TK
5105.525
H374
2008

**BUILDING A SECURE AND LEGAL FILE SHARING SYSTEM
USING JXTA PLATFORM**

By

Khalid Abdel Hafeez

B.Sc in Electrical Engineering, Jordan, 1988

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Applied Science

in the Program of

Computer Networks

Toronto, Ontario, Canada, 2008

©Khalid Abdel Hafeez, 2008

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis or dissertation to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis or dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Acknowledgements

I would like to express my gratitude towards Professor Bobby Ngok-Wah Ma for his guidance and insight which helped immensely in refining my thesis.

I would like to thank also my wife and my kids for their patience, support and encouragement.

Table of contents:

Abstract	vii
1. Introduction and Background	.01
1.1. Client Server Model	.02
1.2. P2P model	.03
1.3. P2P Applications.	.04
1.4. Problem description	.05
1.5. Requirements	.07
1.6. Thesis Organization	.08
2. P2P Applications	.09
2.1. Napster	.09
2.2. Guntella	.10
2.3. Freenet	.12
2.4. Bittorrent	.14
2.5. Chord	.16
3. Implementation Technologies	.19
3.1. JXTA	.19
3.2. CMS (Content Manager Service)	.45
4. Protocol Design and Implementation	.49
4.1. Overview	.49
4.2. Connection Establishment Mechanism	.56
4.3. Security	.58
4.3.1. How the application addresses the main security requirements	.62
4.3.2. How the application deals with Peer-to-Peer threats	.63
4.4. Certificates and Key Management	.68
4.5. Network Impacts	.69
4.6. The structure of the Peer Implementation	.70
4.7. The structure of the Server Implementation	.77
4.8. The User Interface	.82
5. Security Study of the Implementation	.92
5.1. Comparison between P2P and CS	.92
5.1.1. Simulation parameters	.92
5.1.2. Simulation results	.95
5.2. Further advantages of P2P system	.100
6. Future improvements	.101
7. Conclusion and Contributions	.103
8. References	.104
9. Appendices	.107
10. Glossary	.115

List of Figures

Figure1.2.1: Client-Server Model	02
Figure1.3.1: Peer-to-Peer Model	03
Figure 1.5.1 The Implemented Protocol	05
Figure 2.2.1: Gnutella Search request processing.	11
Figure 2.3.1: How Freenet protocol works.	13
Figure 2.4.1: Bittorrent and HTTP downloading speeds	15
Figure 2.5.1(Chord ring) Identifier Circle of $m=5$, 5-nodes and 6-keys	16
Figure 2.5.2: path taken from Node N_9 for key K_{29}	17
Figure 2.5.3: Chord ring with finger table	18
Figure 2.2.4: Path taken by a query from node 9 for key 31.	18
Figure 3.1.1: JXTA Virtual Network	20
Figure 3.1.2 JXTA peer-to-peer architecture.	21
Figure3.1.3: JXTA ID Mapping	22
Figure 3.1.4: JXTA Network	24
Figure 3.1.5: JXTA Pipe Types	29
Figure 3.1.6: JXTA specification protocol hierarchy	31
Figure 3.1.7: JXTA Protocol Stack	31
Figure 3.1.8: How Peer Discovery Protocol works	32
Figure 3.1.9: How Peer Information Protocol works	33
Figure 3.1.10: How Pipe Pinding Protocol works	34
Figure 3.1.11: How Endpoint Routing Protocol works.	36
Figure 3.1.12: How JXTA peer advertises an advertisement	38
Figure 3.1.13 shows the JXTA three types of discoveries	40
Figure 3.1.14: How JXTA peer discovers an advertisement	41
Figure 3.1.15 JXTA pee behind Firewall	42
Figure 3.1.16: Request Propagation to peer behind firewall	42
Figure 3.1.17: Request Propagation via Rendezvous Peer	43
Figure 3.1.18 JXTA search request propagation	44
Figure 3.2.1 CMS Message exchange	48
Figure 4.1.1a: The Implementation main group	50
Figure 4.1.1b: The initiation of the client peer application	50
Figure 4.1.2: The implemented file sharing protocol	52
Figure 4.1.3: Peer-Peer interaction	55
Figure 4.2.1: Connection Establishment Mechanism	57
Figure 4.3.1: State diagram for verifying the received order	59
Figure 4.3.2 Message encryption	60
Figure 4.3.3: CTR (Stream Mode Encryption/Decryption)	61
Figure 4.3.1.1: SHA256withRSA signature process	62
Figure 4.3.1.2: Man-in-the middle attack	64
Figure 4.3.2.1: The DoS testing network	66
Figure 4.6.1 Peer Application block diagram	70
Figure 4.7.1 Server Bock diagram	77
Figure 4.7.2 Tokens Credentials	80
Figure 4.8.1: Introductory window of peer application	82
Figure 4.8.2: Main window of peer application	83
Figure 4.8.3: Send message window of peer application	84
Figure 4.8.4: Received message window of peer application	84

Figure 4.8.5: The download order	86
Figure 4.8.6: Confirmation window	86
Figure 4.8.7: The upload order	87
Figure 4.8.8: The file save window	87
Figure 4.8.9: The feedback window	88
Figure 4.8.10: The payment	88
Figure 4.8.11: The receipt	89
Figure 4.8.12: The peer information window	89
Figure 4.8.13: The file sharing window	90
Figure 4.8.14: The file searching window	90
Figure 4.8.15: The files searching result window	90
Figure 4.8.16: The renew tokens account window	91
Figure 5.1.1.1: Processing download order mechanism	93
Figure 5.1.1.2: Success ratio vs fake traffic (song file)	96
Figure 5.1.1.3: Success ratio vs fake traffic (movie file)	96
Figure 5.1.1.4: Success ratio vs fake traffic for P2P with different buffers	97
Figure 5.1.1.5: Success ratio vs fake traffic for CS with different buffers	98
Figure 5.1.1.6: Success ratio vs fake traffic for P2P under low-level attack	99
Figure 5.1.1.7: Success ratio vs fake traffic for CS under low-level attack	99

Those people log to the internet using their personal computers and use the internet applications and services like World Wide Web and FTP which are based on client-server architecture. In this architecture all signaling communication between any two peers has to travel to the server. In peer-to-peer networks, on the other hand, there are no points of failure and each peer has an equivalent capabilities and responsibilities. Peer-to-peer networks are usually simpler than client-server networks and rely mainly on the computing power of end systems.

In this thesis, we proposed a hybrid architecture that combines both the client-server and peer-to-peer architectures where there is a server peer to legitimize every transaction between two peers.

In the rest of this chapter we will give a brief description of both Client-Server Model and Peer-to-Peer model. We will then introduce some popular peer-to-peer file sharing protocols like Napster, Gnutella and Freenet.

Chapter-1: Introduction and Background:

This thesis deals with the design and implementation of building a secure and legal file sharing Peer-to-Peer system using JXTA platform [1]. Many peer-to-peer applications like Napster [3][5] and Gnutella [3][6] have been closed and sued for their violation to others' copyrights. Because respecting copyrights and intellectual property is the key to the success of any commercial file sharing application, we build our system to satisfy the need of peers to transfer files easily among them and to assure providers that their products will not be shared without respecting their property.

Today more than one billion people are using the internet which can be defined as a network of millions of networks; each network consists of many computers. Those people log to the internet using their personal computers and use the internet applications and services like World Wide Web and FTP which are based on client-server architecture. In this architecture all signaling communication between any two peers has to travel to the server. In peer-to-peer networks, on the other hand, there are no points of failure and each peer has an equivalent capabilities and responsibilities. Peer-to-peer networks are usually simpler than client-server networks and rely mainly on the computing power of end systems.

In this thesis, we proposed a hybrid architecture that combines both the client-server and peer-to-peer architectures where there is a server peer to legitimize every transaction between two peers.

In the rest of this chapter we will give a brief description of both Client-Server Model and Peer-to-Peer model. We will then introduce some popular peer-to-peer file sharing protocols like Napster, Gnutella and Freenet.

P2P networks can be categorized based on:

- 1- What they are used for (like file sharing, media streaming, telephony, chat, collaboration and white boarding, games, distributed computing,)
- 2- Their degree of centralization (centralized, decentralized or hybrid).

1.1 Client Server Model:

Figure 1.2.1: Client-Server Model

Client-Server model as shown in Figure 1.2.1 has been the most dominant model in computer networking for a long time. In this model, a finite number of servers offering services to many clients. The clients rely on the server to access the services like files, devices and even processing power. Because it is centralized, the network management, maintenance and security are not hard to implement. The servers will have more control over resources and allow only authorized users to access them.

The client server model has some disadvantages such as traffic congestion when many more users trying to access one server at the same time. Another issue is when a server is down; the clients will have no means to access the service. This problem is known as a single point of failure. One of the solutions is to have many servers offering the same services.

1.2 Peer-to-Peer model (P2P):

Figure1.3.1: Peer-to-Peer Model

Peer-to-Peer (P2P) concept [27] was crucial in the Internet development where every computer on the network is of equal rights and responsibilities to the others. In pure P2P systems there is no differentiation between a server and a client as shown in Figure 1.3.1. A peer may be a client of other peers and at the same time a server for others. Peers connected to each other over the internet are using either TCP or HTTP protocols. TCP/IP is the main protocol to transfer information over the Internet while HTTP is built on top of TCP/IP to allow peers to communicate over port 80 since most organizations keep that port unblocked on firewalls to allow web browser traffic.

What does Peer-to-Peer network mean; some people like Li Gong of Sun Microsystems [16][4] refer it to the wide range of technologies that greatly increase the utilization of information, bandwidth and computing resources on the Internet where there is no centralized point. While others like Ed Dumbill of XML.com [4] defines it as "P2P is whoever says they are P2P".

P2P networks can be categorized based on:

- 1- What they are used for (like file sharing, media streaming, telephony, chat, collaboration and white boarding, games, distributed computing,).
- 2- Their degree of centralization (centralized, decentralized or hybrid).

In centralized topology a super peer is used to help other peers in the network to locate each other (Napster) [5]. The role of the super peer or the server is crucial and without it the peers don't know about each other. While in decentralized topology there is no such server and peers need to use some decentralized methods to locate others (Gnutella and Freenet) [6] [8]. In hybrid topology the super peers (peers that have more responsibilities than the other peers) are utilized but the peers can still communicate directly (JXTA, Green Tea and Shwup) [1] [2], they can find each other and use the advertised services without the help of the super peer. The roll of the super peer here is limited. For example, in our implementation the server has only an accounting and monitoring role.

Some advantages of P2P systems:

- Eliminating the need of a centralized server which may cause a single point of failure.
- Distributing data storage among peers, in this case a peer can be either a generic storage space or a container for a list of items to be sought by other peers especially in search operations.
- Reducing the bandwidth requirement. When we have many peers downloading from a server at the same time, that server has to have a large amount of bandwidth to support the simultaneous downloads, while in P2P systems, if one peer downloads that information from another peer it becomes a server for others to download from.
- Speeding the process time. Instead of using a single machine to accomplish a certain task, the task can be distributed over many peers; this may reduce the processing time.
- Increasing the robustness and the availability of the resources. P2P systems may have the same resources or services replicated over many peers to make them available all the time.

Although P2P systems have many advantages, they have also many problems that have to be addressed:

- Security: since there is no centralized monitory it is difficult to prevent malicious peers trying to deceive other peers for their benefit. Access control with proper authentication and authorization among peers is crucial. Another issue in security that should be addressed is the integrity of data transferred between peers.
- Bandwidth: since P2P applications represent both the client and the server at the same time so they need inbound bandwidth as much as outbound bandwidth (symmetrical bandwidth). This could be a problem for some applications where the inbound activities are less than the outbound activities like web servers and also a problem for ADSL network connections where the inbound and outbound traffic is different. Another issue related to the bandwidth is that P2P systems may use broadcasting to locate services and this consumes much more bandwidth than traditional systems and could be vulnerable to denial of service attacks.
- Intellectual Property: in P2P systems it is very easy for users to transfer copyrighted materials to others all over the world. This is a big challenge for P2P applications to prevent such a theft.
- In P2P systems it is very hard to control what peers can do on the network. They may propose invalid or infected data by viruses to others or may they leave the network without any notice.

Figure 1.5.1: The Implemented Protocol

Figure 1.5.1 summarizes the protocol implemented in our solution as follows. Peer A will send a request to download an advertised file from peer B which will respond by an upload order to peer A if that file is still advertised for downloading. All communications are conducted in secure fashion.

1.3 P2P applications:

File sharing is the most common application used over P2P networks. The peer can share some of its files and at the same time download other files from remote peers. Because there is no control over what the peer can download, this type of applications represents a high percentage of the overall internet traffic [29][30]. In the early versions of Napster [5] which was developed in 1999, peers can download MP3 music files free of charge and that's why it was sued and shutdown.

Instant Messengers is another P2P application where peers can exchange short messages between them like AOL and MSN Messenger and others. These Messengers nowadays integrate voice chatting with them so users can talk to each other free of charge. Chat applications are also considered P2P because peers are directly interacting with each other.

Gamming is one of the attractive areas for P2P technology like chess tournament where P2P application is used to exchange game moves among the two opponents and any number of observers.

Collaboration and White boarding applications are P2P in nature where ideas and resources can be shared through interactive communications without reliance on a central server.

Last but not least is the distributed computing where the system can take the benefit from idle processing power of peers' computers to accomplish sophisticated tasks.

1.4 Problem description:

Although file sharing is a legal technology with legal uses, many users use it to download and upload copyrighted materials without permission. This illegal download has become more common because of the availability of high-speed Internet connections, and the high-quality of MP3 audio format. This increase in illegal downloading costs the music and other copyrighted material industries a lot of money and causes many businesses to close.

In our system we tried to come up with a solution where the rights of the producers are saved and at the same time giving the downloader the opportunity to sell and advertise that product to gain some credit. The more sales the user can make, the more credit he can collect. The user can use his credit to download more new files.

By this solution we increase the distribution of the material for the benefit of the producer and the user at the same time and all in legal fashion.

Figure 1.5.1: The Implemented Protocol

Figure 1.5.1 summarizes the protocol implemented in our solution as follows: Peer A will send a request to download an advertised file from peer B which will respond by an upload order to peer A if that file is still advertised for downloading. All communications are conducted in secure fashion.

After the connection between the two peers is established as will be explained in chapter 4, peer-B will start uploading the file to peer-A. After finish downloading, peer A will send a payment signed by its private key to the provider peer (peer B) who will send this payment to the server peer in order to receive the credit.

Peer A will include in the payment an optional feedback about the downloading process. These feedbacks will be used later as a satisfaction measure for every peer and to clean malicious peers from the network.

After receiving and verifying the payment, peer B will send Peer-A a receipt, those receipts will be used by the server for accounting purposes. Every payment should have a corresponding receipt so the peer could not claim the payment more than once.

The server peer after receiving the payments will credit some incentive to the uploading peer (peer B) to courage him to advertise more files on the network and deduct the whole tokens from the downloading peer (peer A) and then send both of them new token credentials. The tokens are used as a virtual currency.

During the interaction between peers, the server does not get involved. Peers can download files and collect payments/receipts and when a timer expires (may be after one month); the peers can contact the server to send him all collected payments and receipts. Because of this, we make the interaction between peers and the server peer as minimal as possible to reduce the consumption of bandwidth.

This is a brief description of what our solution is and in the later chapters we will explain the protocol in more details.

1.5 Requirements:

The implementation satisfies the following specifications:

- 1- Any qualified user can register and use the application and the application should be user-friendly.
- 2- The user can share any kind of files and he or she is the one who controls his shared files.
- 3- All users can find each other and all shared files.
- 4- The user can search for a file and download it or download many files at the same time.
- 5- The solution must have the capability to work through NAT and Firewalls.
- 6- All communications between peers should be in a secure fashion and the file transfer should be encrypted.
- 7- The code should be documented.

1.6 Thesis Organization:

The rest of the thesis is organized as follows:

Chapter 2: Here we will give a brief description about some technologies related to peer-to-peer like Napster, Gnutella, Freenet, BitTorrent protocol and Chord protocol.

Chapter 3: In this chapter we will talk in more detail about JXTA and CMS (Content Manager Service), which are the main technologies used in our implementation.

Chapter 4: Here we will talk about our implementation and the main functions in the implementation. Also we will show how the implementation works and explain in more details about the protocol used to solve the problems stated in chapter 1.

Chapter 5: In this chapter we will analyze the implementation and show how this implementation addresses the main security concerns in the peer-to-peer network settings.

Chapter 6: Here we list some future improvements to the system and how they will be implemented.

Chapter 7: This thesis will be concluded.

Chapter-2: Peer-to-Peer applications:

In this chapter we will give a brief description of some P2P applications such as Napster, Gnutella, Freenet, BitTorrent and Chord.

2.1 Napster:

Napster [3][5] is a program designed by Shawn Fanning in 1999 at Northeastern University. It is designed to allow the sharing of MP3 music files created from CDs or other sources and it is based on centralized P2P architecture. The Napster server role is to hold a searchable index of all files shared by all peers on the network.

When a peer starts, first it connects to the server by a username and a password and has the option to let the client program to search for MP3 files in its machine and send only the names of them to the server for sharing.

The peer can send a search request to the server looking for certain file, the server then send him back a list of all files found and the connection information to the peers who have those files including username, IP address, port to connect to, the connection speed and the file size.

The peer will make a direct connection to the remote peer and start the downloading process without the involvement of the server.

The system works very well and any peer who has a fast connection will see a line up from others to download from him. The system will serve those downloading requests in a first in first served fashion.

The server is a single point of failure and without it, peers can't locate each other and P2P communication is no longer possible.

In Dec 1999, Napster was sued by the music industry for copyright infringements.

2.2 Gnutella:

Gnutella [3][6] was designed by AOL's Nullsoft division as an alternative to Napster, but it was quickly canceled because AOL does not want to allow the illegal exchange of copyrighted materials. But it continues to grow because some developers outside AOL have the source code of this program.

The main difference between Napster and Gnutella is that Gnutella uses HTTP protocol to transfer information. The Gnutella has a decentralized topology and there is no server. A Gnutella peer who wants to share files looks like a web server with links to all shared files and the peer who wants to download logs first to the server peer the same way as the web server.

When a peer wants to search for a file, the peer will send a search packet with a maximum hop value to indicate the maximum number of times the packet will be forwarded on the Gnutella network. The peer will send this packet to all known peers. When a peer receives the search packet, this peer will send what files matches the search to the requesting peer and forward the search packet to all peers he or she knows about.

This search process has disadvantages;

- First the requesting peer can never be sure that the search request will reach the peer who has the information.
- Second, the search process will consume a lot of bandwidth by broadcasting the search packet to all known peers.
- Third, if the peer wants to increase the max hop value, the answer may take long time to reach him.
- And finally, the results will be sent back to the requester the same sequence it is initially traveled.

