

STARS


University of Central Florida
STARS

Electronic Theses and Dissertations, 2004-2019

2006

Sarp Net: A Secure, Anonymous, Reputation-Based, Peer-To-Peer Network

Sean Mondesire
University of Central Florida

 Part of the [Electrical and Computer Engineering Commons](#)
Find similar works at: <https://stars.library.ucf.edu/etd>
University of Central Florida Libraries <http://library.ucf.edu>

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Mondesire, Sean, "Sarp Net: A Secure, Anonymous, Reputation-Based, Peer-To-Peer Network" (2006).
Electronic Theses and Dissertations, 2004-2019. 6128.
<https://stars.library.ucf.edu/etd/6128>



SARP NET: A SECURE, ANONYMOUS, REPUTATION-BASED, PEER-TO-PEER
NETWORK

by

SEAN C. MONDESIRE
M.S. University of Central Florida, 2006

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2006

© 2006 Sean Courtland Mondesire

ABSTRACT

Since the advent of Napster, the idea of peer-to-peer (P2P) architectures being applied to file-sharing applications has become popular, spawning other P2P networks like Gnutella, Morpheus, Kazaa, and BitTorrent. This growth in P2P development has nearly eradicated the idea of the traditional client-server structure in the file-sharing model, now placing emphasizes on faster query processing, deeper levels of decentralism, and methods to protect against copyright law violation.

SARP Net is a secure, anonymous, decentralized, P2P overlay network that is designed to protect the activity of its users in its own file-sharing community. It is secure in the fact that public-key encryption is used to guard eavesdroppers during messages. The protocol guarantees user anonymity by incorporating message hopping from node to node to prevent any network observer from pinpointing the origin of any file query or shared-file source. To further enhance the system's security, a reputation scheme is incorporated to police nodes from malicious activity, maintain the overlay's topology, and enforce rules to protect node identity.

ACKNOWLEDGMENTS

I would like to give thanks and express my deepest gratitude to all of those of whom have supported me throughout the course of this work and my education: my mother Sharon Mondesire, aunt Karen Coke, grandparents Courtland and Sylvia Coke, and Helen Welch for their love and encouragement; my advisor, Dr. Joohan Lee, for his guidance, inspiration, and invaluable suggestions; my research lab-mates Hua, JianYong, and Soyoung for their insightful discussions and assistance; and UCF's School of Electrical Engineering and Computer Science for the financial support and allowing me to pursue a higher education in the field that I love.

TABLE OF CONTENTS

LIST OF FIGURES	vii
LIST OF TABLES	viii
LIST OF ACRONYMS/ABBREVIATIONS	ix
CHAPTER ONE: INTRODUCTION.....	1
CHAPTER TWO: BACKGROUND.....	3
2.1 Centralized Servers:	3
2.2 Decentralized Servers:	6
2.3 Popular File-Sharing P2Ps:	8
2.3.1 Gnutella:.....	8
2.3.2 KaZaa and FastTrack:	11
2.3.2 BitTorrent:.....	13
2.4 Classification Summary:.....	15
CHAPTER THREE: COMMON P2P SECURITY ISSUES.....	16
3.1 Validity of Queried Files:	16
3.2 Verifying Transferred Files:	18
3.3 Malicious Users:	19
3.4 Intrusion Detection:	22
CHAPTER FOUR: RELATED WORK.....	24
CHAPTER FIVE: SARP NET.....	27
5.1 Topology:.....	27
5.1.1 Node Roles:.....	27

5.1.2 Group Formation:.....	32
5.1.3 Node Limitations:	35
5.2 Key Distribution:	37
5.3 Routing Protocol:	38
5.4 Fragment Caching:.....	46
5.5 Group-Based Reputation Scheme:	47
5.6 Addressed Security Issues:	50
5.7 Prototype:.....	52
CHAPTER SIX: DATA ANALYSIS.....	56
6.1 Simulator Overview:.....	56
6.2 Experiment 1:.....	58
6.2.1 Results:.....	60
6.3 Experiment 2:.....	64
6.4 Experiment 3:.....	65
6.5 Discussion:.....	66
CHAPTER SEVEN: CONCLUSION.....	69
7.1 Conclusion:	69
7.2 Future Work:.....	70
APPENDIX: EXPERIMENT RESULTS	71
LIST OF REFERENCES	74

LIST OF FIGURES

Figure 1: Client-Sever Architecture.....	4
Figure 2: Decentralized P2P Architecture	6
Figure 3: SARP Net’s Node Roles.....	31
Figure 4: The SARP Net Topology	35
Figure 5: SARP Net Prototype: File Query Web Results.....	54
Figure 6: Utilized Bandwidth Percentage vs. Time for Experiment 1.....	62
Figure 7: Percentage of Bandwidth Utilization for Experiment 1	63
Figure 8: Reputations Created and Malicious Node Detection vs. Time (Experiment 1)	64
Figure 9: Average File Transfer Speed.....	68
Figure 10: Bandwidth Utilization of SARP Net vs. Freenet.....	68

LIST OF TABLES

Table 1: Simulation Results Averaged Over 10 Runs of Simulator for Experiment 1.....	71
Table 2: Simulation Results Averaged Over 10 Runs of Simulator for Experiment 2.....	72
Table 3: Simulation Results Averaged Over 10 Runs of Simulator for Experiment 3.....	73

LIST OF ACRONYMS/ABBREVIATIONS

DDOS	Distributed Denial-of-Service
DOS	Denial-of-Service
FID	Forward Identification
FRID	File Request Identifier
FTID	File Transfer Identifier
GL	Group Leader
HTL	Hops-to-Live
ID	Identifier/ Identification
IRID	Interaction Request Identifier
LL	Leader-of-Leader
NODE	Any computer connected to the peer-to-peer network.
PEER	Any computer connected to the peer-to-peer network.
P2P	Peer-to-Peer
QID	Query Identifier
RID	Reputation Identifier
SSL	Secure Socket Layer

CHAPTER ONE: INTRODUCTION

Since the advent of Napster, the idea of peer-to-peer (P2P) architectures being applied to file-sharing applications has become popular, spawning other commercial peer-to-peers like Gnutella, Grokster, Morpheus, Kazaa, eDonkey, and BitTorrent. This growth in P2P development has nearly eradicated the idea of the traditional client-server structure in the file sharing model, placing emphasizes on distributed and faster query processing, deeper levels of decentralism, and methods to protect user identities to avoid copy-right violation prosecution.

The gain in popularity of the P2P paradigm has attracted all walks of life to attempt to benefit from the potential power of any P2P system. In many cases, P2P networks provide users with a collection of computer resources to exploit and use for personal benefit. In a P2P network designed to distribute workload to multiple computers, the workload can be accomplished faster and less resource-expensive than using a single computer or a multiprocessor machine. The most well-known example of this is the SETI@Home project that distributes mathematical calculations to its peers for research [30].

Security-wise, P2P systems run the risk of exposing their users to several security threats. Common attacks and malicious activities performed on nodes include distributed denial-of-service (DDOS) attacks, poisoning attacks, IP harvesting, flooding, data relay refusal, malware distribution, and leeching (nodes refusing to share data but are willing to query and download data) [32]. These threats run the risk of seriously compromising networks, damaging a system's safety credibility, and spreading malicious content such as viruses and worms.

SARP Net, the proposed model, is the structure of a completely decentralized file-sharing system that incorporates heavy security mechanisms while supporting efficient queries,

fault tolerance, and other features desired by typical file-sharing applications. Its main goal is to help maintain a high level of security for its users while protecting their identities through message hopping.

The main contributions of this work include the model's reputation scheme and the system's open design. The reputation scheme described in this work builds off of the idea that friend-to-friend networks help establish an automatic web-of-trust that can be utilized to filter out malicious nodes. The system's robust open design allows many areas of the protocol and node functionality to be extended to support a large number of different distributed computing features. This takes away the limitation that this system is only designed for file-sharing. The system can easily be customized to help distribute streaming multimedia, web browsing, file storage, document publication, and malware alerts.

SARP Net is described in detail in the following chapters. First, a background about P2P systems is provided, followed by a survey of the most prevalent P2P security issues and related work. Finally, SARP Net is discussed, accompanied with the experiments, results, conclusion, and future work.

CHAPTER TWO: BACKGROUND

Since the advent of Napster in 1999, Peer-to-Peer (P2P) technology has received overwhelming levels of interest [16]. This spark in attention has influenced a technological revolution from centralized servers dominating socket-connected systems to the use of a decentralized flow of information. P2P protocols are now being employed in many different facets of information sharing across the internet-medium. Such uses include file sharing, instant messaging systems, streaming multimedia, and distributed computing.

The definitive quality that any “true” P2P system possesses is the absence of a centralized server [9]. All nodes in a true P2P network have the ability to portray any role and possess any privilege the network offers. Typically, in the client-server model, servers and hosts have higher privileges than the clients they serve. In the place of this centralized host, decentralized networks have intricate architectures of computers that relay data among each other. Since the main interest of any P2P network is to distribute data and workload in an efficient manner, most of the major innovations in P2P file-sharing design have focused on file distribution speed and query performance.

The following sections provide an understanding of the different types of P2P networks, their strengths and weaknesses, centralized versus decentralized P2P networks, and potential P2P problems and solutions.

2.1 Centralized Servers:

By definition, a centralized network is a system that possesses a centralized server who relays all queries and responses to other entities in the system. These main hosts and servers are

given the highest level of privilege and ultimately controls all interactions between computers connected to the network [20]. This structure is most commonly known as a client-server architecture where clients are the computers who make queries, requests, and file transfers in the system. Servers are the host computers who accept these queries and requests from clients and relay the data to other nodes. Figure 1 depicts how in a small and simple client-server network one server could be responsible for all requests from its clients.

