years. In fall of 2007, he started his graduate studies in computer science. He received his Masters in Computer Science from the University of Texas at Arlington, in May 2009. His research interests include computer networks and wireless sensor networks.