Figure 2.2.1 shows the search request passed between three peers:

- 1- Peer A initiates the search request for a file to all known peers with a maximum hop count.

- 2- Peer B receives the request and record it in a database for later use and checks if it has the requested file, if yes it will send it to Peer A.
- 3- At the same time Peer B will decrement the hop counter and forward the request to all known peers except the one who send that request.
- 4- Peer B will receive the results from downstream peers.
- 5- Peer B will look up for the source of the request in the database.
- 6- Peer B will forward the results to the requesting peer.

Figure 2.2.1: Gnutella Search request processing

2.3 Freenet:

Freenet [4][8] is a decentralized peer-to-peer protocol designed by Ian Clarke to provide freedom of speech with strong protection of anonymity. It can be considered as a large storage space that distributed over all connected peers. It implements a protocol similar to the Distributed Hash Table (DHT) to locate peers and data called Key Based Routing.

In Freenet file sharing system, the documents are encrypted and replicated across the participating peers all over the network so it is difficult for any peer to find out who are hosting a given file (anonymity). The stored documents are then allowed to be retrieved later by an associated key. The Freenet network is composed of many peers came together and send messages among themselves, some peers are end users from whom the documents are requested and others are only for routing the data. A Freenet peer knows only about his neighbors whom he sends the messages to, and does not care if that neighbor forwards those messages to other peers or not (to ensure anonymity). The message will be routed from neighbor to neighbor until it reaches the final destination or the max time-to-live expires. In Freenet each peer maintains a data store containing documents associated with keys, and a routing table associating nodes with records of their performance in retrieving different keys.

The protocol works as follows:

When a peer wants to find a document with a key, he will send a request to one of his neighbors who is most likely has the document or knows from where to get it.

If that neighbor does not have that document, he will decrement the Hops-To-Live counter and forward the request to his most likely peer that can find it and keep a record of that request. If that peer does not have any neighbors or the request bounce back to him again, he will send the originator a failed message to let him choose the second most likely peer. And so on until the document is found or the Hops-To-Live counter expires.

If the document is found, it will be sent back to the originator the same sequence the request took, the intermediate peers may choose to cache the document on the way to make it very hard to censor as there is no one source node.

In Figure 2.3.1: peer A sends a request to peer B, peer B will forward the request to peer C who replied with a Request Failed message indicating that he does not have the data and he has no other neighbors, peer B will send the request to the next possible peer, peer D. Peer D forwards the request to peer E, peer E forwards it to his only neighbor peer B, but peer B has this request before so he will reply by a Request failed message to peer E who will reply with a Request failed message to peer D, then peer D will choose the next peer in the list (peer F) and forward the request to him. Peer F has the data and so he will reply with a Data reply message to peer D who will forward it to peer B and then to the requester peer A.

Figure 2.3.1: How Freenet protocol works.

Freenet does not use broadcasting to search for a document, and this saves a lot of bandwidth, but some users criticize this approach claiming that it is hard for regular users to use and is not reliable. Because Freenet is originally designed more for secrecy than for usability so it is difficult to use compared to some other open networks like Gnutella.

2.4 BitTorrent Protocol:

BitTorrent [10][11] is a file sharing communication protocol design by Bram Cohen in 2001, it is a P2P protocol but still has a central server called a tracker to coordinate the actions of all peers who come together to send and receive portions of a file.

For example if "S" is the peer who has all portions of a file and "A", "B", "C" and "D" are the peers who want to download that file. In BitTorrent, they don't have to download it completely, but each one of the peers will download quarter of the original file, so after one cycle peers "A", "B", "C" and "D" have different parts of that file, then they can download it from each other saving the bandwidth of the original peer "S" and give it the opportunity to serve other set of peers. The main peer "S" called the seed.

If a peer wants to share a file, first it should create a small file called "torrent" which contains information about the file and the tracker. This torrent file can be published in a website or some other place and be registered with a tracker. Peers who want to download the file; first they should obtain the respective torrent file, and then connect the specified tracker who will tell them from which other peers to download all pieces of the file.

In case of a decentralized environment, each peer wants to share a file becomes the tracker of that file. This approach is implemented by BitTorrent using the Distributed Hash Table (DHT) method.

BitTorrent differs from the classic HTTP full file transfer by that BitTorrent makes many requests over different TCP sockets and download the file in a random or rarest-first fashion to ensure high availability, while HTTP makes a single request over one TCP socket and downloads the file sequentially.

These differences make BitTorrent more redundant, more resistance to abuse and a lot cheaper than HTTP. But the disadvantage is that in BitTorrent the download will gradually rise to the high speed then slowly fall back down towards the end of the download. In HTTP the download speed

will rise to maximum very quickly and remains there to the end of the download as shown in Figure 2.4.1.

Figure 2.4.1: Bittorrent and HTTP downloading speeds

How to share a file in BitTorrent? The peer will divide the shared file in equally sized portions (64K – 1M), and creates a checksum for each piece using a hashing algorithm such as SHA-1 and save it in the torrent file, this checksum will also be used later to make sure the downloaded piece is error free.

BitTorrent is best suited for newer files or files with high demand rather than older or lower demand ones, because there will be many users interested in downloading the new file, so a new user will find many peers have many parts of that file on the network. On the other hand if only one interested peer is found, then the seed will have to upload the file completely to the interested peer making the protocol the same as the traditional client-server system.

Another problem in BitTorrent is that it does not offer a search mechanism to find a file by name; instead the peer has to know from the beginning from where to get the torrent file. Also this protocol does not offer any incentive to peers that just finished downloading a file to become an up loader for that file.

2.5 Chord:

Chord [9] is a distributed lookup protocol that addresses the problem of finding a node that has a desired data in P2P system where nodes arrive and departure without prior notice.

It assigns keys to nodes with consistent hashing, all nodes receive almost the same number of keys, in N -node system when the N^{th} node joins or leaves the system, only $O(\frac{1}{N})$ fraction of the keys need to be relocated.

Each node requires to maintain information about only $O(\log N)$ of other nodes, and a lookup requires $O(\log N)$ messages.

The consistent hashing assigns an m -bit identifier to each node and key in the system using SHA-1, the $(m \text{ bits})$ should be long enough to ensure a collision free system.

In consistent hashing the identifiers are ordered on an identifier circle modulo 2^m , so key K is assigned to the node whose identifier is equal or follows K and called the $successor(K)$ which is the first node clockwise from K .

Figure 2.5.1(Chord ring) Identifier Circle of $m=5$, 5-nodes and storing five keys

In Figure 2.5.1 if the node with an identifier of 28 joins the system, then K26 will be relocated from N30 to the new node N28. And if node N9 leaves the system, only K8 will be assigned to N20.

So in a system of N -nodes and K -keys, each node is responsible for a maximum of $\frac{K}{N}$ keys and when a node joins or leaves, only $O(\frac{K}{N})$ keys are relocated to the joining node or to the successor of the leaving one.

For the same scenario, if node $N9$ wants to search for $K29$, in simple key lookup, node $N9$ needs only to know its successor $N20$ to send it the query for $K29$ and then $N20$ will send the query to its successor $N23$ and then to $N30$. The result will return the reverse path followed by the query as shown in Figure 2.5.2.

Figure 2.5.2: path taken from Node $N9$ for key $K29$

To accelerate this method, Chord uses what is called the finger table which contains at maximum m -entries and the i^{th} entry for node n is the successor of $(n+2^{i-1}) \bmod 2^m$ where $1 \leq i \leq m$. And for every entry in the finger table, it will include the Chord identifier and the IP address (and the port number) as shown in Figure 2.5.3.

Figure 2.5.3 (Chord ring) Identifier Circle of $m=5$, 6-nodes and storing six keys

Figure 2.5.4 shows how the lookup query issued by node $N9$ for key $K31$. First $N9$ will check its finger table to see that the largest finger that precedes $K31$ in its table is node $N27$, so $N9$ will ask $N27$ to resolve the query. In turn, $N27$ will determine the largest finger that precedes $K31$ in its table is $N30$, so it will forward the query to $N30$. Finally $N30$ determines that its finger $N1$ is the successor of $K31$ so it will forward it to the requester $N9$.

From the previous example and because each node has finger entries that are apart from each other at 2^{i-1} intervals, each node can forward the query at least halfway of the remaining distance to the target. So at maximum the target can be found after contacting an $O(\log N)$ nodes.

Figure 2.2.4: Path taken by a query from node 9 for key 31.

Chapter-3: Implementation Technologies:

In our implementation we used JXTA platform to build our peer-to-peer network and the Content Manager Service (CMS)[2] as the basis for file sharing. In this chapter we will give a brief description of both technologies.

3.1 JXTA

JXTA (short hand for juxtapose) [1][2][3][4][5] is a peer-to-peer platform developed by Sun Microsystems in 2001 to be juxtapose for client-server computing model. This platform is not created based on one software language although since it was begun by Sun [2], JAVA was the first implementation language for JXTA. The goals of JXTA is to support peer-to-peer programming on any device from PC to PDA or any electronic device with a heart beat.

The JXTA platform is composed of six protocols that can be implemented in any software language. The JXTA network is composed of many nodes (peers) interconnected with each other; every peer may be connected using different networking protocol like TCP/IP, HTTP, GSM or others. Each peer can provide a set of services to others by advertising them and must run the services that offered by the group that he or she wants to participate in. The advertisement of a service is done by an XML document (called advertisement) to allow the peer discovers and determines how to use that service.

Peers in JXTA can form peer groups, each peer group has its own set of services and purposes like chat group, file sharing group or others. Peers in JXTA network can use either sockets or pipes to send messages between each other. JXTA sockets are reliable and bi-directional connections while pipes are asynchronous and unidirectional message transfer mechanism that are bound to specific endpoints like TCP ports and IP addresses. Messages are simple XML documents that contain routing, digest and credential information. Figure 3.1.1 shows how JXTA maps the physical network to JXTA virtual network where peers don't see the underlying network but using the high level JXTA pipes or sockets to communicate with each other.

Figure 3.1.1: JXTA Virtual Network

Some of the basic JXTA goals are:

- Peers should communicate, discover and monitor each other and form peer groups among them.
- Peers should advertise and discover network resources.
- The platform should be language and network transport and topology independent.
- Peer groups use authentication and credentials to control access at the group level.
- The platform should provide an infrastructure for routing and communication between peers, especially those who are behind firewalls.
- The platform should be easy to use and stable.
- The platform should support different communication protocols.

Figure 3.1.2 JXTA peer-to-peer architecture

JXTA is made up of three different layers as shown in Figure 3.1.2:

- a. Platform layer (Core layer):** this is the layer where the code of implementing the JXTA protocols is found. Those protocols provide the functionality to peers, peer groups, security and monitoring. Above the protocols there is the World Peer Group which is the universal peer group the peer will join when it starts JXTA application. This main group gives the peer the main functions he needs like discovery, join, exchange messages and create new groups.
- b. Services layer:** is the layer which built on top of the core layer to accomplish certain task. Some of these services are essential like membership services and others are for added conveniences like storage systems, file sharing, indexing and others.
- c. Applications layer:** in this layer the developer build his or her code to let the peers come together to perform certain function like file sharing, entertainment and distributed computing.

The line between the services and application layers is not rigid, one may see others application as a service that can benefit from in his or her application, while others may see it as a complete application that can be used.

JXTA main technologies and concepts:

1- IDs:

In P2P systems, many peers, peer groups, files or other services may have the same name which leads to a problem in defining each of them, JXTA solves this problem by giving each service an ID also known as URN (Uniform Resource Name) as shown in Figure 3.1.3 which is a unique string to identify one of the following:

- **Peers:** urn:jxta:uuid59616261646162614A78746150325033F3BC76FF13C2414CBC0AB663666DA53903.
- **Peer Groups:** urn:jxta:uuid-A19999539B18489C8E47860E6C89549E02.
- **Pipes:** urn:jxta:uuid59616261646162614E504720503250338E3E786229EA460DAC1A176B69B731504.
- **Content:** md5:2b9cbd6ab82c8fee8fe2a2b9e7eab7a85.
- **Module Classes:** urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000305
- **Module Specifications:** urn:jxta:uuidDEADBEEFDEAFBABAFEEDBABE00000001994191D8846242F8AF4FBB14AF2CDFAF06.

Figure3.1.3: JXTA ID Mapping

2- Peers:

A peer can be defined as an application executing on any device with a heart beat and uniquely identified by a peer ID and implements the core JXTA protocols. Each peer can function as a client to request information from others or as a server to answer requests from other peers or may be both at the same time. Peers are designed to spontaneously discover each other and form peer groups.

Peers can advertise one or more network addresses called peer endpoints which are used for direct point-to-point connections between peers. Those connections are not always available, but an intermediary peer may be used to route messages to others separated by boundaries like NAT, firewalls and proxies.

JXTA peer types:

- i) Minimal-Edge peers: peers that implement only the core JXTA services and depend on others to be their proxy for other services like caching, searching and discovery to fully participate in JXTA network. Sensor and home automation devices are typical Minimal-Edge peers.
- ii) Full-Edge peers: they are the majority peers in the JXTA network because they implement all of the core and standard JXTA services to participate in all JXTA protocols. PC's, servers and phones are examples of Full-Edge peers.
- iii) Super-Peers: peers that implement and provision resources to support the deployment and operation of JXTA network, each of them may implement one or more of the following functions:
 - Relay: to store and forward messages between peers that have no direct connection because of firewalls or NATs.
 - Rendezvous: they usually have stable IP addresses and well-known DNS names. They help edged and proxied peers in their advertisement search because they maintain global advertisement indexes. They also handle message broadcasting.
 - Proxy: they translate and summarize requests, respond to queries and provide support functionality to Minimal-Edge peers usually separated by firewalls and NATs.

Figure:3.1.4: JXTA Network

3- Peer groups:

Peer group as shown in Figure 3.1.4 is a set of peers come together to perform certain task or interest. Each peer group has a unique peer group ID and a membership policy which could be either open for any peer to join or restricted to who has the right credentials to gain membership. When a peer starts the JXTA it joins first the Network Peer Group and allowed to join one or more other groups at the same time.

Groups form a hierarchical parent-child relationship, each group has a single parent and its advertisement is published in the parent group and in the group itself, while the search requests are propagated within the group itself only.

Each group has the following set of core and standard services and has the option to create and implement additional services:

The core services that each peer must implement to participate in JXTA network:

- Endpoint Service: used to implement the Endpoint routing protocol and to exchange messages between peers.
- Resolver Service: used to send generic query requests to other peers like finding an advertisement or the status of a pipe.

The Standard Services that can be implemented by the JXTA peer:

- Discovery Service: used to search for peers, groups, pipes and other services.
- Membership Service: used by peers to securely establish identities and trust with a peer group. The identities are used to determine who is requesting the operation and if this operation is allowed or not.
- Access Service: is used to check the credentials of the requesting peer and make sure that peer is allowed to access that service.
- Pipe Service: is used to create and manage pipe connections between peers within the peer group.
- Monitoring Service: is used to allow a peer to monitor others in the same peer group.

Reasons for creating peer groups:

- Creating a secure environment: where a specific security policy can be enforced and it may be simple like username/password or public key cryptography.
- Creating specialized environment: like file sharing or chatting groups.
- Creating monitoring environment where peer groups allow peers to monitor others for any purpose like (heartbeat, traffic, etc).

4- Network Service:

There are two types of network services in JXTA network:

- Peer Services: if a peer published a service like content or pipe, that service is only available on that peer, if that peer fails, the service will

not be available. The content or the file is published using the peer group services.

- Peer Group Services: peer group service is a collection of multiple instances of the service running on many peers within the peer group; if one peer fails the service will still be available if there is at least one member in the group. Discovery, pipe and rendezvous services are examples of peer group services.

5- Modules:

A module is a piece of functionality designed to be downloaded or obtained outside the core JXTA implementation. They provide a generic abstraction to allow a peer to instantiate a function or service. A peer group could advertise a specification that has certain functionality within the JXTA network and if a peer wants to use that functionality he should discover that specification. The specification does not provide the functionality but only provide information about it. In JXTA system, many peers may have different implementations in different languages for the same specification but still provide the same service for redundancy.

The module abstraction has a module class, module specification and module implementation, each of them has its own advertisement that could be published and discovered by other peers:

- Module Class: is mainly used to advertise the existence of the behavior, it has a unique ID called ModuleClassID.
- Module Specification: it is used mainly to access the module since it contains all the information to do that. For example it may contain the pipe to be used for accessing that service. There can be many Module Specifications for a given module class, each of which has its unique address called ModuleSpecID that contains the ModuleClassID in it to indicate the associated module class.
- Module Implementation: it is the implementation of a give module specification class. It could be many implementations for the same specification and each contains the ModuleSpecID that implements.

6- Messages:

JXTA messages are the basic units of communication between peers, they are sent between them using one of the services available in the JXTA network like Endpoint Service, Pipe Service, JxtaSocket and JxtaBiDiPipes.

The message is in XML format to allow different peers to utilize a given protocol the way it suits them. Because the data is tagged, it is easy for any peer to take part of the message that peer needs and leave other parts if they did not fit its capability.

There are two on-wire message representations (XML and binary) defined by JXTA protocols to get the most benefit from the underlying transport protocol. JXSE uses the binary format because it can be compressed and encrypted.

7- Advertisements:

An advertisement is an XML document (encoded with UTF-8) that describes resource or content. JXTA protocols use advertisements to pass information between peers. Peers may discover those resources by searching for the corresponding advertisements, when found peers may cash them locally.

Advertisements are published with a lifetime, when expires, the advertisement will be flushed from the system without the need for centralized control. To extend the lifetime of an advertisement, it has to be republished before it expires.

Types of JXTA advertisements:

- Peer Advertisement: it holds specific information about the peer like name, ID, available endpoint and Digital certificate as shown in appendix A.1.
- Peer Group Advertisement: it describes peer group specific resources like name, ID, description and service parameters as shown in appendix A.2.

- Pipe Advertisement: it describes the pipe communication channel by having the unique pipe ID, pipe type (point-to-point, propagate, secure,) as in appendix A.3
- Module Class Advertisement: it formally documents the existence of the module class by having a unique ModuleClassID, name and description.
- Module Specification Advertisement: it contains name, description, unique ModuleSpecID and a pipe advertisement to allow other peers to search for the implementation and use it.
- Module Implementation Advertisement: it includes name, the associated ModuleSpecID and code, package and parameter field to allow the peer to execute that implementation.
- Rendezvous Advertisement: it describes the peer that acts as a rendezvous for that peer group.
- Peer Info Advertisement: it holds some information about a peer like uptime, traffic and last time message is sent or received.