The client-server architecture tends to be the most basic design of internet communication between two or more computers where clients are initializing and executing messages and servers are relaying them. This holds true in the realm of P2P networking. In the case of file-sharing P2P networking, a centralized server accepts all client connections and relays commands initializing file searches, file transfers, chat messages, and other network requests to all or a subset of connected clients.

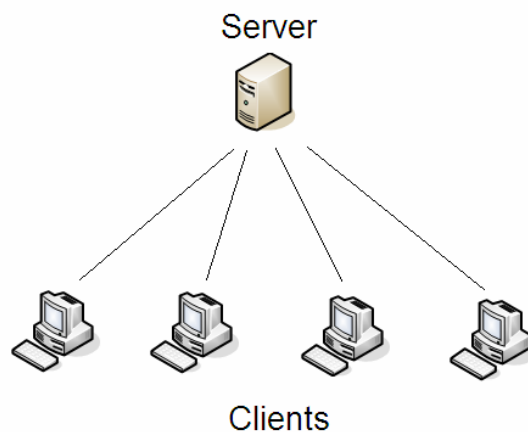


Figure 1: Client-Sever Architecture

Napster is the most well-known centralized file-sharing network to date. Even though its architecture has a few complex components and has been modified over the years, the original 1999 Napster uses all of the basics of a centralized P2P system.

Each Napster client connects to multiple servers through a single session on the network. When a client attempts to log into the Napster network, the client first connects to a main server. This main server accepts or rejects a client's personal information, including a username, password, email address, and the number of files that the client will share. If the client was accepted, a directory server receives a listing of all the files the client will share. The directory server handles all queries for file searches. When the client wishes to perform a file search the client would send a message with the search criteria to the directory server and wait for a response. If the server has a listing of that file in its shared files look-up table, it will reply with the file's basic information such as the file's source, file name, and file size. To download a file the client establishes a connection to the file source who is already listening to a designated port. Next, the requesting node notifies the file source of which file is to be transferred, and depending if authorization is granted, the file will then be transmitted through that connection. More detailed information can be found at [2] or [3] about Napster.

The rate at which a query result is returned to a client is extremely fast in centralized networks in comparison to other designs. In Napster's case, since the directory server has a listing of all files and host addresses, only one computer has to execute a search. Other many other centralized designs distribute the search workload at the cost of forwarding query messages. Here, once a server receives a query message, it will relay the message to a subset of its clients where each recipient will search for the file on its own and send the results of the query back to the server. The server will then gather all of the results and relay them back to the requesting client. Even though this is not as fast as Napster's implementation, this method saves the centralized server from performing a lot of searches and requiring clients to upload their shared file listings at the cost of relaying query messages.

The centralized genre of P2P architecture comes with a couple of noteworthy problems. The first is the need for fast, reliable servers. These servers have to possess enough bandwidth and resources to handle a magnitude of incoming and outgoing data at once. The major disadvantage of centralized P2P networks is its dependency on a small number of computers. If a server goes down due to power failure, security breaches, or network activity overload, the entire system can halt and connections will be dropped. With this flaw, centralized systems suffer from a lack of fault tolerance.

2.2 Decentralized Servers:

A decentralized network is a system that distributes the hosting responsibilities among other computers [4]. This absence of critical central servers creates a need for a scheme to handle all connections and queries in a way that does not place a large workload on any single network entity yet guaranteeing a sense of operational reliability. Figure 2 displays a decentralized formation, where clients share responsibilities in order to process network data, make requests, and maintain the topology of the network. Notice the lack of a central host.

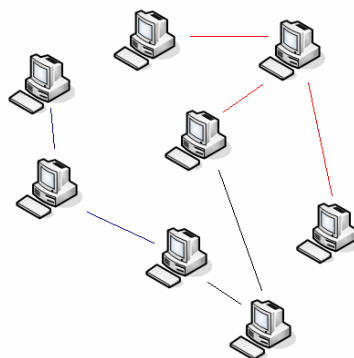


Figure 2: Decentralized P2P Architecture

In a decentralized design, several issues arise when developing a working scheme. The biggest issue is to assign incoming connections to already-connected peers. This requires a connection between an incoming client and a server before the client can connect to other peers. In the connection, the server will assign the client one or more addresses to connect to in order to join the network. Typically, this assignment will initiate a connection to other peers or one of many servers in the already established system. The problem that this bootstrap server causes is that it automatically forces any decentralized system to have a centralized point since all clients must primarily connect to a single server whose address never changes and is known by all clients. The compromise this issue has spawned deals with logic: since nodes that are new to the network must receive information about who to connect to, nodes must know at least one other computer to connect to. Without some sort of bootstrapping information, a node can never join the system so a bootstrap server is needed.

Other concerns with a decentralized architecture are their handling of security and reliability issues. Since all decentralized networks have either multiple peer-servers or force queries to go through several chains of peers, message interception and system vulnerabilities can easily be exploited. Messages being relayed from one peer to another can always be captured by intermediate connections. Even though most P2P networks pass encrypted messages between peers this security technique combats threats from attacks outside of the system. Meaning that messages containing query information must be decrypted and known by all reached clients, malicious intermediate peers can easily extract file search strings, query results, and IP addresses of the requesting client and peers who satisfy the requests if a protection system is not implemented.

Typically, decentralized systems perform queries slower than its centralized counterpart. In truly decentralized systems, query messages must traverse a web of peers before propagating back to the requesting client. In comparison to a centralized design, where one message is sent to a single server and the server sends the result back to the requesting computer, it is easy to see how the speed differs between the two different schemes [4]. When multiple peer-servers are used in a decentralized system, the query speed can be similar to Napster's centralized design if each peer-server acts in the same manner as Napster's directory server.

2.3 Popular File-Sharing P2Ps:

There are many different techniques to design a decentralized system. Some of the most innovative implementations have been designed by commercial institutions which have spawned P2P file sharing applications such as Gnutella, Morpheus, KaZaa, BitTorrent, and E-Mule to name a few. Each system provides a unique design of a file-sharing network with features that focus on query and file transfer speed.

2.3.1 Gnutella:

The Gnutella protocol is among the most popular P2P designs to date. It was the first mainstream file-sharing protocol to use a decentralized server in 2000. The first two generations of the Gnutella network have influenced the P2P file-sharing community immensely. From 2000 to 2002 the first generation (Gnutella versions 0.1-0.6) was heavily decentralized and seemed to be the best implementation of a "true" P2P system with the existence of bootstrap servers. In 2002, Gnutella 2 was developed. This second generation recognized the main problems faced in

the first Gnutella versions and sought to correct them by stepping away from decentralization in its design to handle peer connections. Both generations will now be discussed, describing the three main issues both architectures addresses: how peers connect to the network, how disconnections are handled, and how queries are performed. Complete descriptions of Gnutella can be found at [12] and [31].

The best way to describe the first generation of Gnutella is as a complex network of peers who pass messages to each other without an intermediary. The creators of Gnutella gave each peer the title of “servent” which denotes a merger between the typical network roles of server and client. This means that each Gnutella node acts as both a client and server to other peers, handling message relays and query operations.

When a client wants to join the Gnutella network, the client must obtain a list of already connected servents. This list can be acquired by connecting to a bootstrap server whose sole purpose is to collect IP addresses of servents and relay all or a subset of them to connecting computers. Once a computer has a list of servents, it will attempt to connect to a subset of them. Any servents addresses that are not accepting connections will be discarded.

Queries are handled in a web based manner for this version of Gnutella. To initialize a file query, a servent will send a query message to all of the servents it is connected to. This query message will contain the query command tag, the text string to search for, the IP address of the requesting servent, and the search depth. Each servent will perform the query and forward the message to the other servents it is connected to. This step is repeated until the search depth has been reached. If one of the servents has the requested file then it will send a message with the file information directly to requesting servent by establishing a connection if one does not already exist. If the requesting servent wants to download a file specified in the query results then a

direct connection is established between one or more hosts. When multiple hosts are used “swarming” is performed. Swarming is a technique which breaks down a target file into fragments and several hosts upload different fragments to the requesting server. When all of the fragments have been transferred, the downloading computer combines each fragment into a single file and validates it. Validation takes place on two occasions: before any file transfers take place when several hosts are used to transfer one file and when the transfer is complete. Hash functions are used in this form of validation by comparing hash values on each file host that is used for a transfer and then to compare the transmitted file with the one from the source.

With the use of a bootstrap server to distribute servers lists, Gnutella seems to have a centralized server. On the contrary, this bootstrap server can easily be removed without jeopardizing the connectivity of the network’s active nodes since its protocol allows for users to specify which servers to connect to. Therefore, as long as there are at least two servers connected, a Gnutella network will exist. This takes away the dependency on a single computer to always be connected like Napster and other centralized networks.

The main drawback to the Gnutella network is the speed for queries. Propagation of the network can take large amounts of time for a single query to reach the specified search depth of the system. Furthermore, the direct connections that are created from these queries take additional time and slow the rate for a query result to be returned. To handle this rate issue Gnutella 2 was created in 2002. Going on the fact that centralized servers tend to perform queries more efficiently than their decentralized cousins, the developers of Gnutella 2 incorporated a design with qualities from both architectures.

Gnutella 2 breaks away from the use of servers and separates peers by assigning “leaf” and “hub” roles. Leaves are peers who initiate connections to one or two hubs at any given time.

A hub is a peer who accepts large amounts of leaf connections, collect file listings from all connected leaves, and performs file searches for queries. This method takes away from the network congestion the first generation of Gnutella has caused while guaranteeing that any file on the network is reachable. Gnutella 2 also incorporates a few notable additions such as data compression, the use of UDP, and a file preview function.

To decrease the bandwidth used for file transfers, the second generation of Gnutella uses a compression scheme to attempt to minimize the file size for transmission. With compression, faster transfers can take place creating more sources for shared files. Finally, a preview function was added to this version of Gnutella. The preview of audio and video allows users to test a file before a transfer is complete. File preview combats time wasting by the downloading peer for receiving preview-able dummy files. In addition to this, file ratings have been installed which allow users to comment on a transferred file. Ratings help notify future downloading nodes if a file is corrupt, malware, poor quality, etc. New updates are constantly being made to the Gnutella network to improve performance and security. These two early generations provided a foundation for decentralized file-sharing.

2.3.2 KaZaa and FastTrack:

KaZaa is another P2P file sharing network that became popular with the downfall of Napster. The KaZaa software uses the FastTrack file-sharing protocol to connect thousands of computers to its single network. The FastTrack architecture can be seen as an extension to the Gnutella protocol since it takes Gnutella's basics fundamentals and produces a product whose design is very similar to Gnutella 2.

Instead of labeling peers as Gnutella's "hubs" and "leaves", FastTrack assigns "node" and "supernode" roles to its incoming connections [4]. In much respects, supernodes perform the same tasks as hubs and the same goes for nodes and leaves. Throughout the KaZaa network there are many supernodes who each accept many connections from nodes. A computer connecting to the network is given supernode status if and only if it is a fast computer with excellent bandwidth and a fast connection. This selected few handles message passing between nodes which include chat messages, query requests, and other network commands.

Each KaZaa application comes with a list of supernodes pre-installed. When a computer wishes to join the network it will go through the list until a connection can be made to a supernode. The incoming computer will then be given a list of supernodes to be used the next time the computer wishes to enter the network and to select which supernode the incoming computer wishes to use as its designated supernode for the current session. Once a supernode is chosen, a connection is made and the incoming node will transmit a list of files it will share. If the incoming computer is a supernode then it will connect to several other supernodes, broadcast its supernode status, and await nodes to establish a connection to it.

When a user initiates a file search query, a message containing the search string is sent to a supernode. The supernode will search through its listing of all files shared by the nodes it oversees and reply with any matches. The supernode will also forward the query message to other supernodes to broaden the search. When a file is located, a direct connection to the node who initiated the query is established if one does not already exist. Once the connection is made, the file information and the IP address of the file host are sent to the query author. When the query author wants to download a file, a direct connection is made between it and the file hosts, resulting in the file being fragmented among hosts and transmitted.

One interesting addition KaZaa brought to the file-sharing development community is the use of “participation level”. The amount of bytes uploaded by a peer determines how many query results will be returned to it and the downloading priority level it will possess. The use of the participation level was implemented to thwart peers from only downloading and never sharing (uploading) any files on the network.

2.3.2 BitTorrent:

BitTorrent [6] can be classified as thousands of directly connected node groups, each sharing short-lived and sharing commonly interested files. In the most basic of BitTorrent implementations, there is an absence of a centralized server to initialize connections or to process queries. To handle file requests, a .torrent file is used which contains the IP address of the file’s tracker server and information about the targeted file. File information would include the file name, file offset and length, and hashes for each portion of the fragmented file. The tracker server is a computer who maintains a list of other computers who possess some percentage of the requesting file. When the requesting computer connects to the tracker, the list of uploaders will be transmitted along with which portions they have. A connection is made to all or a subset of the uploaders and specified blocks are transmitted to the requesting computer. The tracker is constantly updated with the state of sharing nodes and which file fragments they share.

To initiate any BitTorrent file transfer, the .torrent file must be located. Torrent files can be placed on websites, bulletin boards, newsgroups, etc to point potential downloaders to where that the file is available. This .torrent file is created when a user wishes to share a file. It acts like an announcement that the file is available. Once a user downloads a .torrent file and opens it in a

BitTorrent client application, it will connect to the tracker specified and begin the process of locating uploaders to begin the download. All users are given the option of running their own tracker server to their shared files. This makes BitTorrent act as a large number of small file-sharing networks.

The most interesting feature of BitTorrent is its method for increasing the availability of a file. When a user has downloaded a portion of a file, immediately that portion can be shared to other users. This increases the number of hosts and speed of transfers since files are being downloaded from several sources. To guarantee that correct blocks are being transferred, the hash values from the shared file are used to compare the portions to be transmitted. If the hash values are not the same, a transfer will not occur, saving the downloader the time of downloading a corrupt or wrong block and an uploader from having to waste bandwidth.

Another key feature BitTorrent has is its ability to not clog the network when a rare file is being requested by a lot of users. When a bottleneck forms in a P2P system, a lot of strain can be placed on a single sharer. BitTorrent handles this problem by uploading different blocks of a file to different users when a file is rare. When several blocks are spread out among users, requesting users can download missing blocks from other users, not burdening one computer with many requests. This scheme, with the addition of downloading blocks out of order, speeds up file transfers of rare files which seems to be a problem with other P2P systems.

The advantage for Bit Torrent is its handling of rare files and its use of swarming. Also, the protocol uses a participation monitor that depends on how much a user has uploaded. If there are multiple users requesting a file, priority will be given to those who have higher participation levels, resulting in slow downloads for those who do not share.

2.4 Classification Summary:

In short, the most influential P2P protocols all differ in one significant way or another. Napster was a centralized file-sharing system who supported fast queries but lacked the notion of using several hosts to download a single file and fault tolerance. The first generation of Gnutella created a unique way of connecting thousands of computers without the need of a centralized server and implemented one of the first mainstream uses of swarming. Gnutella 2 was an evolution of the first generation that went toward a more centralized approach by introducing the assignment of hub and leaf roles to its peers but produced faster query results than its predecessor. KaZaa has many similarities to Gnutella 2. Using the FastTrack protocol, Kazaa does not need a centralized server due to its dependence on supernode listings to always be updated. BitTorrent is a protocol that differs immensely from the other P2P systems mentioned. Instead of creating one massive network of peers, BitTorrent thrives on the creation of .torrent files to create very small computer groups where all connected users share an interest in the same file. BitTorrent's handling of rare files is unlike other file-sharing schemes due to its handling of scarce fragments.

CHAPTER THREE: COMMON P2P SECURITY ISSUES

With the popularity of many commercial P2P file sharing applications, such as Napster and Kazaa, security has become an aspect of P2P development that has begun to receive a lot of attention. Until recently, security has not been a main focus for many P2P development schemes. Alternatively, query speed and fault tolerance approaches monopolized system design in hopes to be a beneficial tradeoff from secure systems. With in the last few years, P2P schemes, such as Indra [13] have placed major emphasis on security for distributed environments. In this section, the major security issues P2P systems face will be discussed.

3.1 Validity of Queried Files:

One security feature file sharing systems must incorporate is a strong mechanism to allow users to verify the files to be downloaded are the intended files from their initial query. The need for this feature is that most of the popular file-sharing applications suffer from dummy files being introduced to their shared file libraries. These dummy files are files malicious users share on the network in hopes of tricking an unsuspecting user to download them. At first glance, these files seem to be the exact files querying users are looking for since they may have file lengths that seem appropriate, IDE tags that match queried music or movie information, or in the most basic of cases, have what seems to be a valid file name. Once a dummy file transfer is complete and the user realizes the file is not the desired one, time, bandwidth, and possibly money is wasted.

Several preexisting file sharing programs use file previewing to combat this issue. Software like LimeWire and Morpheus allow their users to preview partially downloaded files

during the file transfer. For example, if an mp3 file is being downloaded and 1 megabyte (MB) of the total 3 MB has been transferred, the user can preview the 1MB of audio to determine if the transfer should continue or to kill the download. Although beneficial for downloaders, this approach does not remedy the problem of dummy, corrupt, or undesired file transfers entirely. One problem with this form of previewing approach is that some file formats, such as DivX video [7], cannot be previewed until the entire file has been downloaded. In many cases, large portions of the file must be downloaded in order to preview. By the time a user is able to use the preview function, large amounts of time has passed and large amounts of data have been transferred. A second problem occurs when dummy files contain looped versions of the queried audio or video. The problem here lies when the user previews the transferred portion of the file and does not detect the loop. The file could continue to be allowed to be transferred so when the download is complete, the user is left with a dummy file. Once again, time and bandwidth, among other things, are wasted.

Other existing solutions to dummy files are the employment of peer and file reviews. Peer reviews allow users of P2P systems to rate each other on the quality of the files they share and transfer. Once a file has been transferred, the downloader can evaluate the newly acquired file and write a review (normally a rating 1-5) about the uploader and send it to the host or server computer for storage. A similar manner was used for file reviews, where the reviews of the files were sent to the hosts and also stored on the downloader's computer. These rating/review systems allowed users to use the opinions of others before downloading a file by retrieving reviews when query results are returned, making these P2P systems integrity based.

A few problems arise from this solution. The first problem is the mandatory storage of additional information. Each review must be stored on some computer for easy retrieval. If the

hosts involved were Gnutella servers and could disconnect from the system at any point, backup methods would have to be employed to guarantee the reviews and ratings were long-lived. A second problem is that these reviews and ratings were more data being returned during queries. This ultimately would slow down query results from being returned to the users, hurting one of the most important goals of any P2P file system (fast query results). In addition, these integrity based systems must deal with malicious users. This opens the risk of users returning false ratings or biased reviews in hopes of discrediting or promoting a user or a file.