8- Pipes:

Pipes are virtual, asynchronous, unidirectional communication channels used by JXTA peers to exchange messages within the peer group boundaries. Pipes have unique IDs, so the peer can carry its pipe even if its physical network location changes. Peers don't worry about the network topology or where other peers located on the network to send them messages, as long as the pipe is invoked, the message put in one end of the pipe comes out from the other end. Pipes use the concept of endpoints to refer to the input and output points of communication, the channel is referred to the connection between the endpoints.

Pipe endpoints are bound to peer endpoints which are correspond to the peer network interfaces like (TCP port and an IP address) which are used to send and receive messages. The output and input pipes must belong to the same peer group.

Pipes offer the following very low-level modes of communication:

- Point-to-Point Pipe (Unicast pipe): it connects two pipe endpoints together as shown in Figure 3.1.5a.
- Propagate Pipe: connects one output pipe to many input pipes as shown in Figure 3.1.5b.
- Secure Unicast Pipe: it is a point-to-point pipe that provides security by implementing TLS/SSL protocol.

Figure 3.1.5: JXTA Pipe Types

Pipes also offer high level communication modes to provide bidirectional, reliable and secure communication channels:

- JxtaSocket and JxtaServerSocket:
- JxtaBiDiPipe and JxtaServerPipe:

9- Security:

Security is one of the main concerns in P2P environment [19][22] because it is much harder to implement than in a centralized system. Security needs are different from one application to another, but the basic requirements must be satisfied:

- Confidentiality: the contents of a message must be encrypted such that unauthorized peers cannot retrieve the content even if they obtain the message.
- Authentication: to grantee that the sender and the receiver are who they claim to be.
- Authorization: to make sure that the sender and the receiver are authorized to send and receive data.
- Data Integrity: to make sure the message is not modified in transit.

- Refutability: the message will be signed and the sender cannot deny that he or she signed that message.

In JXTA system, the XML messages provides the ability to add credentials, certificates, digests and public keys to enable the basic security requirements.

Message digests and signatures are used to grantee the integrity of data. Messages also can be encrypted and signed for confidentiality and refutability. Credentials are used for authentication and authorization.

Security Attacks in P2P networks:

- Active Network Attacks: here the attacker is an active peer.
 - Masquerades: the attacker pretends some valid or privileged peer.
 - Man-in-the-middle: the attacker intercepts the communication between two peers to modify or corrupt the data flow.
 - Replay attack: the attacker captures the exchange of the information between two peers and repeats the exact steps to look like it is a genuine session.
- Passive Network Attacks: here the attacker is in inert state.
 - Eavesdropping: the attacker captures the data silently.
 - Traffic Analysis: the attacker captures and analysis the data to learn more about it.

JXTA platform provides the following security features:

- TLS (RFC 2246) as a Secure Transport Layer by using the secure pipes which internally use TLS to prevent passive attacks.
- Peer Certificates: each peer generates a self signed certificate called the root certificate to be used with TLS and is distributed to others by its peer advertisement.
- Personal Security Environment: every peer has a peerID and a password that form the first defense against local attackers because it is used to encrypt the peer's private key.

JXTA platform depends on PureTLS, Cryptics 3, Cryptics ASN.1 and Bouncy Castle Crypto APIs for its security requirements.

10-JXTA Protocols:

JXTA defines a set of six protocols which are asynchronous and based on query/response model for communication between peers. Peers use those protocols to discover each other, advertise and discover network resources and for communication and routing messages. Peers are required to implement only the protocols that they will use and not all of them. Figures 3.1.6 and 3.1.7 shows the JXTA protocol stack and how each protocol depends on others.

Figure 3.1.6 JXTA specification protocol hierarchy

Figure 3.1.7 JXTA Protocol Stack

1) Peer Discovery Protocol (PDP):

It is used to advertise the peers own resources or discover other peer's resources like (peer, peer group, pipe or any resource that has an advertisement) within a peer group.

JXTA uses either IP multicast or rendezvous network (based on distributed Hash Table) to discover advertisements. A peer may be programmed with a predefined set of rendezvous nodes.

PDP works either locally (within the requesting peer's own cache) or remotely (across the network) and is implemented by the discovery service from the peer group.

The discovery service should follow these steps:

- a. Receive a local or remote request for discovery from an application.
- b. For local discoveries it searches only the local peer's cache.
- c. For remote discoveries it builds a query message and forwards it to all known peers as shown in Figure 3.1.8.
- d. Places the received responses in the local cache based on their types.

1- Peer A sends a Discovery Query Message to all known peers and its registered rendezvous peers.

2- Rendezvous peer will process the discovery query and may respond by a Discovery Response Message contain advertisements from its local cache. It will also propagate the query to its all known peers.

3- All peers receive the query will search their local cache for matching advertisements. If found it will send a Discovery Response Message contains the found advertisements to the requesting peer directly.

Figure 3.1.8: How PDP works.

2) Peer Information Protocol (PIP):

It provides a set of messages to obtain peer status information. The PIP ping message is sent to a remote peer to check if the peer is a live or to get more information about that peer like (uptime, traffic information and peer advertisement), while the PeerInfo message is used to send a response to the ping message and it contains (the credential of the sender, source peerID, target peerID, uptime and peer advertisement) as shown in Figure 3.1.9.

Figure 3.1.9: How PIP works.

3) Peer Resolver Protocol (PRP):

It allows the peer to send query requests to other peers directly or through rendezvous service within the peer group. Both PDP and PIP are built to use PRP. The PRP is designed to wrap query messages from high-level components and deliver them to the JXTA network.

The resolver query message is used to send a resolver query request to remote peer in the same peer group. This message contains the credentials of the sender, a unique query ID, a specific service handler and the query.

The Resolver Response Message is used to send a message in response to the resolver query message and it contains the credentials of the sender, a unique query ID, a specific service handler and the response.

There is no guarantee that a peer will receive a response to its query.

4) Pipe Binding Protocol (PBP):

It is one of the most used JXTA protocols and is used by peer group members to bind a pipe advertisement to a pipe endpoint. PBP defines a pipe and specifies how this pipe can be used for communication between peers in the same peer group. PBP is a high-level service used to allow peers to exchange messages. PBP uses PRP to obtain a path from one peer to another as shown in Figure 3.1.10. The PRP will use the Rendezvous Protocol to propagate messages to a remote peer or it will use the Endpoint Protocol for direct communications.

PBP query message is sent by a peer pipe endpoint to find a pipe endpoint bound to the same advertisement.

PBP answer message is sent back by each peer bound to the pipe to the requesting peer and it contains pipe ID, peer information and a Boolean value indicating whether the input pipe is available or not.

Figure 3.1.10: Pipe Binding Protocol.

5) Endpoint Routing protocol (ERP):

It enables a peer to send messages to remote peers without having a direct connection to them by passing them to an intermediary peer who will pass them to their destination as shown in Figure 3.1.11.

ERP defines a query and response messages to be used in discovering other peer's routing information.

Route query requests are sent by a peer to request route information for a destination peer.

Route response contains the responder peer ID, the destination peer ID and a semi-ordered sequence of peer IDs. This sequence may contain complete or partial route to the destination and in some cases may contain several route alternatives.

The Message routing procedure used by the Endpoint Routing Protocol is roughly as follows:

- a. The peer originating the Message checks if it has a route to the destination, if not a query for a route will be sent.
- b. For the peer who is not originating the Message and does not have a route for the destination or to any peer listed in the route attached to the message, it then sends a failure message to the peer from which it received the Message. JXTA peers do not currently generate route queries for messages they do not originate as this is too easily used to create distributed denial of service attacks (DDoS).
- c. The peer will remove its peer ID from the message route and all peer IDs which preceded its ID in the Message route. Excluding all Peer IDs already in the Message Route, prepend the route it knows to the destination peer to the Message route.
- d. Starting at the last peer ID listed in the Message Route, the peer will check if he has a direct connection to any of the listed peer IDs. If so, remove all peer IDs before that peer in the Message route and forward the message to directly connected peer.
- e. If no direct connection exists to any of the peer IDs listed in the Message route then forward the message to the first peer listed in the Message Route.

1- Peer A wants to send a message to peer D but it does not have a route to it. So it sends a Rout Query Message to all known peers and its registered rendezvous peer trying to determine a route to peer D.

2- Peer B does not know a route to peer D so it will send a failure message to peer A.

3- The Rendezvous peer knows a route to peer D so it will send a Rout Response message to peer A.

4- Peer A receives the Route Response Message. It adds an Endpoint Route Message to the message it wants to send to peer D and sends it to the first address in the received route message.

5- When peer C receives the Endpoint Route Message it determines the next peer in the route, amends the endpoint route message and sends the message to the next address in the route.

6- When peer D receives the Endpoint Route Message it determines that it is the final destination. The original message is extracted and sent to appropriate service by the Endpoint Service.

Figure 3.1.11

6) Rendezvous Protocol (RVP):

It is used to propagate messages within a peer group. When a peer becomes a rendezvous it creates a rendezvous advertisement to advertise the rendezvous service to other peers in the same group.

RVP is divided into three parts:

- PeerView Protocol which is used by the rendezvous peers to organize themselves by synchronizing their views of each other.
- Simple Lease Protocol which is used by client peers to register the interest in receiving propagation messages.
- Message Propagation Protocol which is used for propagating messages to peers within a group.

The RVP allows rendezvous peers to work together to form a *global peerview* which is an ordered list (by peer ID) of peers currently acting as rendezvous peers within a given group. This list is used to route messages within that group. Each peer maintains a local version of the list and the goal of the PeerView protocol is to keep the individual local peerviews consistent across all rendezvous peers within a group. This consistency is required for optimal query propagation. To achieve this consistency every rendezvous probes others every 30 seconds by default.

11- How JXTA peer publishes a services:

JXTA uses Loosely-Consistent Distributed Hash Tables (LC-DHT) which defines the mechanism for publishing and discovering resources within JXTA network. Each resource has an advertisement and peers maintain and publish attribute tables for their advertisements which consist of tuples (index attribute, value) which is associated to life duration and the publishing peer ID. These attributes are published by edge peers to their associated rendezvous peers. The rendezvous peer when receiving this publication request it will keep a copy of the tuples and then replicate them across the entire network of rendezvous peers by applying SHA-1 hash function on each tuple to determine on which rendezvous to be replicated based on its local peerview.

The rendezvous peer uses the following procedure to determine to who should this advertisement is replicated:

- 1- $Hash = SHA-1(tuple);$
- 2- $pos = floor(hash * \frac{I_i (size-of-peerview)}{MAX_HASH})$
- 3- Return peerview entry at position (pos).

For example if the number of rendezvous peers in a network is ($I_i = 6$), and an edge peer E_1 wants to advertise an index attribute Name and its

value is Test as shown in the Figure 3.1.12, so the hash function will be applied to the string "PeerNameTest" and assume that the output is 116 and the MAX_HASH value is 200, and the local peerview as shown in table-1:

Peerview entry	0	1	2	3	4	5
Peer ID (R_i)	006(R_1)	020(R_2)	036(R_3)	050(R_4)	088(R_5)	180(R_6)

Table 1

Figure 3.1.12: How JXTA peer publishes an advertisement.

To publish the advertisement (Adv) the following two steps will be applied:

- 1- The edge peer E_1 will send the tuple to its registered rendezvous R_1 who will compute the appropriate replica peer for this advertisement by applying:

$$position = \text{floor}(116 * \frac{6}{200}) = 3.48 = 3$$

- 2- Peer R_1 will replicate the tuple (116, E_1) to the third peer in R_1 's peerview which is peer R_4 according to table-1. The same tuple will also be stored in R_1 to increase its availability to all edge peers connected to the same rendezvous.

12-How JXTA peer discovers other peers and services:

Discovery is the process of finding another peer or service in the same peer group. There are three types of discoveries:

- a. Local discoveries: each JXTA peer has an empty local cache of advertisements when the peer first started. This cache will be seeded with time from new discovered advertisements. When a peer initiates a search, it checks its own cache first, if it did not find the requested advertisement, it will initiate a remote discovery. As long as the peer initiates new searches and discovers new advertisements, it will populate its local cache. The information in the local cache may be stale and that's why each of the advertisements has to have a time-to-live, so when it expires, the advertisement will be flushed from the cache.
- b. Direct discoveries: the peer can contact each of the peers in its local network and discovers what services available using broadcast or multicast of TCP/IP. A peer can broadcast a discovery request message with search criteria to other peers in the same network. But for peers outside the local network, the peer should use indirect discovery (propagated discovery). Both the direct and indirect discoveries will occur at the same time if both TCP/IP and HTTP protocols are utilized under JXTA.
- c. Propagated discoveries: if a peer wants to discover other peers outside its local network, it should contact a rendezvous peer as an intermediary for discovery operations. The rendezvous peer caches all peers comes in contact with it over the time. If there are many rendezvous peers in the network, they will forward the discovery to one another and a large discover will take place in a short period of time. Each discovery process has a time-to-live to prevent it from looping on the network forever. In the request there is a list of all peers that have seen it before. The peer will discard the request if it is seen before. This allows the request to exist in the network efficiently.

Figure 3.1.13 shows the three types of discoveries:

Figure 3.1.13: Types of JXTA discoveries

To continue the same example as shown in Figure 3.1.14:

If edge peer E_2 wants to discover the advertisement (Adv), it should specify the query by indicating the index attribute and its value as a discovery protocol message to be propagated over the JXTA network as follows:

1. The message is first forwarded by E_2 to its rendezvous R_2 via the resolver protocol.
2. The rendezvous peer R_2 checks first if it has the hash value of this advertisement ($hash=116$) in its local cache, if yes it will forward the query to the edge peer E_1 and will go to step 5.
3. If the hash value (116) is not found then R_2 will apply the same formula:

$$position = \text{floor}(116 * \frac{6}{200} = 3.48) = 3$$
and forward the query to R_4 as per its peerview as shown in table-1.
4. Peer R_4 is the correct replica peer who will forward the query to peer E_1 .
5. Peer E_1 will send the requested advertisement directly to the requesting peer E_2 .

Figure 3.1.14: How JXTA peer discovers an advertisement using its Rdv peer.

What about if rendezvous peers have different peerviews: this could happen due to churn where peers join and leave the group unexpectedly, so the rendezvous peer who perform the discovery may choose a replica peer different of what is computed by the publishing rendezvous peer. In this case a backup mechanism is used where the calculated replica will forward the query to the upper and lower neighbor rendezvous peers who highly probable may store the requested advertisement. In this case the query is walking the

whole peerview in both directions until it reaches the upper_rdv (R_6) and lower_rdv (R_1) rendezvous peers respectively.

We can see that the LC-DHT has a complexity of $O(1)$ in the best scenario or (two messages at most) where classical DHTs have a complexity of $O(\log n)$, but in the worst case where the calculated replica peer is not the same in the publishing and discovering processes, the complexity is in order of at maximum $O(r)$ where r is the number of rendezvous peers in the JXTA network.

13- How JXTA handles private networks:

Not all peers are connected directly to the internet but many of them are setting on private networks as shown in Figure 3.1.15, so JXTA has to deal with NATs and Firewalls.

Figure 3.1.15: JXTA peer behind the firewall

The job of firewalls is to restrict the traffic coming in or going out the private network. The administrator will lock down all ports that TCP/IP could use for sending data except those for certain services like email or web browsing.

JXTA uses HTTP protocol to send data to peers behind firewalls like web browsing and those peers can respond using a response protocol defined in the HTTP protocol. In this situation the outside peers will not find the internal peers, and to solve this problem, a relay peer has to become as a bridge between the internal and the external peers. When an external peer wants to send a message to an internal peer, first it will contact a relay peer with the message who will hold the message until the internal peer sends a request to the relay peer, and then the relay peer will send the message within a response to the internal peer as shown in Figure 3.1.16. After that the internal peer has the address of the external peer so it can contact the external peer directly.

Figure 3.1.16: Request Propagation to peer behind the firewall

NAT or Network Address Translation is a process that allows the router to translate the private addresses (10.x.x.x, 192.168.x.x and 172.16.x.x) to a real IP addresses. When a peer in the private network needs to access the outside network, its request will be wrapped by the router and the router's IP address will be used instead of its private address. When the response comes, the router will check for the wrapper and forward it to the peer in the private network. So all the work is done by the NAT and the peer does not need to do anything.

14- Shared Resource Distributed Index (SRDI):

JXTA supports a shared resource distributed index (SRDI) to provide an efficient mechanism for propagating query requests within JXTA network. When a peer publishes a new advertisement, it uses the SRDI service to make an index from the name or the ID of the advertisement and forward that index only and not the advertisement itself to its rendezvous peer. The rendezvous will hash the index and according to the hashed value will forward the index to another peer as shown in Figure 3.1.17.

Figure 3.1.17: Request Propagation via Rendezvous Peer

SRDI uses the SHA1 hash addressing scheme where the address space of 160-bit hash is distributed over a set of rendezvous peers. Each rendezvous now is responsible about a range of addresses like the way explained in the DHT section.

Figure 3.1.18: JXTA search request propagation.

In Figure 3.1.18, if peer A initiates a search request, it will be propagated first to local peers in the local subnet via broadcast or multicast and to its registered rendezvous peer Rdv1 at the same time. If the local peers have the requested information they will respond directly to the requesting peer. The rendezvous peer will respond to the requesting peer if it has the requested information in its cache, else if it has the index of the requested information in its SRDI cache, it will contact the peer who published that information and that peer will respond directly to the requesting peer.

If Rdv1 does not have neither the information nor its index, it will walk the set of rendezvous peers based on a limited-range walker algorithm with a query that has TTL until it finds the rendezvous that has the index which will contact the associated peer to respond directly to the requesting peer.

On the other hand if peer A issues new advertisements, it will send an SDRI message with a set of indexes to its rendezvous Rdv1. Rdv1 will replicate those indexes based on there hash values to other rendezvous on the network.

3.2 Content Manager Service (CMS):

CMS [1][2][3][4] is a JXTA service for sharing contents among peers in a peer group. It allows a peer to share contents or to retrieve contents from other peers and to perform a search operation based on the content's name. CMS provides a protocol based on JXTA pipes for transferring contents between peers who are not required to run HTTP in order to exchange contents like other P2P systems.

The content can be any data file like text, music or executable file. Sometimes this content is called codat (code/data) to indicate that it could be a code or data or both.

The content has a unique content identifier which is a 128-bit MD5 checksum of the actual content which makes also easier to tell if two contents are the same for two peers sharing the same content. The content ID is also used when requesting the content data.

CMS content is also has a content advertisement that looks like:

```
<?xml version="1.0">
  <!doctype jxta:contentAdvertisement>
  <jxta:contentAdvertisement>
    <name>index.html</name>
    <cid>md5:1a8baf7ab82c8fee8fe2a2d9e7ecb7a83</cid>
    <type>text/html</type>
    <length>23983</length>
    <description>Web site index</description>
  </jxta:contentAdvertisement>
```

The content advertisement has a name beside the content identifier which is a user defined field and it may be not unique but it must be there, and some other optional fields like the file type, the content length in bytes and a brief description of the content. If the content name and the type are not specified then a default ones based on the content name will be chosen, but there will be no default description unless it is specified.