3.2 Verifying Transferred Files:

Another security issue that file sharing systems must handle is to verify that a downloaded file is an exact copy of the file selected to download from the initial query results. One cause of this problem is the ability of uploaders to modify the file contents of files to be uploaded while the files are being transferred. This problem occurs when file sharing programs allow users to pause or temporarily suspend transfers.

One solution is to use hash values of files before and after every file transfer. A hash function, using the files to be transferred as input, is used to generate these hash values. Once a file has been transferred, the downloader can take the hash value of the downloaded file and compare it to the hash value of the file stored on the uploader's computer. One common issue with file verification using hash values is when transfer pausing is allowed. Here, the uploader must send the hash value of file content that was transmitted to the downloader for verification. If the hash values differ, then the downloader should reject that uploader as a host.

A second use of hash functions in file sharing applications is when a user can download a single file from multiple sources. In this use, hash values of returned files from queries can be compared to determine if multiple sources of the same file exist.

Cryptographic hash functions, like the MD5 algorithm, with a 128-bit hash value, are widely used for file verification [21]. The popular P2P application eDonkey uses the MD4 algorithm in its early versions to verify file partitions[15].

3.3 Malicious Users:

The very real possibility of individuals and groups of individuals attempting to harm a P2P system poses a real threat to the health, existence, and security of any P2P community as evident in [28]. Since the very base of P2P per networks are their peers, attacks can originate from anywhere within the network. Attack motives include but are not limited to clogging the system, obtaining IP lists of users, using the systems resources, spread spam messages, and infect users with worms, viruses, trojans, spyware, and other forms of malware [27].

One method previously discussed involved poisoning attacks, which are when malicious users share dummy files which differ from their descriptions [22]. This attack's main purpose is to dupe users into wasting time and bandwidth. More sinister poisoning attacks include dummy files with malware. With the spread of viruses in a file sharing system, sections of a P2P network can easily be disabled from attack, resulting in computer damage at individual nodes, unavailability of rare or precious files, and in systems with the group hierarchies, difficulty for groups to thrive [27].

Current solutions to malware attacks involve the users making file reviews, scanning files with anti-virus software, and using intrusion detection systems (IDS) such as NetBiotic [27] to detect malicious activity. Schemes like NetBiotic allow malware to be detected at local levels and then broadcast alerts other connected computers about the threat. Depending on the severity of the threat, IDS's can choose what actions need to take place in order to neutralize the attack. Other IDSs will be discussed later to handle other forms of attacks.

Two other attacks P2P systems must be prepared for are Denial of Service (DOS) and Distributed Denial of Service (DDOS) attacks [27]. DOS attacks are when malicious users send large number of packets of data to a computer with the intent to slow the computer down, or in extreme cases, knock the victim computer offline [28]. In coordinated attacks, DDOS attacks involve multiple computers attacking a computer system. In P2P systems, servers-like nodes or hosts are the more vulnerable computers to these attacks. If a host is victim to a DOS or DDOS attack, all of its connected users can suffer from slow query responses, not be able to query the adequate portions of the network, and may lose the ability to connect to peers outside of its group or neighbors. If a host is disconnected from a DOS attack, all of its clients must connect to another host to interact with the system. The coordination of DDOS attacks does not require all of the attackers to be malicious. A prime example of this involves DDOS originated by one computer that trick non-malicious users into sending a targeted computer data or requests for data by falsifying query results [32].

An attack related to DOS attacks is flooding. Flooding is an attack that consists of malicious users who are connected to the system sending large amounts of queries and other system commands throughout the network. In many cases, these attacks send a flood of messages to a node with the intention to do the same damage as a typical DOS attack. In some

regards, flooding can cause more problems than DOS attacks, depending on the architecture of the network. For example, if queries are forwarded from server to server, as in Gnutella, each visited server can see the same attack as the original server. To combat this possible vulnerability, checks must be made to guarantee that each node is not allowed to send a large enough amount of commands, queries, or data in a given time to potentially harm the system. Also, during direct connections, peer applications should also limit the amount of communication being transmitted between peers for a given interval as an extra precaution.

IP harvesting is another security threat that must be attended to for P2P systems. IP harvesting is the gathering of user IP addresses with the intent to either perform some form of computer attack or to use the address to trace the personal identity of peer [32]. With an IP address, a user's name, phone number, and address can be obtained if divulged by an ISP. This information can be obtained by groups like the Recording Industry Association of America (RIAA) and Motion Picture Association of America (MPAA) with court orders or the threat of suing an ISP's over copyright violations. These groups constantly search file sharing networks for copyright law violators and in many cases, take legal action. In order to guarantee user privacy, protocols like Friend-to-Friend (F2F) and anonymous P2P systems have been created. F2F systems are P2P networks that base connections on trusted peers and strong levels of encryption. Here, peers who know each other ("friends") can only share communication and data with each other, friends of friends, friends of friends of friends, etc. Users of F2F networks can only share data with people in a web-of-trust, where every member can be traced to a direct friend. The most popular of F2F systems is WASTE, which was developed at NULLSOFT in 2003 by Justin Frankel [29].

Anonymous P2P protocols attempt to hide node identities behind a cloud of deniability [5]. This way, it is difficult to locate the original source of any request on the system. There are several anonymous P2P protocols in existence; among the most popular are Freenet [5] and the BitTorrent client Azureus [1].

3.4 Intrusion Detection:

In P2P systems, the act of intrusion can occur in a lot of different places. Two of most important areas of intrusion are at the network level and at system privileges. At the network level, intrusion is present when unauthorized persons attempt to gain access to the system [13]. This includes being able to connect to peers, make queries, and perform other system features authorize peers are capable to perform. An example would be of malicious users attempting to gain access to a password protected system with authorization. In P2P systems that allow users to set privileges on network items, such as files, groups, bandwidth speeds, and peer-communication, potential intrusion needs to be anticipated.

The occurrence of users that successfully access features that they do not have authorized privileges for is a major security problem since it threatens confidentiality of peers and reliability of the system. The need of intrusion detection mechanisms is essential for any secure P2P system. Many existing intrusion detection and prevention schemes, like Indra [13], target passive and aggressive attacks that try to break in or disrupt a network. Such schemes attempt to recognize common areas of attack, such as port scanning, multiple failed logins, and possible DOS and DDOS attacks. Similar to how NetBiotic [27] forwards warning messages from nodes that were victims of attack to other nodes on the network, “neighborhood watch” and P2P

communication has become a good mechanism for detecting intrusion attempts and preventing the spread [13].

Sybil attacks are a security issue that many P2P networks must address. Sybil attacks mostly target the reputation systems of networks that employ such a scheme by attempting to make them meaningless. This form of attack occurs when users are permitted to create multiple identities on a network. With these multiple identities, users can perform malicious tasks on some of their identities and behave correctly on others. This allows malicious users to always have access to the privileges that well behaved nodes have and disrupt the network at the same time [8]. As one can imagine, this is a problem for reputation schemes attempting to punish malicious nodes and phrase behaving ones.

CHAPTER FOUR: RELATED WORK

Indra [13] is an intrusion detection and prevention scheme that is based on the P2P architecture. It is powerful in the fact that once there is an attack at a single node level, messages are sent out through the network, notifying other nodes of the attack. The protection starts with each host on the network running the Indra daemon. The daemon watches for common intrusion attempts and ones based on previous attempts. Since hosts rely on the validity of security messages, a Web of Trust is used, where all nodes are connected by trust relationships. This assists in preventing nodes from reporting false intrusion attempts and disrupting the system. In addition, efficient group communication is essential for the distribution of intrusion information. The efficient group communication helps guarantee that messages are sent and received. The basic architecture of Indra depends on four separate roles: Watchers, Access Controllers, Listeners, and Reporters. Watchers look for suspicious activity, either locally or over the network. Access Controllers provide controlled access to the resources. Listeners listen to the watchers and transmit warnings to the Access Controllers. Reporters communicate with other hosts by receiving and forwarding warnings on intrusion attempts. A few of the common intrusion threats Indra monitors for include multiple failed logins, port scan attempts, and suspicious system calls [13].

Vlachos et al's NetBiotic [27] is an application that uses a collection of several computers in a group of peers to exchange messages with information about attacks. The difference between NetBiotic and Indra is that the former takes into account more local attacks, such as viruses, and determines how to handle the threat across the network. The NetBiotic application

has two roles: Notifiers and Handlers. Notifiers are daemons responsible for monitoring the local node on which it is running to gather information on possible security attacks. This daemon monitors security applications like firewalls, anti-virus software, and IDS's for suspicious events. Handlers are the NetBiotic daemon that receives messages sent from the Notifiers of other nodes and performs actions based on those messages to protect that node and the network [27].