CMS is not a private network because all of its transmissions are unencrypted and because JXTA peers are not anonymous, sharing copyrighted data with CMS in its current version could lead to user prosecution. Another problem in CMS is that it does not provide virus protection; one way to be protected from viruses is to scan files while arriving.

Initializing CMS is done as a JXTA service that has a self-contained application which handles input and output requests. The initialization process takes two directory names, one is used to store group related files while the other for bookkeeping functions that need to persist between sessions. Only one copy of CMS per group has to be started.

CMS manages a persistent store that contains references to the locally shared files and their advertisements. When a file is shared, the MD5 is calculated and stored with only a reference to the actual content. And when that content is retrieved from remote peer, it will be verified by its MD5 checksum to make sure that it is not changed since it was last shared.

Each instance of CMS manages a single input pipe for receiving both content requests and responses. The advertisement of this pipe is passed in each CMS message. The CMS messages are encoded as JXTA pipe messages using tags to separate each field and relies on JXTA Search to provide this search routing to find a remote content. The CMS itself only supports local content searching within the local peer itself.

CMS messages are:

LIST_REQ: Request message is used to obtain a list of contents shared by a peer. It contains a unique request id, the input pipe advertisement for receiving the responses and an optional search string for filtering the results. This message contains the following tags:

- RID: a unique request ID.
- PIPE: the input pipe advertisement for receiving the responses.
- QSUBSTR: an optional search string for filtering the results.

LIST_RES: Response message, sent by the responding peer and it includes one or more content advertisements for the contents shared by the peer. If a search string is included in the LIST_REQ message, only the advertisements whose names contain this string will be returned. This message contains the following tags:

- RID: request ID of the corresponding LIST_REQ message.
- PIPE: input pipe advertisement for sending content requests.
- COUNT: number of content advertisements in the message.
- CADV: one or more content advertisements.

GET_REQ: Request message, is sent by the requesting peer to fetch certain content (identified by the content id contained in the content advertisement) through the pipe in the LIST_RES message.

This message contains the following tags:

- RID: request ID.
- CID: the requested content ID.
- PIPE: input pipe advertisement for sending content responses.

GET_RES: Response messages, they are sent by the responding peer after checking if the peer is still sharing content for the specified content ID. They include the data for the requested content and have a maximum size of 64Kbytes which is the maximum size of JXTA messages. The responding peer may send one or more of these messages.

This message has the following tags:

- RID: request ID of the corresponding GET_REQ message.
- SIZE: number of bytes in the block.
- OFFEST: offset of the block.
- DATA: data block bytes.

GET_RES messages may not received in order or may be dropped for errors, so the peer receiving those message must be prepared for these issues.

The application must register a listener to be notified when the retrieval is finished. If it does not receive this notification within a certain time, it should abort the transfer and the file has to be requested over again.

The Figure 3.2.1 shows how CMS protocol works:

Peer A sends a LIST_REQ message to peer B, this message contains a pipe to be used by peer B to send LIST_RES message through it. The LIST_RES message also contains a pipe and a number of content advertisements that matches the request. Peer A will send a GET_REQ message that contains the content ID that will be transferred and a pipe so peer B will start sending messages of maximum size 64KB to peer A until the file is finished using that pipe.

Figure 3.2.1: CMS message exchange.

Chapter-4: Protocol Design and Implementation:

In this chapter we will present the details of the implemented protocol and the implementation.

4.1 Overview:

In this section we will give an overview of the entire implementation which has two main parts, the server peer and the client peer:

1- The Server Peer.

This server works as a certificate authority (CA). It issues a signed digital certificate to any peer when it starts the Client application for the first time. The server is also responsible for creating the Token credentials for every peer contacting him and keeping track of those credentials, this means that the server is the only one who will sign, add and subtract Tokens from those credentials to prevent forgery.

2- The Client Peer.

Each peer can communicate with other peers on the network, send messages, advertise files, search for files and download others. The peer also can contact the server peer for accounting purposes.

The client peer creates two JXTA bidirectional pipes, one pipe to deal with server communications and the other pipe to deal with remote peers communications as shown in Figure 4.1.1a.

Figure 4.1.1b shows the initiation of the client peer application. The peer will start the JXTA and connect first to the NetPeerGroup and starts searching for the FileSharingGroup. After joining this main group it will try to register itself to a rendezvous peer and if it could not find a one it will become a rendezvous by itself. At the end it will create the two main pipes and start searching for files and peers in the network and log them in the main window.

Figure 4.1.1a: The Implementation Main Group

Figure 4.1.1b: Peer Initiation procedure

Peer–Server interaction:

When a user starts the Client Peer application, first the peer will create a self-signed certificate to be used in the communication with the server, the peer contacts the server to download a digital certificate signed by the server to be used in all future communications with other peers and with the server too. Included in the identity certificate, the server will send this peer an initial token credential charged with 100 Tokens to be used in later downloads from other peers on the network.

This self-signed certificate is used here for testing only. In real application the peer has to contact the server either offline or by email to obtain the signed identity certificate and to pay for the initial tokens credentials.

After each file transfer transaction, the uploading peer collects a payment from the downloading peer; while the downloading peer receives a receipt from the uploading peer. At the end of each billing period, T_{bill} , each peer sends the total payments and receipts it has collected during T_{bill} to the server. The server then calculates the net credits/debits of the peer in that period. In addition the server uploads a new certificate and credentials to the peer for the new billing period.

The client peer can also contact the server to download a file if it could not find it in the network, this downloading process will cost the peer higher price compared of what will be offered from other peers in the network.

Peer–Peer interaction:

After downloading the identity certificate from the server, the peer will be a legitimate peer and can communicate with other peers on the network. This peer can send messages with its new certificate to other peers and to the server peer too. The peer can also advertise files on the network, and download files advertised from other peers. After downloading a file, the peer should send immediately a payment to the uploading peer. The payment contains its ID and name, the provider peer ID, file name, time stamp, the price and the signature of the downloading peer. If for any reason that payment is not sent immediately, it will be saved in a list and after a short time it will be sent again to the uploading peer.

The timer has been chosen here as ($T_{bill}=20$ min) for testing purposes only, but for a real application the timer may be set for one month so the peer will contact the server after using the service for one month to update its credentials by adding its commission from uploading files and subtracting the tokens it used to download other files, it is like a monthly bill. The longer the billing period is; the less interaction between peers and the server will be. With a fixed billing period peers can use the service without the presence of the server in most of the transactions.

Figure:4.1.2: The Implemented File Sharing Protocol

Figure 4.1.2 summarizes the peer-to-peer and server-to-peer interactions which have been briefly described in section 1.4. Steps 1 to 6 represent the peer-to-peer interactions:

Step-1: peer A will send a downloading order to peer B. The downloading order has the following fields:

PeerA_ID,: this is the unique ID of peer A.

PeerA_Name: this is the name of peer A.

PeerB_ID: this is the unique ID of the uploading peer (peer B).

Filename: this is the file name that peer A wants to download from peer B.

Price: this is the price of the requested file advertised by peer B.

Nonce_A: this is a nonce created by peer A and will be used with another nonce created by peer B in the generation of the symmetric key to be used in the encryption and decryption of the requested file. This nonce will be

created every time peer A wants to download a file. This nonce also is included in the data to generate the signature.

Time Stamp: this is the time of generating the download order and is used to defend against the DoS and Reply attacks as will be explained later in chapter 5.

Signature: this is a signature signed over the complete downloading order signed by peer A's private key, so peer B will validate this signature to make sure that this order is sent by peer A and it is not tampered in transit.

Step-2: peer B will check its buffer; if it is full it will send a "not available message" to the requester. If the buffer is not full it will send an "upload order" message to peer A after verifying the downloading order sent by peer A and the file requested in the order is still available for download.

The "upload order message" has the following fields: PeerB_ID, PeerB_Name, PeerA_ID, Filename, Price, Nonce_B, Time stamp and Signature.

Step-3: peer A will send an "Accept Message" to the uploading peer (peer-B) indicating that it is ready for the file download. This message has the following fields: PeerA_ID PeerB_ID the request ID, the content ID, Time stamp, the pipe it will use to receive the file through and the signature. The content ID is the same ID that advertised by the uploading peer. The uploading peer will wait for this message for one minute at maximum and if it did not receive it, it will flush that order from the buffer. After receiving this message, peer-B will assume that the connection is established and start step-4 and send busy messages to peers who have orders in the uploading peer's buffer. The details of the connection establishment mechanism will be explained in the next section.

Step-4: in this step peer B starts sending the requested file to peer-A. Peer-B will un-share the requested file so it will not be available for other peers and after finish uploading the file it will share it again. By removing the file temporarily from the network, the downloading peer will always contact a free uploading peer.

Once peer A retrieved the file it checks the integrity of the file by hashing it using MD5 hash function and comparing it with the hash value that was advertised by the uploading peer (peer-B). If the verification is correct, peer-A will save that file. If the file is too large, it can be divided into segments and provide integrity check for every segment.

Step-5: in this step peer A will send a payment to the uploading peer B. This payment has the following fields: PeerA_ID, PeerA_Name, PeerB_ID, Filename, Price, Time stamp, Feedback and Signature.

The feedback is optional and should be encrypted by the server's public key so it is hidden from the uploading peer. It includes the actual speed of the downloading process (how many Kbytes/s) and an overall rating which is a number from 0-100% to indicate how satisfied the downloading peer from the uploading peer.

The server will collect those feedbacks in a database to be sent to any requesting peer asking for information about another peer. This collected information will help the system to build a performance scale between peers, so the peer with high rating will be more preferable in the system. The asking peer will see the average speed of the remote peer and its overall average rating and has the choice to deal with that peer or search for more preferable one.

The uploading peer (Peer B) will save this payment in a database for later sending to the server to collect its commission.

Step-6: in this step the uploading peer (peer B) will send a receipt to the downloading peer (peer A) indicating that it received the payment and has the following fields: PeerA_ID, PeerA_Name, PeerB_ID, Filename, Price, Received Payment, Time stamp and Signature. If for any reason the uploading peer does not receive the payment, it will send the receipt to the server indicating that it did not receive the corresponding payment.

Step-7: in this step the peer will start sending the collected payments and receipts during the billing period to the server after a timer (T_{bill}) is expired.

Step-8: in this step the server will credit the uploading peer a commission which is a percentage of the payment as an incentive to make him participate more in the network by advertising more files and staying longer in the network.

The server will also deduct the total tokens from the downloading peer (peer A) and send both the uploading and downloading peers new credentials for the next billing period.

The server will compare the payments with the receipts to make sure that the peer did not claim the payments more than once.

Figure 4.1.3 shows the messages exchanged between two peers in a file downloading process.

Figure: 4.1.3: Peer-Peer Interaction

4.2 Connection Establishment Mechanism:

The connection establishment mechanism is quite straightforward. When a connection request arrives, the uploading peer will put it in the buffer and send a response back. The peer will then wait for a pre-defined period (say 1 minute). If the peer receives a reply to the response, the connection is established and the downloading process may commence. If no reply is received after the pre-defined period, the request will be discarded. In our implementation, the uploading peer only allows one download session at any given time. In order to prevent other downloading peers from sending connection request during the on-going download, the uploading peer will immediately “un-share” the file in the peer-to-peer system once the connection is established.

Even our implementation can only allow one download session; it does allow the uploading peer with the buffer size of Q to handle Q download requests simultaneously. When there are N connections requests arrive at the uploading peer in a burst, where $N \leq Q$, the uploading peer will send responses to all these requests. The first valid reply to the response will trigger the connection establishment between the uploading peer and the peer that sent the reply. The other pending requests will subsequently be discarded and a busy message is sent to the associated downloading peer(s). This approach improves the performance of the implementation under the Denial-of-Service (DoS) attack as we will see in the next chapter.

The following example illustrates the connection mechanism described above. Consider there are 4 connection requests arrive at the uploading peer in a burst as shown in Fig 4.2.1. Suppose $Q=3$, consequently only the first three requests are buffered and responses to them are sent; the last request is dropped. Assume the reply from peer 2 arrives before the replies from the other 2 peers; then peer 2 successfully establishes a connection with the uploading peer. Peer 2 will start receiving the file data while the other 2 peers will receive busy messages.

Figure 4.2.1: Connection establishment mechanism

For security, the connection establishment mechanism is built on top of TLS/SSL protocol or cryptographic algorithms and services provided by Java Cryptographic Extensions (JCE) or Bouncy Castle Crypto APIs.

The use of secure pipes will reduce the communication throughput and increase the Round-Trip-Time (RTT) for each message as stated and measured in [50]. Because of this all messaging between peers in this implementation are encrypted by using the Bouncy Castle APIs implementations of the secure messaging standards known as Cryptographic Message Syntax [7].

As shown in the Figure 4.3.2, the enveloped data is constructed by encrypting the data by a session key and encrypting that key with the

4.3 Security:

A major challenge in P2P networks is security, since there is no centralized server, there will always be malicious peers connected to the system. Those peers can attack the system in many different ways. They can provide false responses to queries, they can send infected files with viruses and they can also route messages to wrong destinations.

In our system, we built the application over the JXTA platform technology where each peer has an advertisement that contains a digital certificate called the identity certificate. This certificate can be used to authenticate the peer and make sure that he or she is a legitimate user by verifying its identity certificate with the server's public key.

The peer could not participate on the network unless it has a valid certificate. When the peer receives a message from a remote peer it will check its certificate after checking its timestamp, if not valid all messages from that peer will be discarded.

The peer builds a temporary database to save the time stamp of last received valid messages and the sender's IDs within the last two minutes to defend against the reply attack.

The process of validating a remote message is shown in Figure 4.3.1. This approach improves the performance of the implementation under the Denial-of-Service (DoS) attack as we will see in the next chapter.

The following example illustrates the connection mechanism described above. Consider there are 4 connection requests arrive at the uploading peer in a burst as shown in Fig 4.2.1. Suppose $Q=3$, consequently only the first three requests are buffered and responses to them are sent; the last request is dropped. Assume the reply from peer 2 arrives before the replies from the other 2 peers, then peer 2 successfully establishes a connection with the uploading peer. Peer 2 will start receiving the file data while the other 2 peers will receive busy messages.

First the peer is in a state of waiting an order. If it received one, it will check the time stamp of the message and it should be less than the threshold of two minutes, if not it will be discarded.

Second it will check if the order is not seen before by checking the temporary database, this to make sure that the order is not a replay attack and not to be processed more than once.

Third it will verify its root certificate to make sure that the remote peer is a legitimate peer. If it is not a legitimate peer it will discard its message and return back to the waiting state.

Finally it will verify the order against the signature attached with this order. If it is verified it will be processed, otherwise it will be discarded.

Figure 4.3.1: State diagram for verifying the received order

For security we can use either secure pipes which are built on top of TLS/SSL protocol or cryptographic algorithms and services provided by Java Cryptographic Extensions (JCE) or Bouncy Castle Crypto APIs.

The use of secure pipes will reduce the communication throughput and increase the Round-Trip-Time (RTT) for each message as stated and measured in [50]. Because of this all massaging between peers in this implementation are encrypted by using the Bouncy Castle APIs implementations of the secure messaging standards known as Cryptographic Message Syntax [7].

As shown in the Figure 4.3.2, the enveloped data is constructed by encrypting the data by a session key and encrypting that key with the

recipient's public key. So the enveloped-data message is then the result of encapsulating the encrypted data along with the information required for each recipient. When the recipient receives the enveloped data message, the recipient will extract the pertinent recipient information to recover the session key which will be used to recover the original data.

The data consists of the original data and a signature to make sure that the data is not tampered in transit and to assure that the message is sent by the peer who claims to be the sender. The signature algorithm used is SHA256withRSA updated by the private key of the sender. When the receiver receives the message he or she has to verify it with the sender's public key. If the message is verified, it will be processed otherwise it will be discarded.

The encryption algorithm used in enveloping the data is AES (Advanced Encryption Standard) which is one of the most popular algorithms used in symmetric key cryptography. AES was developed by Joan Daemen and Vincent Rijmen and called "Rijndael", it has a fixed block size of 128-bits (16 bytes) and a key size of 128,192 or 256 bits in length.

With AES we used the CBC (Cipher Block Chaining mode) which reduces the likelihood of patterns appearing in the ciphertext by XORing the block of data to be encrypted with last block of ciphertext produced and then applying the raw cipher to produce the next block of ciphertext.

Figure: 4.3.2: the process of encrypting a message.

In the process of downloading a file, the file has to be encrypted by a session key known to both the downloader and the up-loader. This key is only used in one file download and if they want to download another file they will create another new key. The method used to encrypt the file is "AES/CTR/NoPadding".

AES is the Advanced Encryption System that has been described above, CTR is the mode in which we are using AES. CTR (Counter Mode) also known as SIC (Segmented Integer Counter), described in RFC3686. It is a streaming block cipher that works by producing a stream of bits (block cipher) that is then XORed with the plain text as shown in the Figure 4.3.3.

Since CTR requires no padding it is suitable for encrypting files with different sizes.

The session key is generated from the downloading peer's nonce (24 bytes) and the initialization vector is generated from uploading peer's nonce (16 bytes).

This mode of block cipher does not provide data integrity; this means that the attacker can still alter the cipher data so it is very important to use integrity protection. In our application the file will be hashed with MD5 to produce a checksum for verification operation.

Figure:4.3.3 CTR (Stream Mode Encryption/ Decryption)

4.3.1 How the implementation addresses the main security requirements:

- a. **Privacy:** it means that the data transmitted from one peer to another should not be comprehensible to any other peer on the network. To provide privacy in our implementation we encrypt all communication messages between peers using symmetric key which is also encrypted by the recipient's public key. In this case only the intended peer can decrypt the message.
- b. **Authentication:** in this process we guarantee that the sender is who he or she claims to be. In our application each message contains the sender's ID which can also be checked from its peer advertisement to make sure that he or she has sent the message.
- c. **Authorization:** to guarantee that the sender is authorized to send the message, we checked the sender's identity certificate which is included in its peer advertisement. If the sender has a valid digital certificate signed by the server peer, this means that it is a legitimate peer.
- d. **Data Integrity:** is to guarantee that the data is not modified in transit. In our application we used the SHA256withRSA algorithm to create a signature of the message, the SHA256 is the digest function and RSA is the encryption algorithm as shown in the Figure 4.3.1.1. The receiver will validate the data by verifying the signature with the sender's public key. Also we used MD5 as a hash function to make sure that the file is not altered while downloading it.

Figure 4.3.1.1: SHA256withRSA signature process

e. Refutability: to guarantee that the message is sent by the peer who claims to be the sender, the signature attached to the message will be verified by the sender's public key. If it is verified, this means that this message is a valid one otherwise it will be discarded.