Another category of secure P2P systems attempt to guarantee the anonymity of all users at the cost of increased bandwidth and query speeds. One such system is Freenet [5]. Freenet is decentralized data store which heavily relies on its key-based routing system to protect queries and file retrieval. In essence, Freenet is a network of nodes communicating with one another through the use of encrypted messages. In this system, no broadcast searches or centralized servers every have to be used to relay data. Similar to the Gnutella protocol, Freenet nodes connect to several other nodes running the Freenet protocol to establish a network of neighbors. When a node wishes to query for a file, encrypted messages, containing the key of the target file, are passed from neighbor to neighbor. When a neighbor receives a query command, it will search its local data store to detect if portions of the file are stored locally or if the location is known on the network, and return the results to the neighbor who passed the query message. If the location is unknown at an individual node level, the query message is passed to the neighbors of the current node. If the file location is not known after reaching a predefined search depth, messages are returned in reverse order, notifying neighbors a search branch has not located the file. The use of relaying messages from node to node guarantees the anonymity of the query author. The guarantee is made since nodes can always claim to be relaying another node's query. To further guarantee the anonymity of users, especially producers and consumers of the files being shared,

Freenet encrypts and fragments shared files, to distribute the content across the network. The distribution of encrypted file fragments prevent nodes from knowing what files they are sharing but the key-based routing system allows querying nodes to locate desired files. The goal here is that even if a consumer knows the identity of the producer, the producer cannot be held accountable for the file transmission since it cannot have any idea of what file it is sharing. Another anonymous P2P system is GNUnet [11]. Similar to Freenet, it also is decentralized and claims to be censorship-resistant.

Another genre of secure P2P networks uses the Friend-to-Friend (F2F) approach to protect users. The main idea of F2F networks is that nodes only connect to nodes either they can trust or to nodes a friend of someone in a friend-of-a-friend web can trust. Basically, files are only shared and queries are only made within a web-of-trust where each node can be traced to another node through friends associations. There are two major disadvantages arise in F2F networks. One, the existence of malicious friendly nodes can completely disrupt a F2F network. Two, files outside of the friend web are unreachable. The most well known F2F network is WASTE [29].

CHAPTER FIVE: SARP NET

SARP Net is a P2P overlay network that is designed to protect the activity of its users. It is secure in the fact that public-key encryption is used to guard network messages from prying eyes internal and external of the network. The protocol guarantees user anonymity by incorporating message-hopping from node to node to prevent any network observer from pinpointing the origin of any query or shared-file source. To further enhance the system's security, a reputation scheme is incorporated to police nodes from malicious activity, maintain the overlay's topology, and enforce rules to protect node identity. The following sections describe in detail the design of SARP Net.

5.1 Topology:

SARP Net is based on a three tier hierarchy where each level denotes a unique role to the system. Each role is used to maintain the formation of node groups to process queries, file transfers, and distribute reputations.

5.1.1 Node Roles:

User-node: The bottom tier is comprised of user-nodes. User-nodes are the most basic type of node in the topology, being the most common role throughout the network. This type of user has the privileges making, responding to and forwarding queries, query and return interaction receipts, transfer files, and post reputations based on interactions of other connected nodes. User-nodes cannot establish connections to other nodes outside of their group for any reason than to

process file transfers. Connections to all or a subset of nodes within the same group are permitted to exchange messages directly. Even though the user-node role is the lowest leveled, user-nodes are the driving force of each group since they produce the most queries, query responses, file transfers, and reputations.

Group-Leaders: The middle tiered role is that of group-leaders. Group-leaders represent the current hosts for a group of which user-nodes connect to in order to join the network. In effect, group-leaders connect each group member to other groups by establishing links to leader-of-leader nodes. Nodes of this role govern the group by monitoring the activities of their user-nodes by making judgment based on reputation and alert messages from nodes internal and external to the group. If a user-node is determined to be malicious, the set of group-leaders have the privilege of isolating a node from the group or restrict the node's privileges to make the group safe. Group-leaders divide their resources (bandwidth and processing power) between providing the services user-nodes offer, forwarding interaction receipts, resolving interaction disputes, validating incoming users, and maintaining the formation of the group. A user-node has the opportunity to be promoted to group-leader if the user-node excels more than the other user-nodes when an appointment of a new group-leader is needed. Role promotion and demotion will be discussed later.

Group: A group is not a role a node can obtain but can be seen as a virtual container for user-nodes and group-nodes who are connected by references and associations. Each group is made up of a number of user-nodes and a fraction of as many group-leaders to govern them. The number of group-leaders is proportional to the number of user-nodes currently online. For

example, a group rule can be set for every 10 user-nodes one group-leader is assigned. Group-leaders of a group are interconnected to allow free flowing messages to be exchanged. Group-leaders of a group must share the same group-state and update other group-leaders if a state is altered. The establishment of groups allows subsets of nodes in the network to be governed: punishing malicious nodes, rewarding benevolent nodes, maintain the livelihood of a group, keeping group-members updated, accepting new members from node references, and verifying member identity when existing members join the network.

Leader-of-Leader: The top tier of the hierarchy is the leader-of-leader (LL) role. LL's connect groups to other groups by acting as a go-between by forwarding messages from one group's group-leader to a different group's group-leader. The role of LL has two major jobs: forward messages from one group to all of the other connected groups and to punish groups who repeatedly allow their malicious user-nodes to hurt the integrity of the system. LL's act very similar to group-leaders except they are responsible for group-leader behavior. The activities of LL's are checked by other LL's of the same leader-group and group-leaders in their range of connection. LL's are promoted from the group-leaders who have fast connections, display superb reputation ratings, and have managed their groups properly. LL's divide their resources by forwarding message between groups and other LL's and monitoring group behavior.

Leader-Group: Leader-groups differ from groups of user-nodes and group-leaders by being comprised of LL's. Each leader-group has a proportion of LL's to groups. For example, if there is a 10:1 ratio of LL's to groups of a leader-group of 30 groups, three LL's will be interconnected to manage the groups. Each leader-group is connected to other leader-groups by

the connections their LL's establish. Since it is not feasible for each leader-group to be connected to all leader-groups in the network, a leader-group is connected to a subset of other leader-groups. With this setup, each LL of a leader-group is connected to the same leaders of other leader-groups.

Bootstrap Server (optional): A bootstrap server is proposed to act as a reference for nodes that do not have another entry point into the network. The bootstrap server maintains information on each group and IP addresses and port of online LL's and group-leaders. When a group is formed, its identification is sent to the bootstrap server as well as the group's public-private key pair. This key pair is used to verify that user querying for an IP of a group-leader is an actual member. When the query is made, the incoming user must send a message using the group's private key. The bootstrap server will then authenticate the message with the group's public. If the message is valid, the server will transmit the IP and port numbers of the group's online group-leaders. Needless to say, each member of a group must have the group's key pair in order to join the group. Without the key pair, the user cannot access online group-leader information and cannot enter the network from that entry point.

To maintain the state of the network, each group-leader and leader-of-leader is required to constantly update the bootstrap server with status reports of the system. The status messages must include IP addresses and ports of online group-leaders and LL's, number of connected users and their roles.

Bootstrap servers have the privilege of pruning the network of malicious nodes and reassign roles. For example, if a LL is not taking action on a group who repeatedly is being reported malicious, the bootstrap server can tell the surrounding LL's of the malicious LL to either demote or boot the malicious LL from the leader-group. More details about network safeguarding are described later.

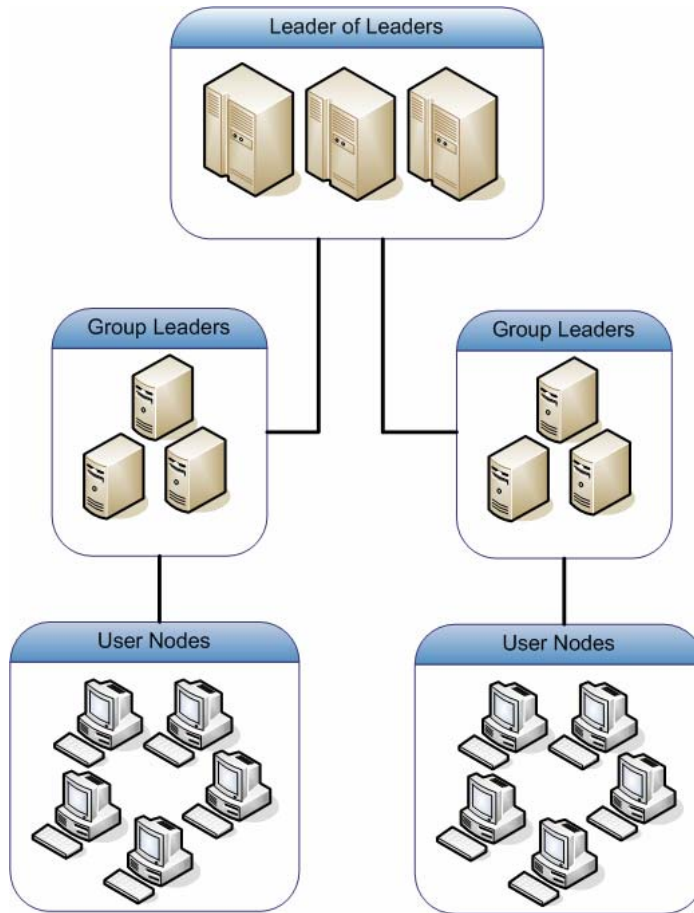


Figure 3: SARP Net's Node Roles

5.1.2 Group Formation:

With or without a bootstrap server, the network can allow for the creation of groups. When a connected user wishes to establish a group, a group creation request command is sent to the closest LL overseeing that section of the network. The command contains the future group's name, the group identification where the creator currently has membership to, the creator's username, and the future group's public key. The LL will then perform a query to retrieve the creator's group reputation and find out how many groups were created by members of this group in the past. If the creator's group is highly reputable and has a free group creation, then the future group's information will be shared among all of the connected LL's. When the group's creator wishes to invite a new user to join the group, the new user will be given the group's private and public keys to verify the user belongs to the group. If a bootstrap server is used, the new group's information and public key will be transmitted to the bootstrap server directly from the creator.