Each message contains the sender's peer ID which also included in its peer advertisement; if they are different the message will be discarded.

f. Trust: it is one of the major issues in peer-to-peer applications, but because in our implementation we have a server peer that issues the digital certificates to other peers on the network, so the peer who has a valid certificate only will be trusted and will be allowed to communicate with others.

Also because our server peer collects rating information about each peer, so any peer can ask for that information about any other peer and decide to communicate with it or not based on that information. So if two peers offer the same service but one of them has a rating of 60 and the other has a rating of 90, the downloading peer can choose to download from the peer with a higher rating.

4.3.2 How the implementation deals with Peer-to-Peer threats:

a. Privacy:

Peer-to-peer systems make it possible and easy for users to share personal files. In many peer-to-peer systems, the application will allow sharing of a shared folder by default, so any file in that folder by default will be shared with all users on the network. Some of P2P applications also will search for certain type of files and share them automatically with others. This way of file sharing violates the user's privacy because either by mistake or misuse a sensitive file may be exposed to the public.

In our application, the user has the right to share any file or to un-share it when he or she wants. Also the peer shares only an index to the file and not the whole file, so if any remote peer wants to download the file, he or she should send a valid request to the uploading peer. And if the owner peer accepts that request, the file will be transferred.

b. Eavesdropping/ Traffic Analysis:

In this threat an adversary can sniff and store all traffic in the peer-to-peer network. Because in our application all communication messages are encrypted by a symmetric key which is encrypted by the public key of the recipient, the adversary could not learn anything from that enveloped message.

The old symmetric key will be destroyed when the session is over and a new key will be produced every time the two peers want to transfer a file.

c. Man-in-the-Middle attack:

In this attack the adversary must participate in the communication continuously. The adversary will pretend that it is the receiver peer to the sender peer and at the same time as the sender peer to the receiver peer as shown in Figure 4.3.1.2. By acting as proxy and appearing as the trusted client to each side, the adversary can perform various attacks against the confidentiality and the integrity of the data passing through it.

By being in the middle the adversary can attack the connection by eavesdropping, replay attack or denial of service attack.

Figure 4.3.1.2: Man-in-the middle attack

In our application, each peer has its own signed certificate from the server; the certificate is used to sign each message so the remote peer can verify it and make sure that it was sent by who claims to be the sender. So this attack is very hard to affect the network except the denial of service attack which will be explained later.

The adversary can receive the messages from the sender but it could not decrypt it because it does not have the receiver's private key.

d. Duplicate names, peer ID's and pipe names:

In JXTA network the lookup is done by using the name, so duplicate names will make communication between peers unpredictable. To solve this problem we should not allow duplicate names and ID's for peers and their pipes.

Because our implementation has a server peer whose job is to legitimate peers by issuing them a valid certificate, the server in a real application should make sure that the peer's name and ID are unique in the file sharing group.

For the pipe name, we make it unique in our implementation by adding the peer's ID to its name so if the peer's ID is unique then its pipes' names are unique too.

e. Denial-of-Service attack(DoS):

DoS attack is the most financially expensive among all security incidents as stated in [51]. It is an attempt to make a computer or a network resource unavailable to its intended users. The adversary will saturate the victim's machine by external requests so the victim could not respond to legitimate traffic or his response will be very slow. Also the adversary could target and consume the bandwidth between the victim and its intended users so they could not communicate adequately. The adversary could be motivated by revenge, money, and politics or for no rational reason.

Here we would like to discuss the consumption of computational resources, like disk space and CPU time and also the link bandwidth of a victim peer targeted by an adversary peer.

The adversary can sniff or create a message and replay it so many times intending to bring down the victim peer. The adversary should have more bandwidth and CPU power than the victim peer to succeed in consuming the resources of the victim. To defend against this kind of attack, the application we implemented will check first the time stamp of that message. If the message is stale (more than 2 minutes) it will be discarded. In addition our implementation builds a database where the last

received messages are saved temporary for a time threshold of two minutes. The victim peer will check the received message against that database, and if it finds that peer is already in the database, the received message should have a timestamp higher than the saved one. If the message has a time stamp either the same or lower than what is saved in the database, it will be discarded. This makes the victim peer not to respond to any order more than once.

Finally the victim peer will check the validity of the sender's identity certificate and because this adversary does not have a valid certificate or its certificate does not match with signature included in the message, all messages received from it will be discarded and the pipe will be closed immediately. So the adversary needs to invoke the pipe again and send the message. This pipe invoking process takes long time as measured and analyzed by [50].

Based on our analysis in the last section, it seems that the implementation is not vulnerable to the high-level Denial-of-Service attack. To investigate more in this area, we conduct a test where we introduce two adversary peers to the system as shown in Figure 4.3.2.1.

Figure 4.3.2.1

The adversary peer will start sending a false downloading order 60 times to the victim peer and then measure the time it takes to send that order and the time the victim peer takes to validate that order, discard it and close the pipe.

We found that the adversary peer takes longer time on average 214ms to send the false order while the victim peer takes on average 110ms to process that order and discard it. This happens because each time the adversary wants to send a message to the victim it should invoke the pipe because the victim will close the bidirectional pipe after receiving a false message. And this explains why the adversary needs to have more processing power and bandwidth in order to bring down the victim peer.

The victim peer creates one end of the pipe (input end) when it started and does not need to do this more than once while the adversary has to look for that pipe end and create the other end (output end) each time wants to send a message to the victim peer. This pipe invoking process takes time and it depends on the network distance between peers and it could take several hundreds of milliseconds as analyzed and measured by [50]. In our case it is the worst case scenario when the two peers are on the same network.

Also we let the two adversary peers start a loop to send a false order to the victim peer; we see that the victim did not hang. This makes us to believe that the system is some how immune to this kind of attack.

4.4 Certificates and Key Managements:

In our implementation we are using RSA algorithm [3][7] to derive the key pair for each peer. RSA was published by Ronald Rivest, Leonard Adleman and Adi Shamir in 1977. The algorithm gets its security from the difficulty of factoring large numbers. The public and private keys are functions of a pair of large primes. Recovering the original text from its cipher and the public key used to create it is believed to be equivalent to the problem of recovering the primes used to make the keys.

In our implementation we used the X.509 v3 certificates [7] as an identity certificate for each peer. The peer first will generate a key pair based on RSA and send a certificate request to the server peer who works as a Certificate Authority (CA) that issues a signed X.509 v3 certificate to that peer.

X.509 is a certificate format that developed by the International Telecommunication Union (ITU) and it is a standard that has been defined on ASN.1 (Abstract Syntax Notation One) structure as the following:

```
Certificate ::= SEQUENCE {  
    tbsCertificate      TBSCertificate,  
    signatureAlgorithm  AlgorithmIdentifier,  
    signatureValue      BIT STRING }
```

tbsCertificate field contains the data that is used to generate the signature value based on the algorithm specified by AlgorithmIdentifier field.

Java has a built-in support for X.509 certificates that allows us to read and write them.

Figure 4.3.2.1

The adversary peer will start sending a false downloading order 60 times to the victim peer and then measure the time it takes to send that order and the time the victim peer takes to validate that order, discard it and close the pipe

4.5 Network Impacts:

Because the size of the messages (like downloading order, uploading order and payment) exchanged between peers is small (1-1.5KB), we don't see that those messages have big impact on the network. But for search operations for peers and files we limit the peer in the local network to respond only by its peer advertisement and not all known advertisements. In this case the requesting peer will receive one advertisement per remote peer. For example if we have a local network of 10 peers and each of them knows about the others, when a new peer joins the network and issues a search request, it will receive only 10 advertisements while if we did not limit the response it would receive 100 advertisements and this sure will have an impact on the network.

For remote peers outside the local network, the rendezvous peer will respond by all known advertisements.

Also in our implementation, if a peer wants to search for information, it first checks its local cache to see if it has it or not. If yes it will use it and will not issue a remote search to save more bandwidth.

4.6 The structure of the Peer Implementation:

Figure 4.6.1 shows the program structure of the peer implementation and the relations of each class to other classes. We will provide a brief description and the main java methods used in each class in appendix A.4. The complete code is shown in appendix A.6.

Figure: 4.6.1 Peer Application block diagram

Description of the main methods used in the peer implementation:

Main(): the program starts by invoking the main method where we select the main directory to work in. This method will call an introductory class to let the user enter his or her information (username, password, buffer size (1-10) and the file directory he or she will use). If the user enters the correct information and did not close the window, this main class will start a new instance of the application (JXTAFileSharingPeer).

JXTAFileSharingPeer(): this method is the heart of the peer application, it calls first the network manager to create and join the default NetPeerGroup. It calls a method to create a temporary digital certificate and adds that certificate to its peer advertisement to be the temporary root certificate so it can be used in the communication with the server peer for the first time only. This temporary certificate is not secure and used just for testing purposes but in real applications the peer will use the signed certificate from time zero which it gets either offline when subscribing for the service or by email.

It calls the doSecurity() method to create an RSA key pair (private and public keys) and the generateRequest() method to create a PKCS10 Certification Request to be sent to server peer when it is found for downloading the signed digital certificate to be used for later communications.

Then it will call the method "findMainGroupAdvertisement()" to find the main group for this implementation which is called "FileSharingGroup", this group has a static ID that is coded in the program because if any user wants to create it, it will create the same instance of that group so any peer will find and join the same group. If the main group is found, this instance of the application will call the method "joinGroup()" to join it. If not found, it will create it, and then join and get services from it as the first peer on the network.

The services that the peer will get are discovery, rendezvous, pipe and Content Manager Service (CMS) services.

This method will start two JXTA server pipes, one for the communication with the server and the other for the communication with other peers. So it will start two listening threads to listen for connections on both pipes.

It also starts a timer to send collected payments to the server every (T_{bill}) time. This timer is set to 20 minutes in our implementation but it could be longer in real application as we described before.

Finally, this instance will start the `run()` method, where the program loops the network searching for peers and advertised files to log them in the main window of the application. The number of files to be logged in the window has no limit in this implementation while in real application it should have a limit in number of responses the peer will get from each remote peer and a limit to the number of files logged in the window so the system will not be overloaded.

peerDiscoveryListener(): this method will be called from the `run()` method when the peer finds a remote peer advertisement. In this method the peer will check the root certificate of the remote peer, if it is valid, it will add this peer to the list of found peers, if not, it will discard that peer advertisement. If the advertisement belongs to the server and this peer did not contact the server before, it will start the method "`connectToServerFirstTime()`".

connectToServerFirstTime(): in this method, the peer will get and save the main information of the server from its advertisement (like name, ID, certificate and pipe) for later use. The peer will send the server a certificate request to download a signed certificate and the initial token credentials. After that the peer becomes a legitimate peer, so it can download and upload files from the network.

startDownloadOperation(): this method is called when the user chooses a file from the remote files list and clicks on the download button. In this method the peer will build a download order and send it to the file's owner. Before sending the order, first the peer should find the owner's peer

advertisement to validate its root certificate and its pipe advertisement for communications.

buildDownloadOrder(): the peer will build an XML file that contains the following fields: file name, price, the downloading peer ID, downloading peer Name, the other's peer ID Time stamp, signature and a nonce. This nonce will be used in the process of encrypting the file and will be different each time this peer wants to download a file. It also contains a signature to be validated from the receiving peer and to make sure that this peer is the one who wants to download the file.

peerListeningThread(): this thread will listen to connections that may be made on the peer's input pipe, so it will create a JXTA bidirectional pipe for communications with the remote peer. It will check if the buffer is full to discard the request, close the pipe and send the requester a buffer is full message.

If the buffer is not full it will register the handler "peerConnectionHandler()" to the created pipe.

peerConnectionHandler(): it handles the connection between this peer and the requesting peer. First it will check the name space of this message if it is known to this peer or not and then de envelope it using its public key.

Each message or order will be validated by the public key of the sender peer. The validation process starts first by checking the time stamp of the message which should be less than the threshold of two minutes then it will check the time stamp against a temporary database that has the sender's ID and the last sent message time stamp (this combination will be saved for two minutes only). The new message's time stamp should be higher than the one saved in the database. After that the message or the order will be verified against the signature that is attached to the message to make sure that the message is sent by the one who claims to be the sender and the message is not tampered in transit.

If the name space is "download order", it will start the startUploadOperation() method. If it is "upload order", it will start downloading the file. If it is "file not found" message, it will close the pipe since the other peer does not have the requested file any more. If it is "send message", it will display the received message on the screen. And finally, if the message is a "payment response" or "Receipt", the peer will collect this payment/receipt and save it in a database to be sent later to the server.

startUploadOperation(): this method will be started from the other peer who received the download order. The other peer will verify the order as explained before. After that, a window will pop up on the screen asking the user to accept or deny the order. If the peer accepts the order, it creates a new nonce and with the requester's nonce will encrypt the file using the algorithm "AES/CTR/NoPadding" explained before and make it ready for downloading. It also creates an upload order that has the new nonce and some other information sent by the downloading peer and sends it so it can decrypt the file.

decryptFile(): this method will be called from the downloading peer after receiving the upload order and finished downloading the file. In this method the file will be decrypted the same way as explained in the security section and a pop up menu will come on the screen asking the user to choose a name for the file if he or she wants to change its original name and to choose where to save it on the hard disk.

sendPaymentToPeer(): this method will be called when a peer finished downloading and decrypting a file. It will send the other peer a payment as an XML file that contains two main tags: first tag is "payment" that contains buyer's name, buyer's ID, file name, price, seller ID and a feedback. The other tag is a signature from the buyer's private key using the algorithm "SHA256withRSA". If the payment is not sent for any reason, it will be saved in a pending list for later sending when a timer expires.

sendReceiptToPeer(): this method sends a receipt from the uploading peer to the downloading peer when it received a valid payment. The receipt has the uploading peer's name and ID, the file's name and price and the corresponding payment to indicate that the payment is received. If the uploading peer didn't receive a payment it will send this receipt to the server to indicate that it did not receive a payment from that peer.

ServerListeningThread(): this thread listens for connections on the pipe that is dedicated for the communication with the server peer. It will create a JXTA bidirectional pipe and registers a method called "serverConnectionHandler" to handle the connection.

serverConnectionHandler(): this method will validate the server ID and check the name space of the message, if it is one of the known messages, it will de envelope the message and call methods accordingly.

If the message is a certificate request, it will call "generateCertificate()" method to build its new identity certificate from the received message, also it will receive its initial token credentials from the server. The new certificate will replace the old one in its peer advertisement.

If the message is a feedback response, it will display the feedback on the screen and if it is a new credential it will save it and delete the old one.

sendMessageWindow(): this message is called if this peer wants to send a message to another peer on the network. It will create a window where the user can write his or her message and then envelope it using the same method explained in the security section and send it to the other peer after finding its pipe and its valid certificate.

sendPeerInfoRequest(): this method will send a request to the server asking some information about certain peer known to this peer by its name or ID. The server will respond by the requested information if it has it. This peer will send a window to the screen showing the personal information of the requested peer like how long it has been on the network, the average downloading speed and the average over all ratings of that peer.

chooseFile(): this method will ask the user to choose a file from the working directory to share it on the network, the user in our implementation will enter also the file price (how many tokens to download this file) and an optional brief description of the shared file so other users can know what this file is about.

inputFileInfo(): this method is called when the user wants to search for a file by its complete name or a sub string of its name. The method will start a CMS session searching for the file, and checks after a while if it received responses from other peers. The files that match with the request will be displayed in a separate list where the user can choose one or more of them to be downloaded. This method is one of the basis methods that can be used in a Bit Torrent protocol for future improvements where a user can share some parts of a file, and download many different parts from different peers.

4.7 The structure of the Server Implementation:

Figure 4.7.1 shows a block diagram of the main methods in the server side of our implementation. We will provide a brief description and the main java methods used in each class in appendix A.5. The complete code is listed in appendix A.6.

Figure 4.7.1 Server Block diagram

In the following we will give a brief description of the main classes in the server part of our implementation:

Main(): this is the main function where the peer starts a new instance of the main application by calling new JXTAServerPeer() method.

JXTAServerPeer(): this is the heart of the server side of the implementation. It starts by calling the network manager class to create and login to the default main group which is "NetPeerGroup". The server will initialize itself to be a rendezvous peer. The user name is hard coded in the application because of the need to be known to other peers so they can communicate with the server safely. The server will act as a certificate authority (CA), it creates a self signed certificate as a CA certificate and then generates an RSA key pair to generate itself a root certificate signed by the CA certificate. This new certificate will be added to its peer advertisement, so it can be used for the communication with other peers.

The server will create a key store to store its certificate and its private key. It also stores all certificates it generates to other peers to be used later for validations.

The server will start searching for the main group "FileSharingGroup" by using the discovery service taken from the default NetPeerGroup, if the main group is found, it will join and get services from it, otherwise it will create it, join and then get the services from it.

After that the server will start a JXTA server pipe and a listening thread to listen for connections that may be made on that pipe. If a peer attempts to connect to the server, the server will start a JXTA Bidirectional pipe and register a connection handler to handle the communication with other peers.

The server will also start a timer to send collected payments from uploading peers to the downloading peers when the timer (T_{bill}) expires (in real application the timer may be one month).

At last the server will start the run() method in this instance.

run() method: in this method the server will loop searching for peers that joined the main file sharing group and log them in the main window after adding them to a list. The logged peers have to have a valid certificate, if not will be discarded.

This method uses this command to get the remote advertisements of the peers:

```
myDiscoveryService.getRemoteAdvertisements(null,DiscoveryService.PEER,null,null,0,peerDiscList  
ener);
```

In this command, the server will register a listener called "peerDiscLitener()" to listen for responses. When a response comes from a remote peer, this listener will be called.

peerDiscoveryLitener(): this method is invoked when the server receives a remote peer advertisement, it checks if the peer has a valid certificate and calls addPeer() method to add it to the list and show its name on the main window.

connectionHandler(): this method is called when the server receives a connection from another peer on the network. First it will check if that peer has a valid certificate and then it will check the message name space, if it is one of the known tags, it will de envelope the message and respond accordingly:

If the message is a certificate request, it will generate the PKCS10 format of the peer certificate request from the message and call the method "generatePeerCertificate()" method to create an X.509 v3 certificate to that peer, at the same time it will also create for this peer an initial tokens credentials charged with 100 tokens as in Figure 4.6.2 and send them both to the requesting peer using the same pipe the order came through. All communications are encrypted in the same way explained in the security section. The first communication with the peer will be encrypted using its temporary certificate and all this peer can get is a self signed certificate, after that the peer should communicate using the CA signed certificate.

```

<?xml version="1.0"?>
<!DOCTYPE TokensCredentials>
<TokensCredentials>
  <Credential>
    <Name>
      Peer6151
    </Name>
    <ID>
      urn:jxta:uuid-59616261646162614E50472050325033267A3A9A41
      2B49C189EB95FEC26A063D03
    </ID>
    <Value>
      100
    </Value>
    <Time>
      1200179056750
    </Time>
  </Credential>
  <Signature>
      74814b883227271b698eccc974f0e70f4366e6e0dffe8f8e5b7fab3c55b4798b
      a67d3382d5abf1dffe237a295dcc16e732489a6b40ca413d87d0e6e9dbdbc6d601c9e80ce42c455b
      de35cd7c0870aff476bbbf2e0f57d6ded455c77c0373fc0ff6e023b43cb729ddaf65ad58d3b07
      5d648f34d709fa641cbb6b83d33339a0
    </Signature>
  </TokensCredentials>

```

Figure 4.7.2 Tokens Credentials

Every generated certificate will be saved in the server key store with the peer ID as an alias. So when the server wants to validate a payment it uses the buyer's peer ID to get its certificate from the key store and validate the payment with its public key.