When a node wishes to locate fellow group-members through another node, a query can be sent containing a group search message encrypted in group's private key. Eventually a node with the group's public key will receive the message, most likely a LL, and retrieve the location of one of the group's active group leaders. The query will then be replied along the same path to the verified incoming group-member. If a group-member joins the network but is the only active one, the closest LL will recognize that group-member to be the sole active group leader and will redirect any future group inquiries to that active member.

When a new joins the group or when a group-leader is lost, group-leaders must make decisions on who to promote based on reputation, available bandwidth, and processing power.

Simple “ping” message can be used to gauge bandwidth and processing power to determine which user-node should be promoted.

Once a new group-leader is assigned, the other group- leaders will update the promoted node’s status by transmitting reputations, collected query data, and group status information. In addition the group-leader will establish connections to all of the group-members it is responsible for. The number of group-leader to user-node coverage is based on the group’s configuration where some groups may want all of their group-leaders to be connected to all of the members or have each group leader be in charge of a section of the group. Groups that allow group-leaders to connect to all group members allow each group-leader to act as a backup where if one group-leader is busy, another group leader is easily accessible to provide the same service. Also, messages can be verified easier with majority voting if repetitive messages are transmitted from one group member to all of the group-leaders. Groups that divide the group responsibilities decrease the workload placed on each group-leader and allow these nodes to focus serving a subset of the group.

Finally, the new group-leader will connect to the assigned LL’s to allow the group to communicate with other groups. Once the new group-leader is updated, that node can be used to help the group process network requests and traffic.

Leader-of-Leaders are assigned in a similar manner as the group-leaders. Based on the system configuration, the ratio of LL’s to groups should be adequate enough for each leader-of-leader to process network messages without creating a bottleneck or slow network performance. The performance of LL’s is vital to the health of the system since they provide the link that connect groups together and manages the communication between different sections of the network. With this reason, LL’s should be the most reliable computers on the network. Keeping

in mind that LL's are still nodes with the capability of becoming malicious, disconnect without warning, and suddenly experience poor performance bandwidth or CPU processing, safeguards must be incorporated. These provisions should gracefully handle the possibility of several LL's failing at a given time to prevent network paralysis. The most important LL criteria must be that the most reputable nodes with the fastest available bandwidth and CPU processing should be permitted to occupy the highest tiered role in the system.

When a group comes online for the first time, a leader-group will be assigned to it. This normally means the leader-group that receives the sole group-member will be the leader-group that will service it. If a LL disconnects, the LL's of that leader-group will promote the most outstanding group-leader or user-node. By promoting user-nodes to LL's, group restructuring is minimized. On the other hand, promoting group-leaders to LL's allow for faster node comparisons at the cost of forcing the chosen group-leader's group to restructure to find a replacement group-leader. It is vital for each LL of a leader-group to be connected to all of the same group leaders and LL's of other leader-groups. This requirement is for the stability of the network and the ability of LL's to reliably take part in the reputation scheme.

Leader-groups may be disbanded when the number of groups serviced by a leader-group becomes small. In this case, the leader-groups will pass on the responsibility of serving its groups to other leader groups. This should increase the number of potential query hits since groups are guaranteed to interact with more groups and their members.

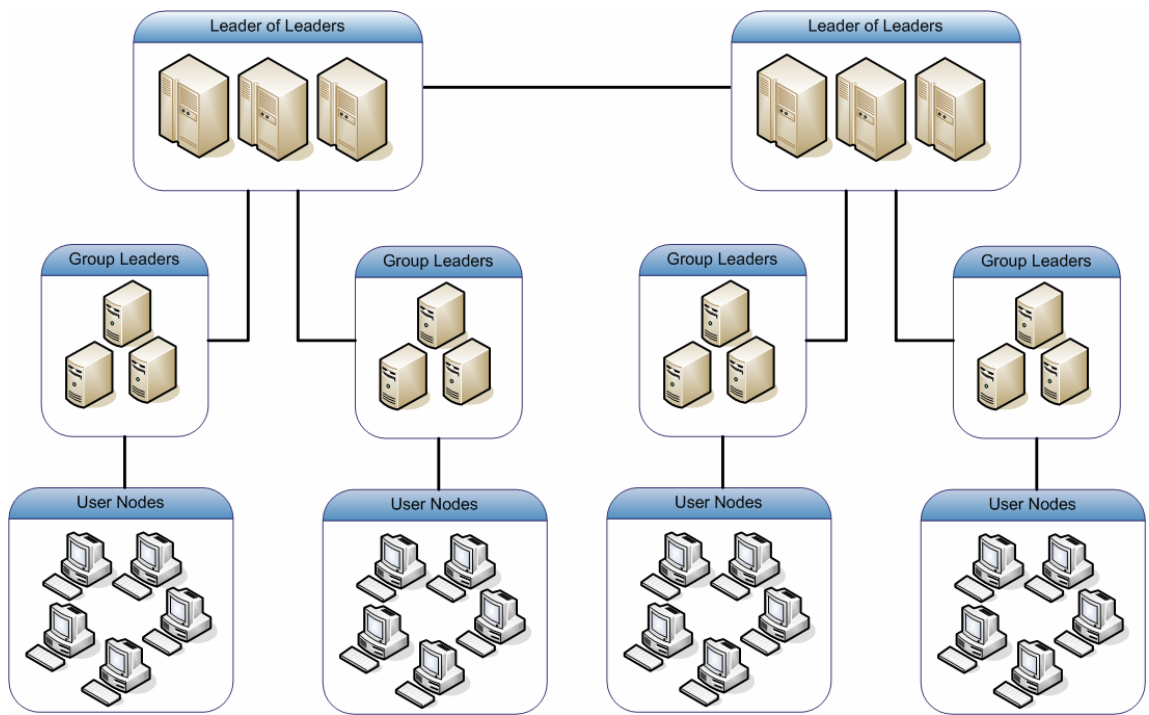


Figure 4: The SARP Net Topology

5.1.3 Node Limitations:

To set bounds on the amount of groups the system can have or the amount of untrustworthy nodes can join the network there are a number of limitations that are applied to all participants in the network. The first limitation is how many people a node can invite to a group. This limitation is monitored by the group-leaders but is stored by all of the group-members or on the group's dedicated server that can also act as a key distributor. For example, each group-member is given 1 invitation after being logged into the system for a total of 50 days, has shared or helped forward over 2 gigabytes of data, and has a high enough positive reputation. With this invitation, the group member can invite anyone to join the group and give that new member the

group's private key. With this invitation and reference from the group-member, the new member can connect to any active node in the system, be redirected to any of the target group's group-leaders for username registration and key exchange. Once the invitation has been used, the acknowledging group-leader will alert all of the group-leaders in the group who will help notify all other group-members that the inviter has used his invitation. If the group has a dedicated server established, this message can be sent to the server instead of having to be distributed to all group-members

In order to limit users from flooding the network, several limitations are suggested. Firstly, user-nodes should not be allowed to forward more than 2 queries per minute. Each group-member should keep a log of all incoming messages dating back to 2 minutes. This log will allow nodes to determine if any node is attempting to flood it or the network with queries. If flooding is allowed, the entire system can suffer from slow processing and see the same effects as DOS attacks. As detailed in the routing protocol, description message forwarding is an integral part of the system. These forwarded messages can too act as a method of flooding the system. To alleviate this potential threat of nodes attacking the network with a series of inauthentic forwarding messages, nodes are expected to process incoming messages on a queue. The message queue allows a node to manage messages and delay the forwarding request to prevent strain on the node's resources and the intended destination. Similar to query time limiting, a node should not forward a high amount of messages to the same destination for a set time. This limitation should be based on a node's group status and amount of traffic taking place. Certainly, the time restraint should not halt the querying process but it should be able to detect if a node is attempting to flood the system with messages.

5.2 Key Distribution:

Each node must store and transfer keys to keep the network secure. Several generated keys are exchanged to protect from eavesdropping, user impersonation, transfer verification, and forwarding receipts.

Upon group creation, the creator must generate a public-private key pair. The public key is shared throughout the network as verification for when a user claims to belong to the group. Each group member will sign a verification message in the group's private key to prove that he belongs to the group. A typical interaction would include a node A connecting to a LL B and claiming to be a group member of group X. A will send B a time stamped message encrypted in the group's private key. B will look up group X's public key to decrypt the message. If B can successfully decrypt the message, B knows that A belongs to group X. Each node on the network must store its group's public-private key pair. Without the key pair, a group member cannot connect and interact with its group.

All nodes must also contain their group identification key pair. Once a user is accepted as group member, it must create the key pair and share its public key with members of the group. This public key will be used for identification verification. After proving that the user belongs to a group and establishes a connection to a group leader, the user must prove its identity. The incoming user will transmit the username he claims to possess and a time stamped message with the private key of the username. If the group leader can decrypt the message, the incoming user can be accepted as the username provided. The group-member's public key must be distributed to all of the group-members to allow a member to join the active group regardless of which group-members are serving as the group-leaders. This key pair provides a method of verifying

users without running the risk of malicious users in the group of obtaining a fellow group member's password.

For each interaction between nodes, data must be encrypted in the source's private key then decrypted by the source's public key once transmitted by the recipient. Interacting nodes have the option of generating a session key that is established and generated once a connection is made. Still, messages need to be signed in the source's private key for interaction receipts.