If the message has "paymentTag" as a tag, this means that the peer is sending "collected payments" from uploading files. The server will then collect those payments and validate them because each payment includes a signature from the buyer's private key. The peer will credit a percentage (as an incentive) of the collected tokens to the seller peer. After an expiration of the timer (T_{bill}), the server will start searching for the buyer peers and create them new token credentials after deducting the whole number of tokens they bought for. If any payment does not go through to the buyer, it will be added to a pending list for later sending. The server will keep a database for the token credentials that every peer has to prevent forgery.

Because the token credentials are signed by the server private key, we can assure that this system is safe, the peer could not claim that it has enough tokens while it is not, and the payments could not be used by the up loader peer unless they go through the server to legitimate them and create new credentials for that peer.

The server will extract the “feedback” from the received payment and add that information to the database of the uploading peer.

If the message has “peerInfoReq” tag, the peer is requesting information about another peer specified by its ID. The server will respond by calling the method “getPeerInformation()” method to get the specified peer information and send it to the requesting peer.

sendMessageWindow(): this is the same method explained in the peer side, which is used to send messages to a remote peer on the network.

sendPeerPayments(): this method is invoked either by clicking on a “send Payments” button on the main screen or when the timer (T_{bill}) expires. In this method the server will start searching for a peer specified by its ID, if found it will send a new token credentials to that peer after deducting the whole amount of tokens specified in the payment. If for any reason the credential did not go through, the server will keep that credential in the database for later sending when the timer expires again.

4.8 The User Interface:

When the user starts the peer application, first it will show an introductory window as shown in the Figure 4.8.1:

The screenshot shows a dialog box titled "Enter PEER information". It contains the following fields and controls:

- Peer Name:** A text input field containing the name "Khalid".
- Max No of Parallel Downloads:** A dropdown menu currently set to "1".
- Peer "Password":** A password input field with ten dots representing a masked password.
- Peer "Shared Dir":** A dropdown menu currently set to "testDIR".
- Buttons:** "Cancel" and "OK" buttons at the bottom right.

Figure 4.8.1 .

In this window, the user is asked to enter a name, the maximum buffer size, a password and a working directory where the files to be shared are saved.

If the user clicks on cancel button, the application will be aborted, otherwise the application will check if the user entered a valid user name and a password, if not the application will go back to the same window to let the user enter a valid name and password.

The default value of the buffer is one so the peer will handle one order at a time but he can increase this value up to 10 which is the maximum size of the buffer as will be explained later in the analysis chapter.

After clicking the OK button, the system will show the main window of the application like the one shown in the Figure 4.8.2:

Figure 4.8.2

In this window, there are two text areas, the top one is to log what groups the peer is joining, and the bottom one is to log what events are going on the application.

There are also three scroll lists, the first one shows what peers are currently on the network, the second one shows what files this peer is sharing with others, and the last list is to show what files other peers in the main group are sharing. The name of the file shown in both lists includes the original name, the price and the name of its owner. Those three lists will be checked every three minutes, if a peer does not hear from it for three minutes or more, its name will be removed from the list.

If the peer chooses one file from its own shared files then the button “remove file” will be highlighted, and if the chosen file from the remote files list, then the button “download” will be highlighted. This means that the peer could not download its own shared files or remove others files from the list.

When the peer chooses one peer from the top list, then the buttons “send message” and “get Peer Info” will be highlighted. So the peer can send messages to another peer or ask the server for some information about that peer.

If the peer clicks on the “send message” button, a window like the one shown in the Figure 4.8.3 will be displayed. The user can enter the message and either clicks “Cancel” or “Send” to send the message. When the message is received by the remote peer a window like the one shown in the Figure 4.8.4 will display the message. The other peer can click on the OK button to dispose that window.

Figure 4.8.3

Figure 4.8.4

When a peer chooses a file from the shared files and clicks on the download button the application will send a download order as in Figure 4.8.5 to the owner peer. The uploading peer will check first the order's time stamp which should not exceed the threshold (2 minutes), and then it will check the temporary list where the last messages' time stamps and the senders IDs are saved temporary for two minutes. If the sender ID is in that list then the message should have a time stamp higher than the one saved in that list. This to make sure that the peer will not respond to an order more than once and to defend against the replay attack. At last the peer will check the certificate and the signature of the order to make sure that the order is not tampered in transit and the sender is the one who sends that order and not a replay order. If all those tests are passed, a message will appear on the uploading peer screen like the one shown in Figure 4.8.6 asking if it will accept the downloading order or not. If it chooses "Cancel", or the file is not any more for downloading, or even that peer did not take an action for two minutes, the message will disappear and the order will be canceled, and the application will send the downloading peer a "file not found" message.

If the uploading peer accepts the order, it will send the requesting peer an upload order as shown in Figure 4.8.7, and after receiving an accept message from the downloading peer it will start uploading the file.

When the downloading process finished, the implementation will ask the user to choose a name and to select where to save the file as shown in Figure 4.8.8. After that a pup up window as shown in Figure 4.8.9 will be shown on the screen asking if the user wants to send a feedback about the downloading process to the server. At the end, the downloading peer will send a payment to the uploading peer like the one shown in the Figure 4.8.10.

The uploading peer will respond by a receipt as shown in Figure 4.8.11 indicating that it received the payment.

```

<?xml version="1.0"?>
<!DOCTYPE DownloadOrder>
<DownloadOrder>
  <FileName>
    testc545.bat Price =10 From Peer =Peer6460
  </FileName>
  <Price>
    10
  </Price>
  <FromPeerID>
    urn:jxta:uuid-
    59616261646162614E504720503250339C2432D57E334BB9AFEED66AEC75C3A603
  </FromPeerID>
  <FromPeerName>
    Peer-Khalid
  </FromPeerName>
  <ToPeerID>
    urn:jxta:uuid-
    59616261646162614E50472050325033CE31D365C52A441E8F07FDDBF4789EC503
  </ToPeerID>
  <Nonce>
    1781883104856027931591776195996821513606139610821432238349946621816405348035002694675
    8513251541796867438278555060336686726005165994711671802462484115552575890904254863361
    3605888226317366049000502416926513223959923046239924671083265312516931488747360273290
    427055247419166252408199925722182166221385328696763491
  </Nonce>
  <Time>
    1210186843055
  </Time>
  <Cookie>
    6e6f436f6f6b6965
  </Cookie>
  <Signature>
    2c7f5be06ddedd2f3f817e367ce88264424de51e41894db0c6a81165650638957c7da144c9eed4cce3398d6d
    6389edf01f80cb81727ae9f6f2f683b18b018a3b81fda6d87343a7406848cf689055f8c32127c2555a5f6088b
    47e116d3bf952d3d4755739c87991403ea49a71c9f93212f54eeefa7f27a41ae715b51ba92a003ed
  </Signature>
</DownloadOrder>

```

Figure 4.8.5: Download Order

Figure 4.8.6


```

<?xml version="1.0"?>
<!DOCTYPE UploadOrder>
<UploadOrder>
  <FileName>
    testc545.bat Price =10 From Peer =Peer6460
  </FileName>
  <Price>
    10
  </Price>
  <FromPeerID>
    urn:jxta:uuid-
59616261646162614E504720503250339C2432D57E334BB9AFEED66AEC75C3A603
  </FromPeerID>
  <FromPeerName>
    Peer6460
  </FromPeerName>
  <ToPeerID>
    urn:jxta:uuid-
59616261646162614E50472050325033CE31D365C52A441E8F07FDDBF4789EC503
  </ToPeerID>
  <Nonce>
2693592828859707422984729354619092229740913941546574189827958537519526796910333406704
8438738483683350535629230634032194162251434080812616020474979025375039855738143118274
5839434952053005233811370331258282776993410920736927821102247292582775685797587774560
53378269780325319107867965556480282362203366251149138
  </Nonce>
  <Time>
    1210186861555
  </Time>
  <Signature>
31dc193c1a47ebec6e4bb9c91dd43b79036a243b28d7118aaa47e5af562752df7edf0454a5da0b43a33a0dc0
f7c8799442584e555717d02111f381ac6ab7bc88d0470d00c3887525312ef69f9e7eb1baa40e8d370662142
b5d7059095d35afd11ef05196331ea2fdfb92e926afea54d57d39c3743c1330a11b1790f7908e4fbb
  </Signature>
</UploadOrder>

```

Figure 4.8.7: Upload Order

Figure 4.8.8

Enter PEER information

Seller Name : Peer =Hosam-Peer

Seller ID : urn:jxta:uuid-59616261646162614E50472050325033508FD1207B574:

Download Speed (KB/s): 2796

Over All Ratings (0 - 100%): 90

Cancel OK

Figure 4.8.9

```
<?xml version="1.0"?>
<!DOCTYPE Payment>
<Payment>
  <Payment>
    <Buyer>
      Peer-Khalid
    </Buyer>
    <BuyerID>
      urn:jxta:uuid-
59616261646162614E50472050325033CE31D365C52A441E8F07FDDBF4789EC503
    </BuyerID>
    <SellerID>
      urn:jxta:uuid-
59616261646162614E504720503250339C2432D57E334BB9AFEED66AEC75C3A603
    </SellerID>
    <FileName>
      testc545.bat Price =10 From Peer =Peer6460
    </FileName>
    <Price>
      10
    </Price>
    <Time>
      1210186862489
    </Time>
    <Feedback>
      3c0033e96d1e5803c3218107c9c774d39b6512960b8b853f1c7f24bc58d065390cfa14d56b17f7d94496d
0209d6c21b955bf3137591f0e5871876d1a5c1d23335de93a2d6f6f71372da5589039cc456c2cfa6f7b6ca
c496ba953f835bc7e2023ee883240280782739d43f2c1f55cf95e6a89ed5cd04e323b0ae39f583404787f
    </Feedback>
  </Payment>
  <Signature>
    320087440b6e2796a63428c04ca471c4676ab3f6098a7b25c7bd8e8c618ea43cfdacd2f5aa994f7087fb1
3c889a078f822c8f93e357dce53cd39fc67f1f66b7f96b4705286079f1de62cb807e41838523a3d6d4905d
8478d7005f7853438f03bdebdd3564896270489e4412fc4834f09601940f91a9e33d03dc4e94fb4cf7eb
  </Signature>
</Payment>
```

Figure 4.8.10

```

<?xml version="1.0"?>
<!DOCTYPE Receipt>
<Receipt>
  <Receipt>
    <Seller>
      Peer6460
    </Seller>
    <SellerID>
      urn:jxta:uuid-
59616261646162614E504720503250339C2432D57E334BB9AFEED66AEC75C3A603
    </SellerID>
    <ReceivedPayment>
      [B@24e801
    </ReceivedPayment>
  </Receipt>
  <Signature>
4bd3e4f21cdeee7079b5af9a7556b903f7885c44081df54aa5065c3fce9c0524c4ef31d6214bef1f4cacc09
edb1960b572bed0208630ecce8de8e40db007fdc3b4e9742ea64538ae52f4c5869205df133e3bfce70334
1547a9143acedd3d04f550d456e3928c211c75bcf64b635d771987b2990ec94c33dd3e11b384aa15a1f4
  </Signature>
</Receipt>

```

Figure: 4.8.11

If the user chooses a peer from the list and clicks on the "getPeerInfo" button, a message will be sent to the server asking it for some information about that peer, and when the response comes, a message like the one shown in the Figure 4.8.12 will be displayed on the screen.

Figure 4.8.12

When a user click on the “share file” button, a window will be shown on the screen asking the user to select a file and enter the price of downloading it as shown in Figure 4.8.13.

Figure 4.8.13

If the peer clicks on “search file” button, a window like the one in Figure 4.8.14 will be shown on the screen. The user will be asked to enter the full or part of the name of the file to be searched for and some description about it. When the results come, a list like the one in Figure 4.8.15 will be shown to the user who will either selects one or more of the files to start downloading them. And when the downloading process finished, the same process will go as in the first explanation of downloading a single file.

Figure 4.8.14

Figure 4.8.15

If the peer clicks on “Renew Tokens”, a window like the one shown in Figure 4.7.16 will be displayed. It shows how many tokens this peer has and let it enter how many new tokens it needs to buy. When it clicks “Renew Tokens” it will send the order to the server who will validate its certificate and update its credentials and send it back to the requesting peer.

The dialog box titled "Send a Renew Order to..." has a standard Windows-style title bar with minimize, maximize, and close buttons. It contains two text input fields. The first field is labeled "Old Tokens Value" and contains the number "100". The second field is labeled "Buy Tokens (0 - 100):" and contains the number "6". Below these fields are two buttons: "Renew Tokens" and "Cancel".

Figure 4.8.16

If the user clicks on the “Exit” button, the application will quit.

Message	Size (bytes)	Processing Time (ms)
Send File Request	1500	220
Send File Response	1500	100
Send Busy	200	50
Accept message	200	100

Chapter-5: Security Study of the Implementation:

In this chapter we will investigate the robustness of the implementation when the system is under a Denial-of-Service (DoS) attack and test the effectiveness of the connection establishment mechanism described in section 4.2. At the same time we will compare between the P2P and Client-Server systems when both adapt the implemented protocol.

5.1 Comparison between P2P and Client-Server (CS) systems under DoS attack:

In this section we will compare the performances between a peer-to-peer system and a client –server system. The performance measure used is the success ratio, which is defined as the ratio of the number of successful downloads to the number of downloading attempts. An unsuccessful download attempt occurs when the downloading peer cannot find a free uploading peer. We assume that once the attempt is found unsuccessful, the downloading peer will give up or try to download the file from the main server, bypassing the peer-to-peer system.

5.1.1 Simulation Parameters:

In this section we will conduct a simulation using OPNET modeler (ACE Whiteboard) to compare between a P2P and Client-Server (CS) systems with respect to success ratio.

The connection establishment mechanism is the same as described in section 4.2 and shown in Figure 4.2.1. The average round trip time (RTT) is 100ms as measured by [50]. The size and the processing time of each message are taken from running our implementation as in table 5.1.1.1:

Message	Size (bytes)	Processing Time (ms)
Send File Request	1500	220
Send File Response	1500	100
Send Busy	200	50
Accept message	200	100

Table 5.1.1.1: shows the size and processing time of each message in the connection establishment mechanism.

The fake traffic:

We consider a high-level DoS attack where the adversary peer has a valid certificate and can generate a valid order. The adversary can either launch a DoS attack where it sends many orders to the victim peer using one host or launching a DDoS attack where it uses many zombies to simultaneously send many orders to the victim peer. The victim peer will respond to the order and wait for one minute at maximum to receive an answer from the adversary as explained in our connection establishment mechanism before it discards the fake order.

We also consider a low-level DoS attack when the peer sniffs a downloading order and try to replay it many times to the uploading peer who will discard it after validating this order. This validation process takes 100ms on average as measured before.

Figure 5.1.1.1 shows how the uploading peer processes the downloading order and handles the fake traffic.

Figure 5.1.1.1: Processing download order mechanism

The requested file:

We conduct the simulation over two different types of files:

- 1- Song files of uniformly distributed size (2-6 Mbytes).
- 2- Movie or segment files of uniformly distributed size of (90-110 Mbytes).

The P2P network:

The P2P network is composed of 53 downloading peers and 10 uploading peers connected together with 640Kbps connections. The downloading peer will select one of the free uploading peers randomly and send him the downloading order. The adversary peer will choose the free uploading peers and target them in order to deplete their bandwidth and processing powers.

The peer hardware is a SUN Ultra 10 system of 333MHz CPU with Solaris system and a FIFO buffer of size Q.

The Client-Server (CS) network:

The CS network is composed of 53 clients that have the same specifications as in the P2P system and one server that have 10 times the processing and the connection speed of P2P peer. The server has 10 service units so it can serve 10 orders simultaneously. The connection establishment mechanism is the same as in P2P for fair comparison.

The server hardware is a SUN Ultra 40 system of 3000MHz CPU with Solaris system and a FIFO buffer of size 10*Q.

How the simulation works:

The downloading peer will select one of the free uploading peers and send it the downloading request.

After a successful or failed attempt, the peer will repeat the downloading operation after a certain time calculated as follows to make sure that the workload in all scenarios is the same.

$$Workload = \frac{No_legitimate_nodes * T_s}{No_uploading_nodes * T_r} \dots \dots \dots (10)$$

T_s : is the file downloading time and equals the file size/link speed.

T_r : is the average time a node waits before it generates a download request.

If the number of legitimate nodes in P2P and CS systems is 53 and the workload is fixed to 0.6, the downloading time can be calculated or measured from the ACE modeler which includes the processing time of each message in both systems, then we can calculate T_r from equation (10) as shown in table 5.2.1.2.

File Size	T_s (sec)	T_r (sec)
Song file	50	442
Movie file	1250	11045

Table 5.2.1.2: shows the calculated waiting time for a peer to send an order.

5.1.2 Simulation results

We ran the simulation in both systems for two and five hours when the requested file is the song and the movie files respectively. We ran the simulation in five scenarios, one of them when there is no fake traffic while the other four of 30, 60, 120 and 240 fake orders per minute.

Figures 5.1.1.2 and 5.1.1.3 show that the P2P system outperforms the CS system in all scenarios because of two factors favoring the P2P system over the CS system:

Firstly, since the P2P system has a number of uploading peers, a legitimate request has a higher probability of connecting with a free uploading peer than connecting with a server in the CS system. The following example illustrates the reason behind it. Consider the CS case with 10 fake requests arriving just before a legitimate request to the server with Buffer=10. The legitimate request will be blocked in this case.

Now consider the case with the P2P system. Again 10 fake requests are generated and sent before a legitimate request. Since fake requests are generated by a number of zombies and their attacks are usually not coordinated, it is possible that multiple of fake orders may be sent to the same uploading peer, leaving some uploading peers free. Thus a legitimate order may be late but still could make a connection with a free uploading peer.

Secondly, the uploading peer in the P2P system will flush the buffer once it established a connection with one of the downloading peers while in the CS system the orders will not be flushed unless all the service units are busy. So when the uploading peer finishes uploading a file it will start with clean buffer while in CS the service unit will start with almost full buffer in most cases.