To distribute keys, group creators can establish a long-lived server that acts as the group's centralized authority. This key distributor should store public keys for all group-members and the public-private key pair of the group. If each group establishes a key distributor and broadcasts information to contact the server, then key distributors can take the place of the typical central authority. The existence of group key distributors will make it easy to recover from key compromises, broadcasts of group alerts, and key look-ups.

5.3 Routing Protocol:

The routing protocol of this network describes how nodes communicate with one another. The passing of messages allow file queries and transfers, reputation look-ups and postings, and network status updates to be processed securely throughout the system.

File Queries: File queries attempt to search the network for shared files that satisfy the criteria in the query message. For example, if a node searched for "documentation P2P" query hits may return information on files that have filenames that contain word "documentation" or "P2P".

Typically, group-leaders and user-nodes should only be allowed to make file queries. This is to prevent any slow down and potential bottlenecks at the leader-of-leader level.

The routing of an initial file query message and a file query message to be forwarded are handled in the same manner. If LL's are given the privilege to start queries, and if a LL starts the query, it will send the message to all of the groups it is in charge of by sending the message to one of each of the connected group's group-leaders. The LL will also send the message to all of the LL's it is connected to. If a LL receives a file query from a LL in the same leader group, it will search for the file locally and not forward the message to anyone. If a LL receives the query from a group-leader, it will forward the message to all other connected groups by sending the message to one of each group's group-leaders, forward the message to the other LL's in its leader-group, and finally send the message to one LL of each of the other leader groups it is connected to. LL's will only forward a query if the hops-to-live (HTL) value is currently greater than zero. In SARP Net, HTL is used to maintain the scalability of file queries since it would be impractical for file queries to be forced to propagate throughout the entire network, especially if the network is large. The HTL value is decremented by 1 when a LL forwards the message. If a group leader initializes or receives a file query from a user-node, it will send a message to all of the group-members it is responsible for and one of its LL's. If the group-leader received the file query from any of its fellow group-leaders, it will perform the search for the file locally and forward the message to the fraction of user-nodes it is responsible for forwarding messages to.

User-nodes handle file queries differently than the leaders. A user-node who initializes a query sends the message to either another user-node in the same group or to one of the group-leaders based on a set forward probability. If this probability is 50%, the user-node will send the message to another user-node 50% of the time; the other 50% of the time the message will be

forwarded to a group-leader. When another user-node receives a file query from user-node it must forward the message to a third user-node or one of the group-leaders, again based on the forward probability. If a user-node received the same query twice from a user-node, it will forward the query to a group-leader.

A potential attack would be for user-nodes to forward a file query from a group-leader to other user-nodes so when a user-node receives the forwarded message it will forward it to other nodes. This would attempt to slow the system by occupying it with redundant forwards. One solution to this is that if a user-node receives 2 messages of the same query, one from the group-leader and one from a user-node, that user-node knows a user-node in the forward chain is misbehaving and should refuse to forward the message again.

All messages are encrypted in the private key of the source node (node A). Let node B represent the file query's destination. The following messages are the text that is encrypted in the source's private key. Below is the message for a file query being transmitted from A to B:

$$E_{Pr_A}(\text{"FQ"} \parallel TS \parallel QID \parallel HTL \parallel FID_A \parallel \text{message})$$

The contents encapsulated by E_{Pr_A} are encrypted in A's private key. "FQ" is the message keyword notifying the destination that this message is a file query. The symbol \parallel represents string concatenation. TS is the date and time stamp the query was made. QID is the query identification number. This identification is generated by the query source and should be unique to all file queries spawned from the source node. HTL is the remaining hops-to-live for the query. By default, HTL should be set to a value between 4 and 6, depending on the size of the network. If the destination node B is a LL and forwards the message, the forwarded message's HTL should be decremented by 1. FID_A is forward identification number generated by the node

that is sending the file query to a destination. This FID should be uniquely generated and should be different than the QID. It represents the identification number used by each node to provide proof that the node forwarded the message to another node. Valid FID's should comprise of a hash of the username and some number that is unique to all forward messages a node will experience during an active session. The label "message" is the file search criteria each user will use to find files to satisfy the query.

Query Hits: Once a node receives a file query message, it will search its local machine for a file that meets the search's criteria and if a match is found, a query hit message is generated. Query hits traverse the reverse path of the query once returned to a group-leader. Each node uses the query table to know who to forward a query hit to in order for the response to reach the query author. This query table is a history of all interaction receipts for incoming file queries and is stored locally. For example, if a user-node is the source (node A) and a query hit is found at another user-node in a different group (node B), B will send the hit message to either its group leader or another user-node in the group (based on the forward probability discussed earlier). Eventually the group-leader will receive the message and forward it to the LL who sent the query to the group. The LL will forward the hit to the source's group, and the group-leader who receives the hit will send the message to all of its fellow group-leaders. The group-leader who received the file query from the user-node will forward the message to that query source. The hit will traverse the user-nodes of the query's source until the source finally receives the message. At this point, the query author should forward the hit to a random group-member to create deniability. Again, let node A be the query author, let node B be the node forwarding the hit to A. If node A forwards the hit to another node (C), B cannot determine if A is the query author.

The forwarding should continue until a node has received the message twice or the forwarding probability has not been satisfied.

$$E_{Pr_B}(\text{“QH”} \parallel QID \parallel HID_B \parallel FID_B \parallel GroupID_B \parallel IP:Port_{MM} \parallel TS \parallel \text{File Info})$$

Above is the initial file hit the hit source will send transmit. Pr_B is the private key of the hit's source, Node B. The text encompassed by E_{Pr_B} is encrypted using the private key of B. “QH” lets the message destination (the node receive this message next) knows that this message is a query hit. QID is the query ID this hit message is replying to. HID_B is the hit ID which should be unique to all other hits to this query generated from this node to this query ID. FID_B is the forward ID used to monitor nodes forwarding habits. $GroupID_B$ is the group ID of group where the hit source resides in. This ID is used to identify where query hits and file transfers are coming from. The GroupID is specified by one of the hit source's group-leaders and is verified by the LL. Upon initial query creation, the query author specifies the group ID. Following the group ID is the IP and port of a middle-man that leads to the transfer of the requested file. This IP address must belong to a group-member of the hit source's group. It is important that middle-men are rotated to prevent any group-member overwhelmed by servicing too many file transfer requests. Next is the time stamp of when the query message was generated, followed by the file information containing the file name, file description, file offset and length, hash, and other vital information in regard to the file.

File Requests: Once a query hit has been returned to the query source (node A), a file request can be performed. To protect A's identity, A has the option to connect to the middle-man directly or ask another user-node in the group to connect on its behalf. If the request is forwarded to

another group-member, that node has the option to connect to the middle-man directly or forward the request again. Similar to forwarding file queries, this decision to connect to the middle-man or forward the request is based on a forward probability. It is essential that the number of forwards is low since long chains from a file query author to the hit author will create noticeable delays in the file transfer. Once a node from A's group connects to the middle-man of B's group specified in the hit message, the middle man will connect to a user-node in its group (node C) and request the file. If C does not have the requested file in its shared files or cache, it will forward the request to another group member who will perform the same recursive search. When a node with the file receives the forwarded search message, it will forward the request just like file queries and reply with the requested fragment of the file with a delay. The fragment is transferred the reverse path back to A.

$$E_{Pr_A}(\text{"FR"} \parallel \text{FRID} \parallel \text{FID}_A \parallel \text{IP:Port}_{MM} \parallel \text{TS} \parallel \text{File Hash} \parallel \text{File Offset} \parallel \text{File Length})$$

Above is the message for a file request. It represents the message transferred from A to the next node in the path to B. "FR" represents that this message is a file request. FRID is the file request identification used to distinguish file requests. FID_A is the forward ID used to make sure nodes are forward the message. IP:Port_{MM} is the IP and port number of the middle-man of the file source's group who is listening for incoming connections for file requests. TS is the time stamp of the message initialization. File Hash is the hash of the requested file to be transmitted. File offset is the beginning offset of the requested file's segment. File length is the distances from the offset to the requested file segment length.

It is important to note that the hit source's identity is never truly known by the query's author. From this, nodes cannot rate individual node file transfers but can leave evaluations about

interactions with other groups. The goal of this is to force groups to maintain a healthy level of behavior and isolate malicious nodes.

Interaction Receipts: Interaction receipts are evidence that two nodes interacted. They are used to provide nodes with proof of forwarding and receiving messages. Interaction receipts here are very similar to the ones described in [25]. The receipts in SARP Net are the only pieces of evidence a node can use to accuse another node of not forwarding messages or message tampering. Each receipt is signed by the two interacting nodes' private keys. For instance, if node A sent node B a message, the message would contain text encrypted in A's private key. Once B receives the message, it will reply to A with the same message encrypted in B's private key along with other data. B will also forward the receipt to a third-party node that A can access. If two user-nodes are interacting, the receipt will be sent to one of their group-leaders. If a user-node and group-leader interact, the receipt is forwarded to another group-leader. If another group-leader is not available, the receipt will be sent to other user-nodes.

Node B will append to the reply receipt (the receipt from B to A) its next forward ID. The forward ID is a number that represents the identifier B will use to forward another node the original message. For example, if B forwards the message to C, that message will have B's forward ID. Since the reply receipt has the forward ID from B to C, node A can query a node that is visible to itself for the forwarding receipt C generated after receiving the message from B. With the reply receipt node C sends B, B can forward that reply receipt to A, so now A has proof that B forwarded the message to some one. If a node does not receive a forwarding-receipt from a node, that node can always query neighboring nodes for the receipt. If no receipt is received, then a node knows either the non-forwarding node is either misbehaving, was providing message

cover to protect a query author's identity, or is the source of the file for file transfers. Since nodes forward hit messages even if it is the query author, queries are difficult to track who is the query author without mass collaboration. For file transfers, the failure to produce a receipt allows groups to pinpoint which nodes are transmitting malware without knowing which filenames produced the query response and divulging information about what files nodes are downloading and which files nodes are sharing.