Fig 5.1.1.2: Success Ratio vs Fake Traffic (Song file)

Fig 5.1.1.3: Success Ratio vs Fake Traffic (movie file)

To study the effect of increasing the buffer size on the success ratio, we ran the simulations for the song file when the buffer size in the P2P system is 5, 10 and 15 and in the CS system is 50 and 100. The results are shown in Figures 5.1.1.4 and 5.1.1.5 for the P2P and CS respectively.

We can see that increasing the buffer size will increase the success ratio because the legitimate order will not be blocked unless the buffer is full. So increasing the buffer size will increase the probability of the order to be buffered and served when the system is under an attack.

If the system is not under attack, the buffer may have negative impact on the success ratio since the legitimate order will be buffered while the server is already in the process of establishing a connection with one of the previous orders so the order will be flushed from the buffer after some time and may not find a free uploading peer at that time. Moreover the uploading peer will need extra memory and processing power.

Fig 5.1.1.5: Success Ratio vs Fake Traffic (song file) with different queue sizes in CS

The low-level attack:

In this kind of attack, we assume that the adversary can sniff one of the orders and replay it many times in order to bring down the system. The uploading peer in this scenario will check the validity of the order and within 100ms on average will detect and discard it.

We ran the simulation for the P2P and CS systems under two different buffer sizes to see the effect of the buffer size in this kind of attacks. For the P2P system, we used buffer sizes of 1 and 5 while in the CS system we used buffer sizes of 1 and 50.

The simulation results are shown in Figures 5.1.1.6 and 5.1.1.7 for P2P and CS respectively. It is clear that this kind of attacks has not much effect on the system because the server will spend short time in validating the order and discard it. Using a buffer also has not much effect on the success ratio since any increase in the buffer size soon will be filled from the coming fake traffic and the legitimate order will have almost the same blocking probability as if there is no buffer.

Fig 5.1.1.6: Success Ratio vs Fake Traffic for P2P using Song file under low_level attack

Fig 5.1.1.7: Success Ratio vs Fake Traffic for CS using Song file under low_level attack

5.2 Further advantages of the P2P system:

In the P2P system, the number of uploading peers is increasing by time and because our file sharing protocol has an incentive to the uploading peers, those peers who finished downloading the file are highly probable to stay on the network and become up-loaders for the same file.

If we assume that the number of uploading peers in the P2P network is (N) and the peers who join the network and become up-loaders by a rate of β and the uploading peers who leave the network at a rate of α then we can predict the total number of uploading peers in the network at any time by:

$$N_T = N + (\beta - \alpha) * t$$

To keep $(\beta - \alpha)$ increasing, we have to have a good incentive program like the one we implement and discourage other peers from leaving the network by cleaning the network from malicious peers and defend the system against the most popular threats as in our implementation.

Increasing the number of uploading peers will decrease the blocking probability in P2P system and make it more scalable and robust than Client-Server system.

In P2P system, the content is usually distributed among peers so the risk of overloading the server with orders is reduced.

In conclusion, the P2P system is more robust against a DoS attack and more scalable as the population of the P2P system increases, the number of uploading peers will increase especially when there is an incentive mechanism built in the system to encourage more peers to become up-loaders and stay longer on the network.

Also P2P system is more reliable than CS system which is clear from the higher success ratio in all scenarios. More over the P2P system is simpler and cost effective way of sharing files than CS system.

Chapter-6: Future Improvements:

1- Digital Finger Print: in this improvement, each file will have a unique finger print delivered from the server; the finger print is divided into two parts. One part is associated with the peer ID or peer name and the second part should be downloaded from the server and could not be transferred to another peer. When a peer uploads a file to another peer it uploads only the first part of its finger print so it will be known from what peer this file has been downloaded and should not upload the server part of its finger print so the file will be in a low quality format. The downloading peer still can run the file but it has a low quality picture or sound. To run the file in the high quality, the peer needs to contact the server for downloading the unique second part of the files finger print.

By this improvement we assure that the peer could not upload a file free of charge and if this happens we can track that file to the first up loader.

2- BitTorrent Protocol: because we have a server peer in our implementation, it is easy to implement the BitTorrent protocol. All what we need is to find a way to divide the file into many pieces and find a suitable hash function like SHA-1 to hash each piece with the file name and the piece number. All the hash values will be saved in a BitTorrent file with some other information related to the file like the size, name, ID, type and description of that file. This BitTorrent file will be saved on the server.

When a peer wants to download a file it first looks on the server if it has a BitTorrent file and download it. From that BitTorrent file, the peer starts searching for each part of the original file. And when the results come, the peer can choose from where to download each part depending on some criteria like the price of the file the rating of the remote peer.

Because we already have a search mechanism in our implementation and also we have rating information about each peer, it should be easy to implement the BitTorrent protocol. All what we need is to agree on a way to divide the file and the hash function.

3- Revoke certificates: in our implementation the server has a database of all issued certificates linked with the ID of the owner peer. This list can be extended to have if the certificate is revoked for any reason or not. The peer can contact the server to see if the connecting peer certificate has been revoked. The peer certificate can be revoked if the peer has malicious attacks or did not send payments to seller peers or even if it's overall ratings is below threshold.

This revocation list will help to clean the network from malicious peers.

4- Replace the server by a group of trusted peers: the server peer is only important in issuing digital certificates to other peers. This means that peers still can communicate with each other without the presence of the server if they have valid certificates. If we found a way to let the peers download the certificates off line like by email and try to find a super trusted peers that each one has part of a shared key to sign and credit the payments and token credentials then we can transfer the system to a pure peer-to-peer system where there is no server. If one of the trusted peers wants to leave the network, it should hand out its part of the shared key to a newly elected peer based on trust criteria. The payments and the tokens should be signed from all members of the trusted group to be valid. The peer will send the payment to one peer of the trusted group and that peer will forward the order to other members of the group to sign it, once the order is signed by all trusted peers, the one who received the order will send it back to the requester.

Chapter-7: Conclusion and contributions:

In this thesis we discussed how to implement a legal and secure file sharing system using JXTA platform. By using a server in our implementation as a Certificate Authority (CA), we can perform strong authentication and authorization mechanisms to make our implementation secure and by letting the server only to sign and credit the payments sent by peers, the system can be considered to respect the copyrights.

We analyzed the implementation and showed how it addresses the main security concerns and how it defends against most popular Peer-to-Peer threats. At the end we conduct a test simulation and showed that the implementation is robust against the Denial-of-Service attack.

The results that have been obtained shows that implementing file sharing application over P2P architecture provides more crucial advantages than in Client-Server architecture such as fault tolerance, robustness, scalability and improved congestion control.

Our implementation is considered to be easy to use and follow all the requirements that have been stated in section 1.6. Because of all of this we consider the implementation is a success in the field of file sharing applications.

It is clear that we can use Java bindings of JXTA protocols to build a high secure peer-to-peer application. Although the solution may not be perfect as stated in the future improvements section where some functions can be added and after that a large scale testing is needed to judge on its success in the real world implementation.

Chapter-8: References:

- [1] Official homepage of JXTA technology <https://jxta.dev.java.net> .
- [2] Official homepage of JAVA technology <http://java.sun.com> .
- [3] D. Brookshier, N. Krishnan, D. Govoni and J. C. Soto (June 2002), "JXTA: Java P2P Programming". ISBN 0-672-32366-4, Sams Publishing 2001.
- [4] Joseph D. Gradecki (August 2002), "Mastering JXTA: Building Java Peer-to-Peer Applications", ISBN: 978-0-471-25084-5, O'REILY 2002.
- [5] Napster – <http://www.napster.ca>.
- [6] Matei, R.; Iamnitchi, A.; Foster, P., "Mapping the Gnutella network", Internet Computing, IEEE Volume 6, Issue 1, Jan/Feb 2002.
- [7] David Hook, "Beginning Cryptography with Java", Wiley Publishing, Inc 2005, ISBN 0-7645-9633-0
- [8] Freenet –, <http://freenetproject.org>.
- [9] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," *Computer Communication Review*, vol. 31, no. 4, 2001.
- [10] Lincoln Scully, "Network File Distribution with the BitTorrent Protocol", Proceedings of the 7th Winona Computer Science Undergraduate Research Seminar, April 19, 2007, Winona, MN, US.
- [11] Website of BitTorrent protocol www.BitTorrent.org
- [12] Nico Maibaum¹, Thomas Mundt, "JXTA: A Technology Facilitating Mobile Peer-To-Peer Networks", Mobility and Wireless Access Workshop, 2002.
- [13] Daryl C. Parker. Sandra A. Collins, and David C. Cleary, *Ericsson Ireland*. "Building Near Real-Time P-2-P Applications with JXTA", Cluster Computing and the Grid, 2004.
- [14] N. Satyanarayana, N. Subramanian, V. NeelaNarayanan, E. UshaRani, "Design and Development of Community Aware Distributed Content Networking Framework using JXTA", International Conference on Internet and Web Applications and Services, 2006.
- [15] Rita Yu Chen and Bill Yeager *Sun Microsystems, Inc.*, "Java Mobile Agents on Project JXTA Peer-to-Peer Platform", proceedings of the 36th Annual Hawaii International Conference on 2003
- [16] Li Gong • *Sun Microsystems Inc.* "JXTA: A Network Programming Environment", Internet Computing, IEEE Volume 5, Issue 3, May/Jun 2001.
- [17] Zhiming Dai, Zhiyi FANG, Xiao HAN, Fengyuan XU and Hongjun Yang, "Performance Evaluation of JXTA Based P2P Distributed Computing System", Computing, 2006. CIC apos;06. 15th International Conference on Volume , Issue , Nov. 2006.
- [18] Xin Xiang, Yuanchun Shi, Ling Guo, "Rich Metadata Searches Using the JXTA Content Manager Service", Advanced Information Networking and Applications, 2004. AINA 2004. 18th International Conference on Volume 1, Issue , 2004 Page(s): 624 – 629.
- [19] W. Yeager and J. Williams, "Secure Peer-to-Peer Networking: The JXTA Example", *IT Professional*, vol. 4, no. 2, pp. 53-57, Mar/Apr, 2002.
- [20] Ambrosi, E.; Bianchi, M.; Gaibisso, C.; Gambosi, G. "JASMINA: An observing behaviours experiment of java threads migration mechanisms over JXTA", Hybrid Information Technology, 2006. ICHIT apos;06. International Conference on Volume 2, Issue , 9-11 Nov. 2006.

- [21] Hao Shi, Yanchun Zhang, Jingyuan Zhang, Elizabeth Beal, Nick Moustakas, "Collaborative Peer-to-Peer Service for Information Sharing Using JXTA", First International Multi-Symposiums on Computer and Computational Sciences - Volume 1 (IMSCCS'06).
- [22] Bill Yeager, Sun Microsystems, "Enterprise Strength Security on a JXTA P2P Network", Peer-to-Peer Computing, 2003. (P2P 2003). Proceedings. Third International Conference on 2003.
- [23] Nakamura, N.; Takahama, S.; Barolli, L.; Ma, J.; Sugita, K. "A Multiplatform P2P System: Its Implementation and Applications", Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on Volume 1, Issue , 28-30 March 2005.
- [24] E. Halepovic and R. Deters, "JXTA Performance Study," in Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, Victoria, BC, Canada, 2003.
- [25] Jere, A.; Meza, M.; Marusic, B.; Dobravec, S.; Finkst, T.; Tasic, J.F. "Peer to Peer Search Engine and Collaboration Platform Based on JXTA Protocol", EUROCON 2003. Computer as a Tool. The IEEE Region 8 Volume 1, Issue , 22-24 Sept. 2003.
- [26] Antoniu, G. Cudennec, L. Jan, M. Duigou, M., "Performance scalability of the JXTA P2P framework", Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International Publication Date: 26-30 March 2007.
- [27] Schollmeier, R., "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications", Peer-to-Peer Computing, 2001. Proceedings. First International Conference on Publication Date: Aug 2001.
- [28] LiJuan Zhou; ZhiTong Li; Bin Liu., "P2P traffic identification by TCP flow analysis", Networking, Architecture, and Storages, 2006. IWNAS apos;06. International Workshop on Volume , Issue , 1-3 Aug. 2006.
- [29] Hui Zhang; Goel, A.; Govindan, R., "Improving Lookup Latency in Distributed Hash Table Systems Using Random Sampling", Networking, IEEE/ACM Transactions on Volume 13, Issue 5, Oct. 2005.
- [30] Schollmeier, R.; Schollmeier, G. "Why Peer-to-Peer (P2P) Does Scale: An Analysis of P2P Traffic Patterns", Peer-to-Peer Computing, 2002. (P2P 2002). Proceedings. Second International Conference on Volume , Issue , 2002 Page(s): 112 – 119.
- [31] Martini, B.; Baroncelli, F.; Castoldi, P.; Aprigliano, A. "Experimental validation of a service oriented network architecture applied to global grid computing", Automated Production of Cross Media Content for Multi-Channel Distribution, 2005. AXMEDIS 2005. First International Conference on Volume , Issue , 30 Nov.-2 Dec. 2005.
- [32] Yeager, B., Sun Microsystems: "Enterprise strength security on a JXTA P2P network", Peer-to-Peer Computing, 2003. (P2P 2003). Proceedings. Third International Conference on 1-3 Sept. 2003, page(s): 7- 8.
- [33] Amit Mathur, Suneuy Kim, Mark Stamp: "Role Based Access Control and the JXTA Peer-to-Peer Framework", Proceedings of 2006 International Conference on Security & Management, June, 2006.
- [34] S. Kazushi, O. Takayoshi, "Peer-to-Peer Content Distribution Solution", NEC TECHNICAL JOURNAL Vol.1 No.1/2006.
- [35] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J. Hugly, E. Pouyoul, B. Yeager: "Project JXTA 2.0 Super-Peer Virtual Network", 2003

- Sun Microsystems,
<http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf>.
- [36] J. Y. Tham, S. L. Lee, C. E. Tan, L. C. Tee, "A Distributed Peer-to-Peer Platform for Synchronized Group Collaboration and Knowledge Sharing," 2004 International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES), Wuhan, Hubei, China, Sept. 12-16, 2004.
 - [37] Sun Microsystems; Guide September 10, 2007: "JXTA Java Standard Edition v2.5: Programmers Guide".
 - [38] Jan Jurjens; "The Security Analysis of Crypto-based Java Programs using Automated Theorem Provers", Automated Software Engineering, 2006. ASE apos;06. 21st IEEE/ACM International Conference on Volume , Issue , Sept. 2006 Page(s):167 - 176.
 - [39] Nicolas C. Liebau, Vasilios Darlagiannis, Andreas Mauthe and Ralf Steinmetz: "Token-based Accounting for P2P-Systems", SpringerLink 2005.
 - [40] C. He and J. Mitchell: "Security Analysis and Improvements for IEEE 802.11i", The 12th Annual Network and Distributed System Security Symposium (NDSS'05), pages 90-110. Feb. 2005.
 - [41] Scott Oaks, Bernard Traversat, Li Gong, "JXTA in a Nutshell", O'REILLY September 2002.
 - [42] Elliotte Rusty Harold, 2nd Edition, (Auguste 2000), Java Network Programming (ISBN: 1-56592-870-9), O'REILY 2000.
 - [43] David J. Eck (eck@hws.edu), (Version 5.0, December 2006) "Introduction to Programming Using Java", <http://math.hws.edu/javanotes>.
 - [44] Kyle London, "Mastering Algorithms with C", O'REILLY, First Edition August 1999 , ISBN 10: 1-56592-453-3.
 - [45] Das, S.M. Pucha, H. Hu, Y.C. , " Distributed Hashing for Scalable Multicast in Wireless Ad Hoc Networks", Parallel and Distributed Systems, IEEE Transactions on March 2008, Volume: 19, Issue: 3.
 - [46] Schutt, T.; Schintke, F.; Reinefeld, A., "Structured Overlay without Consistent Hashing: Empirical Results", Cluster Computing and the Grid Workshops, 2006. Sixth IEEE International Symposium on Volume 2, Issue , 16-19 May 2006.
 - [47] Leonard Kleinrock, "Queueing Systems Volume I: Theory". ISBN 0-471-49110-1. John Wiley & Sons, Inc. 1975.
 - [48] K. Leibnitz, T. Hoßfeld, N. Wakamiya, M. Murata "Peer-to-Peer vs. Client/Server: Reliability and Efficiency of a Content Distribution Service", 2007.
 - [49] Lisa McGarthwaite , "Client-Server versus Peer-to-Peer Architecture: Comparisons for Streaming Video", Proceedings of the 5th Winona Computer Science Undergraduate Research Seminar, April 20-21, 2005, Winona, MN, US.
 - [50] E. Halepovic , R. Deters , "The Cost of Using JXTA". In: Proc. Of Third International Conference on Peer-to-Peer Computing 2003; pp. 160-167.
 - [51] L. Gordon et al., CSI/FBI Computer Crime and Security Survey, Computer Security Inst., 2004.
 - [52] R. Jain, D. Chiu, and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems," Tech. Rep. DEC TR- 301, Digital Equipment Corporation, 1984.

Chapter-9: Appendices:

Appendix A.1

Sample of peer advertisement:

```
<?xml version="1.0"?>
<!DOCTYPE jxta:PA>
<jxta:PA xmlns:jxta="http://jxta.org">
  <PID>
    urn:jxta:uuid-59616261646162614E50472050325033267A3A9A412B49C189
    EB95FEC26A063D03
  </PID>
  <GID>
    urn:jxta:uuid-DCEF4386EAED4908BE25CE5019EA02
  </GID>
  <Name>
    Peer6151
  </Name>
  <Desc>
    this is PEER = Peer6151
  </Desc>
  <Svc>
    <MCID>
      urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000805
    </MCID>
    <Parm>
      <jxta:RA xmlns:jxta="http://jxta.org">
        <Dst>
          <jxta:APA xmlns:jxta="http://jxta.org">
            <EA>
              relay://uuid-59616261646
              162614E50472050325033267A3A9A412B49C189EB95FEC26A063D03
            </EA>
            <EA>
              cbjx://uuid-596162616461
              62614E50472050325033267A3A9A412B49C189EB95FEC26A063D03
            </EA>
            <EA>
              jxtatls://uuid-596162616
              46162614E50472050325033267A3A9A412B49C189EB95FEC26A063D03
            </EA>
            <EA>
              tcp://192.168.10.5:9799
            </EA>
          </jxta:APA>
        </Dst>
      </jxta:RA>
    </Parm>
  </Svc>
  <Svc>
    <MCID>
      urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000105
    </MCID>
    <Parm>
      <RootCert xml:space="preserve" xmlns:jxta="http://jxta.o
      rg" type="jxta:cert">
        MIICJCCAY+gAwIBAgICGr8wDQYJKoZIhvcNAQEFBQAwFTET
        MBEGA1UEAxMKU2VydMvYUGVlcjAe
        Fw0wODAxMTIyMzAzMjZaFw0wODAxMTMxMjU4NDJhMBMxETAPBgNVBAMTCFBlZXI2MTUxMIGFMAOG
        CSqGSIb3DQEBAQUAA4GNADCBiQKBgQCf98ooZ2SSVZLJ7/EkoOoNRG200JhvITd5OgsD0PT65D0q
        4OJS9tintb734Jzy0RJ3DwjDZuKoAXnbwH8I8cGMeYhBg68NWz2kCqdwuWJdFCqunYLHFkKryk6iG
        34wbQDwhfKkMsA5x+Q1I8IY3PV/M+1eU8TBT3rJX0ZoKgFAY5QIDAQABo4GGMIGDMEQGA1UdIwQ9
        MDuAFNNAvMTYw7Aut8kun3RfDk6FY2WRor+kHTAbMRkwFwYDVQQDExBsb290IENlcnRpZmljYXRl
        ggIzeDAdBgNVHQ4EFgQUae5HGFRVo4ntyKLoJNyF/M7Myq0wDAYDVROTAQH/BAIwADAObgNVHQ8B
        Af8EBAMCBAwAwDQYJKoZIhvcNAQEFBQADgYEADhkmJHOM5DVH90dfwsLE6QVl/femWM6QU+Efl1NR
        4IHOy9ntZh5VNSLEvhBn6uxpmB1OZheLV9WZ80zuB1OOOZy8WMPA75CJ6+3FN6qcmdLgZgAHEwT
        IitZTltRsbuPZT8VQ4iW4b81A3OWeBbYXR0ISAnuMMf+vh8GK2+P0BA=
      </RootCert>
    </Parm>
  </Svc>
</jxta:PA>
```

Appendix A.2

Sample of peer group advertisement:

```
<?xml version="1.0"?>
<!DOCTYPE jxta:PGA>
<jxta:PGA xmlns:jxta="http://jxta.org">
  <GID>
    urn:jxta:uuid-DCEF4386EAED4908BE25CE5019EA02
  </GID>
  <MSID>
    urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE000000010306
  </MSID>
  <Name>
    FileSharingGroup
  </Name>
  <Desc>
    File Sharing Main Group
  </Desc>
</jxta:PGA>
```

Appendix A.3

Sample of pipe advertisement:

```
<?xml version="1.0" encoding="UTF-8"?>
<jxta:PipeAdvertisement>
  <Name>JXTA-CH20EX1</Name>
  <Id>urn:jxta:uuid-9CCDF5AD8154D3D87A391210404E59BE4B888209A2241A4A162A10916074A9504</Id>
  <Type>JxtaUnicast</Type>
</jxta:PipeAdvertisement>
```

Appendix A.4

The methods to generate key pairs and certificate request are:

- a. doSecurity(): this method is to generate an RSA key pair.

```
doSecurity(): KeyPair
new SecureRandom();
KeyPairGenerator.getInstance("RSA","BC");
KeyPairGenerator.initialize(1024, sr);
```

- b. generateTempCertificate(): this method to generate the temporary root certificate which is used for the initial communication with the server only.

```
generateTempCertificate(): X509Certificate
new X509V1CertificateGenerator
();
setSerialNumber();
setIssuerDN();
setNotBefore();
setNotAfter();
setSubjectDN();
setPublicKey();
setSignatureAlgorithm();
```

- c. generateRequest(): this method is to generate a PKCS10 Certification Request to be sent to the server when it is found.

generateRequest():PKCS10CertificationRequest new PKCS10CertificationRequest();

- d. addCertToPeerAdv(): it is used to add the signed certificate to its peer advertisement when the peer receives it from the server.

addCertToPeerAdv():PeerAdvertisement new Certificate(); setCertificates(newChain); putServiceParam();
--

The methods to start JXTA platform are:

- a. NetworkManager implements RendezvousListener{}: it is used to start the JXTA platform, to create the NetPeerGroup and to login using its name and password. It also has a stop() method to exit from the JXTA network. It has a method to connect to a rendezvous or to be a rendezvous if it could not found one.

NetworkManager:		
Start(): setPeerID(); setName(); setDescription(); setMode(); setPrincipal(); setPassword(); addSeedRendezvous(); addSeedRelay(); new NetPeerGroupFactory(); getRendezVousService();	login(); new AuthenticationCredential(); setAuth1_KeyStorePassword(); setAuth2Identity(); setAuth3_IdentityPassword(); join(); stop(): stopApp(); unref();	waitForRendezvousConnnection: void isConnectedToRendezVous(); isRendezVous()

The methods to Create or search for the main group (FileSharingGroup) are:

- a. createPeerGroupID(myStringID): this method is to create a fixed ID for the main group from a given string. The ID will be the same each time the application will start to not overload the network with many groups that have the same name but different IDs.

createPeerGroupID ():PeerGroupID IDFactory ();

- b. findMainGroupAdvertisement(Key, Value): this method is to search for the main group by its ID, if it is found it will be joined by calling joinGroup() method.

findMainGroupAdvertisement ():boolean getRemoteAdvertisements (); getLocalAdvertisements();

- c. createPeerGroup(groupID, groupName, groupDescription): this method is to create the main group if the peer could not found it.

createPeerGroup ():PeerGroup getAllPurposePeerGroupImplAdvertisement (); newGroup ();

- d. joinGroup(myLocalGroup): this method is called to join the main group that either found or created.

joinGroup ():void new AuthenticationCredential (); getMembershipService (); apply(); isReadyForJoin();
--

join();

- e. **getServices(peerGroup)**: is a method to get the necessary services that will be used from the main group.

getServices():void	
getDiscoveryService();	getPipeService();
getRendezVousService();	new CMS();

The methods to share or search for files are:

- f. **chooseFile()**: is a method to share a file on the JXTA network.

chooseFile():void
new JComboBox();
addActionListener();
getContentManager().share();

- g. **inputFileInfo()**: is a method to search for a specific file on the JXTA network. It uses the method **displaySearchResults()** to display the results and let the user choose what file or files he or she wants to download.

inputFileInfo():void
new JButton();
addActionListener();
getContentManager().share();

- h. **startSearchForFile()**: this method is to launch a search for a file by a substring or for all files on the network by passing a null as a search criteria.

startSearchForFile():void
new ListRequestor();
activateRequest();
getResults();
new GetRequestor();

The methods to log and update the files list are:

- updateFileList()**: if it is a new file, it will be added to the list. And if it is an old file its time stamp will be updated.
- FilesInfo()**: it will create a new record for the new file.
- addFile()**: it will add the new file to the list.
- removeFile()**: it will remove the file that has not been found for more than three minutes.
- setFilesList()**: it will log the remote files in the display window.
- setLocalFilesList()**: it will log the local files in the display window.

The methods to log and update the peers list are:

- PeersInfo(adv)**: it will create a new record for the new peer.
- addPeer()**: it will add the new peer in the list.
- removePeer()**: it will remove peer that has not been hear from more than three minutes.
- updatePeersInfo()**: it will update the peer's record with the new data like the feedback from another peer.
- setPeersList(peersInfo)**: it will log the peers in the display window and call the **removePeer()** method to the peers who are outdated.

The methods to send messages to the server are:

- f. `connectToServerFirstTime()`: this is the method to handle the first communication between the peer and the server when the server found for the first time.

<code>connectToServerFirstTime ():void</code>	
<code>findPipeAdvertisement ();</code>	<code>connect();</code>
<code>new JxtaBiDiPipe ();</code>	<code>new Message();</code>
<code>setRelaiable();</code>	<code>sendMessage();</code>

- g. `sendPeerInfoRequest()`: this method is to send a request to the server asking for some information about a remote peer like the downloading speed and the overall rating of that peer. So the local peer can decide if it wants to deal with that peer or not.

<code>sendPeerInfoRequest ():void</code>	
<code>new JxtaBiDiPipe ();</code>	<code>new Message();</code>
<code>setRelaiable();</code>	<code>sendMessage();</code>
<code>connect();</code>	

The methods to accept messages from the Server are:

- a. `generateCertificate()`: to build its identity certificate from the stream received from the server.
- b. `displayPeerInfo()`: this method to display the feedback about a remote peer received from the server.
- c. `addCertToPeerAdv()`: this method to add the signed identity certificate received from the server in the local's peer advertisement.

The methods to send messages to peers are:

- a. `startDownloadOperation()`: this method will be started when the local peer wants to download a file from a remote peer. It validates its root certificate and then creates a download order and sends it after invoking the pipe.

<code>startDownloadOperation ():void</code>	
<code>new JxtaBiDiPipe ();</code>	<code>new BigInteger();</code>
<code>setRelaiable();</code>	<code>buildDownloadOrder();</code>
<code>new Message();</code>	<code>validateCertificate();</code>
<code>connect();</code>	<code>sendMessage();</code>

- b. `decryptFile()`: this method is used to build the symmetric key and the initialization vector based on the information received from the remote peer to decrypt the requested file.

<code>decryptFile ():void</code>	
<code>new SecretKeySpec();</code>	<code>startOutputBiDiPipe();</code>
<code>new IvParameterSpec();</code>	<code>sendPaymentToPeer();</code>
<code>new ByteArrayOutputStream();</code>	<code>addPendingPayment();</code>
<code>new CipherOutputStream ();</code>	<code>sendFeedbackToServer();</code>
<code>createPayment();</code>	

- c. `buildDownloadOrder()`: this is the method to build the download order based on the local and remote peers information and the requested file.

<code>buildDownloadOrder ():void</code>	
<code>newStructuredDocument();</code>	<code>initSign ();</code>
<code>createElement();</code>	<code>update ();</code>
	<code>sign();</code>

- d. `sendPaymentToPeer()`: this is the method to send the payment to the remote peer after a successful file download.
- e. `sendMessageWindow()`: is the method to send a general message to the remote peer.

<code>sendMessageWindow():void</code>	
<code>new JButton();</code>	<code>findPeerCert();</code>
<code>new JTextArea();</code>	<code>new Message();</code>
<code>startOutputBiDiPipe();</code>	<code>sendMessage();</code>

The methods to accept messages from peers are:

- a. `startUploadOperation()`: this method is used when the peer receives a download order from a remote peer. It will verify the order as explained in the security section. Then it will generate a nonce and the symmetric key to encrypt the file. After that the local peer will build an upload order and send it to the remote peer. If the peer under a DoS attack it will challenge the downloading peer with a cookie and close the pipe. The downloading peer has to send a new order with the correct cookie otherwise it will be discarded. If the uploading peer received a valid download order, it will create the upload order and send it to the downloading peer.

<code>startUploadOperation():void</code>	
<code>new AcceptWindow(dispatchData);</code>	<code>new Message();</code>
<code>new BigInteger();</code>	<code>sendMessage();</code>
<code>new encryptFile();</code>	<code>Pipe.close();</code>
<code>buildAcceptOrder();</code>	

- b. `buildAcceptOrder()`: this method is to build the upload order and called from the previous method "`startUploadOperation()`".

<code>buildAcceptOrder():void</code>	
<code>new StructuredDocument();</code>	<code>initSign();</code>
<code>createElement();</code>	<code>update();</code>
	<code>sign();</code>

- c. `encryptFile()`: this method is used to encrypt the requested file based on the key generated from the two nonces of both peers.

<code>encryptFile():void</code>
<code>new SecretKeySpec();</code>
<code>new IvParameterSpec();</code>
<code>new ByteArrayOutputStream();</code>
<code>new FileInputStream</code>
<code>new CipherInputStream();</code>

Utilities methods are:

- a. `findPeerAdvertisement(key, value)`: it is used to find the peer advertisement by launching a local and then a remote searches for the peer advertisement specified by its peer ID.

<code>findPeerAdvertisement():PeerAdvertisement</code>
<code>getLocalAdvertisements();</code>
<code>getRemoteAdvertisements();</code>

- b. `findPeerCert(peerID)`: it is used to find the peer certificate by calling `findPeerAdvertisement()` and then call `ExtractCertificate()` methods.
- c. `ExtractCertificate(peerAdv)`: it is used to extract the peer's identity certificate from its peer advertisement.

<code>ExtractCertificate():X509Certificate</code>
<code>getServiceParam();</code>

new Certificate();

- d. `envelopeData(dataBytes, cert)`: it is used to envelop the message to be sent to the remote peer using the public key from its root certificate.

envelopeData():CMSEnvelopedData	
new CMSEnvelopedDataGenerator();	new CMSProcessableByteArray();
addKeyTransRecipient (cert);	Generate();

- e. `deEnvelopeData()`: it is used to decrypt the message received from the remote peer using local peer private key.

deEnvelopeData():byte[]	
new CMSEnvelopedDataGenerator();	setSerialNumber();
addKeyTransRecipient (cert);	setIssuer();
new CMSEnvelopedData();	getRecipientInfos();
new RecipientId();	getContent();

- f. `verifyOrder(ord)`: it is used to verify the message received from the remote peer by checking its time stamp which should be less than two minutes and if this message is not seen before. Then it will check and validate its identity certificate and the signature of the message to make sure that this message is sent by the remote peer himself and it is not tampered in transit.

verifyOrder ():boolean	
initVerify(pbKey);	signature.update(elmValue);
System.currentTimeMillis();	signature.verify(sigValue);

- g. `clearCache(rootDir)`: this method is to clear the working directories and it will be called when the peer terminates the application.

Appendix A.5

The methods to start JXTA platform are the same as in the peer part.

The methods to Create or search for the main group (FileSharingGroup) are the same as in the peer part.

The methods to log and update the Peers in the list are the same as in the peer part except that the server will add a feedback to every peer record when it received it from other peers. The feedback will be averaged over the number of feedbacks the server will get from every peer.

The methods to generate key pairs and CA certificate are almost the same as in the peer part except that the server will generate a self signed certificate as Certificate Authority (CA) certificate and then generate another certificate to be signed with CA to be its root certificate. For the peer certificates the server will generate a version 3 X509 certificate signed by its root certificate and send it to the contacting peer.

The methods to send messages to peers are:

- a. `sendMessageWindow(peerID)`:is used to send message to the peer.

sendMessageWindow():void	
new JButton();	findPeerCert();

new JTextArea();	new Message();
startOutputBiDiPipe();	sendMessage();

- b. `sendPeerPayments(peerID,cert,pipe)`: is used to send pending payments to the buyers and it will call the `updateTokensCred()` method to send the buyer new credentials after deducting the number of tokens he or she used.

sendPeerPayments():void	
startOutputPipe();	new Message();
findPeerCert();	sendMessage();
updateTokensCred();	pipe.close();

The methods used to accept messages from peers are:

- a. `generatePeerCertificate()`: is used to create a signed root certificate to the contacting peer.

generatePeerCertificate():X509Certificate	
new X509V3CertificateGenerator();	setSubjectDN();
setSerialNumber();	setPublicKey();
setIssuerDN();	setSignatureAlgorithm("SHA1WithRSAEncryption");
setNotBefore();	addExtension();
setNotAfter();	

- b. `updatePeersInfo()`: is used to update the peer information when the server receives a feedback from the buyer peer about the seller peer.
- c. `getPeerInformation()`: is used to send information like the downloading speed and the overall rating of a peer to the requesting peer.
- d. `addPeerPayment()`: is used to add the payment received from a seller peer to the database after verifying the payment itself and the sending peer certificate and its time stamp.
- e. `createTokenCred`: is used when the peer first contacting the server asking for new credentials. The first time the token credentials will be charged with 100 tokens.

createTokenCred():StructuredDocument	
newStructuredDocument();	initSign();
createElement();	update();
	sign();

- f. `updateTokensCred()`: is used when a timer expires by the server to update the credentials of the peer when he or she either download or upload a file.

updateTokensCred():StructuredDocument	
createTokenCred();	

Appendix A.6

The complete code of the Peer and Server parts and how to execute them are included in the CD attached.

Chapter-10: Glossary:

Advertisement: JXTA's language-neutral metadata structures that describe JXTA resources such as peers, peer groups, pipes, and services. Advertisements are represented as XML documents.

API: Application Programming Interface.

Credential: A token used to uniquely identify the sender of a message; can be used to provide message authorization.

DHT: Distributed Hash Table.

HTTP: Hypertext Transfer Protocol.

JXTA: JXTA is not an acronym, but is derived from the word Juxtapose. It is a recognition that peer-to-peer is juxtaposed to client server or Web based computing.

JXTA Core Specification: It consists of the required components and behaviors which are present in all conforming JXTA implementations. This includes the Endpoint Routing Protocol (ERP) and the Peer Resolver Protocol (PRP).

JXTA Standard Services: They are optional JXTA components and behaviors. Implementations are not required to provide these services, but are strongly recommended to do so. This includes Peer Discovery Protocol (PDP), Peer Information Protocol (PIP), Pipe Binding Protocol (PBP) and Rendezvous Protocol (RVP).

Message: It is the basic unit of data exchange between JXTA peers; each message contains an ordered sequence of named sub-sections, called message elements, which can hold any form of data.

Module: It is an abstraction used to represent any piece of "code" used to implement a behavior in the JXTA world. Network services are the mode common example of behavior that can be instantiated on a peer.

Module Class: It represents an expected behavior and an expected binding to support the module; is used primarily to advertise the existence of a behavior.

Module Implementation: It is the implementation of a given module specification; there may be multiple module implementations for a given module specification.

Module Specification: It describes a specification of a given module class; it is one approach to providing the functionality that a module class implies. There can be multiple module specifications for a given module class.

Network Peer Group: It is the base peer group for applications and services within the JXTA network. Most applications and services will instantiate their own peer groups using the Network Peer Group as a base.

Peer-to-Peer (P2P): It is a decentralized networking paradigm in which distributed nodes, or peers, communicate and work collaboratively to provide services.

Peer: It is any uniquely identifiable networked device which is capable of interacting with other network devices using standardized protocols.

Peer Group: It is a collection of peers that have a common set of interests and have agreed upon a common set of services.

Peer Resolver Protocol: it is a JXTA protocol used to provide a generic query/response interface applications and services can use for building resolution services.

Rendezvous Peer: An infrastructure JXTA peer which assumes additional responsibilities for the propagation of messages within a peer group.

Resolver Service: It is responsible for the routing and processing queries and responses within peer groups. It uses the Peer Resolver Protocol (PRP) for transmission of queries between peers.

SRDI: Shared Resource Distributed Index. It is a JXTA service used by JXTA peers to make an index for the advertisement they want to publish.

TCP: Transmission Control Protocol.

TLS: Transport Layer Security.

URL: Universal Resource Locator.

UDDI: Universal Description, Discovery and Integration.

UDP: User Datagram Protocol

XML: Extensible Markup Language, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them.

World Peer Group: It is the most fundamental peer group within the JXTA network. This core peer group is generally responsible only for management of physical network connections.

④ BL-62-42