Reputation Messages: Reputation messages are what distribute node ratings across the network. Once a node rates another group, it must distribute the message so it will propagate throughout the network, spreading that node's opinion about the group being rated. The fastest and most common way for a node to alert other groups about a particular group's behavior is from file transfers. Once a file has been evaluated by a node, that node will send a reputation message along the same path as the file transfer traversed, from the rating node, through intermediate nodes in the same group (if any), to the middle-man, through intermediate nodes of the file source's group (if any), to eventually the file source. The first node along that chain that cannot produce a receipt saying that it received the file fragment from someone else will be determined the file source. The node that locates this file source will send a message to its group-leaders with the interaction receipt that it received the file fragment from the file source along with the reputation message coming from the node that made the query, also known as the file's final destination. Also, the file's final destination node will spread the reputation to all of its group members and to the group-leaders of the file source's group to notify the group of the incident.

Interaction and Reputation Queries: Interaction and reputation queries propagate differently. Interaction queries are almost always needed for proof of interaction within the same leader-group, meaning that the interaction is normally for someone governed by the same leader-of-leaders. With that said, these queries have a HTL is set to 1. If the receipt is needed for a node outside of the leader-group, then the HTL can be increased. Reputation queries traverse the network the same way as file queries. Its HTL should be higher than interaction queries in order to poll a larger number of groups. Ideally, this HTL should be set between 4 and 6, depending on the say of the network and the number of nodes.

5.4 Fragment Caching:

SARP Net incorporates caching of file fragments to increase transfer speeds. Each node has the ability to set a cache limit which will store as much data as that limit allows of fragments a node receives as an intermediate node or middle-man node during file transfer. Each cache entry will have metadata that will contain information about how that node received the fragment. This includes the interaction receipt and the fragment's hash value. It should be noted that only a file source can reply to a file query as a query hit. Cache hits should only take place during file requests. If a file request comes in for a particular hash value, any node with that file fragment can reply. This allows for swarming/multi-source file transfers from one group to another. It is multi-sourced since when a file request is forwarded by a middle-man. Middle-men randomly select nodes to forward requests to. If this is done for different file fragments that are stored through out the group, an entire file's fragments could all be transmitted to the middle-man by different nodes.

All reputation ratings are tied with the fragment's original source. This makes it the sole responsibility of nodes to share benevolent files. If a node possesses a malicious file fragment in its cache, it should remove the fragment as soon as the reputation message is received. Violators can be found with query probes by group-leaders to locate misbehaving nodes.

Cache entries should only contain the most requested fragments while respecting a set time-to-live (TTL). A node should delete a fragment from cache when the fragment's source (the node that sent the fragment) disconnects from the network. This makes it easier for nodes to locate file origins when a rating is returned.

5.5 Group-Based Reputation Scheme:

The reputation scheme employed by SARP Net is a simple group-based scheme. Instead of rating each node individually to attempt to predict node behavior based on previous experiences noted by other users, this reputation scheme evaluates a group of individuals' network behavior and applies the rating to everyone in the group. The basis of this idea comes from the proverb "it takes a village to raise a child". In this case, groups are responsible for their group-members while group-members are responsible for their groups. If a group-member misbehaves, the entire group suffers. This forces responsible group-leaders to govern their group-members very strictly. The consequences of group-leaders not maintaining good behaving group-members should see those group-leaders being demoted and potentially suspended or exiled from the group. Group leaders have the tools to determine which nodes are misbehaving in the form of interaction receipts and reputation messages propagating back from other groups.

Groups that misbehave suffer the consequence of not having queries forwarded and answered from other groups. Since file queries and file downloads drive the demand of using file-sharing applications, the lost privilege of file querying severely hurt the group's functionality. It is assumed that nodes of misbehaving groups would soon want to participate in file queries and eventually leave a group to join a reputable one. This is why it is important for group-leaders to be swift when it comes to filtering out misbehaving nodes.

The rating calculation is relatively simple. Nodes rate good interactions with positive 1, neutral interactions with a 0, and negative interactions with -1. Positive interactions for file transfers can be for files that contained no malware and have a file name that accurately describes the file's contents. Neutral interactions are for events that could not be determined if it was a positive or negative event. Negative interactions are for events that were poor, such as malware distribution and occurrences of dummy files. If a rating is negative then interaction with that group is not suggested. If a group's reputation is neutral then it is considered a 50/50 chance of the node being malicious or information is not gathered about that group. Positive ratings mean there is a higher chance of a good interaction would occur. It is important to note that since reputations are solely based on previous interactions with a group's group-members, the reputation scheme cannot guarantee that any future interaction will be innocent or malicious.

To prevent any one group's opinion of another group from dominating an overall reputation, averages are used to normalize ratings among other groups. We will denote the group being evaluated as the target group. To calculate a group's reputation, a node will query the network for any occurrences of reputations for the target group. After waiting (approximately 1 minute), each returned reputation message is gathered and duplicates are discarded. The reputation scheme calculates the target group's rating by summing the average of each returned

group's total unique rating of the target then dividing that sum by the number of groups that returned reputation messages. For example, if group A was the target group and if group B had 2 unique ratings of A {1,0} and group C had 3 unique ratings of A {-1, 1, 1}, the following calculations would occur. First each group's rating is averaged: Group B's average rating of A is 0.5. Group C's rating of A is 0.333. Second, the two averages are summed and divided by the number of groups that returned messages: 0.41665. In this case, group A would be considered trustworthy. For group-leaders evaluating its group-members, a node is considered malicious and should be punished if its overall rating is negative and 3 or more groups have rated it. This is the default rule for determining if a node is truly malicious. For any case of negative ratings, group-leaders can configure their own rule set for determining when to punish a node. One such rule may be evaluating each negative reputation message's interaction receipts to determine if the allegation is plausible.

The following represents the equation to determine a group's reputation rating. R_x is the overall rating of group x. n is the number of groups with at least one rating of group x. $\bar{r}_{i,x}$ is the average rating from group i on group x.

$$R_x = \frac{\sum_{i=1}^n \bar{r}_{i,x}}{n}$$

By rating groups, node retaliation for setting negative or neutral reputation ratings are almost eliminated since individual nodes are not being rated. The only form of retaliation could be entire groups attempting to sabotage other group ratings by flooding the network with false reputations. For starters, each reputation message needs to have interaction receipts appended to it to prove that the interaction actually took place. The second tool against this form of retaliation

is how the system calculates reputations to normalize group responses with other groups. No one group could dominate the evaluation of another.

5.6 Addressed Security Issues:

Intrusion and Malware Detection: SARP Net was designed to be coupled with an intrusion detection system (IDS) and an anti-virus software in order to promote security across the network. If each node is executing some form of these two tools, malware detection and malicious nodes can be identified faster and more efficiently. The SARP Net client software should be implemented with the functionality of easily tying itself with an IDS and anti-virus of the user's choice. This feature would guarantee that the communication between the applications would protect the user's machine and others in the network by constantly monitoring incoming data. If an anti-virus application could automatically scan files as soon as they are transmitted, reputations can be generated as quickly based on the outcome of file scans. With SARP Net's ability to send messages with an author's group reputation, security warnings can be propagated through out the network, alerting nodes of malicious activity and updating both IDS's and anti-virus applications of things to look out for. Of course such alerts would not be the only verification since even the highly reputable nodes can be malicious, but could be accompanied by the software or user contacting the IDS or anti-virus's company directly for verification.

Anonymity as a Security Tool: It is hoped that with an anonymous P2P network, the protection from identifying user activity would increase a node's security. Attacks like spyware's information gathering on incoming query hits should suffer greatly. This is true since nodes are

expected to accept query hits from other nodes with the intention for forward it to the source. This load of data makes it hard to distinguish the source of a query.

The amount of cover that groups provide their members is essential to protect identities. The scale of the cover creates a level of deniability nodes can have of query and reputation making and file possession. The higher the cover: the higher the level of deniability. Groups fewer than 4 nodes suffer from a higher probability of being able to pinpoint which nodes are making which queries and sharing which files in its own group. A solution is to prevent group-members from replying to fellow group-member queries with hits if the group is too few connected nodes.

The risk of Sybil attacks [8] are also addressed in SARP Net. With its design, SARP Net makes it possible for nodes to own more than one identity on the network. The worst case would be for malicious nodes to create malicious groups with the sole intent to spread malware or to harm the system. The limitations on group formations, limitations on node invitations, and monitoring which nodes have created groups should hold against any large collaboration from misbehaving. Simply put, groups consisting of malicious nodes will receive poor reputations and will be detected. Once identified, neighboring groups and overseeing LL's will do their part and discontinue a group's privileges if malicious activity is taking place or be judged negatively by benevolent groups. In the case of a single node performing a Sybil attack, the limitation on the number of invitations a node can give out prevent a node from creating a lot of identities easily. By forcing nodes to behave correctly for a set online-time, sharing a set amount of data, and obtaining positive ratings before gaining the right to invite a new group-member, Sybil attacks are highly discouraged. In addition, each node is linked to the node that referred it to the network, just like in F2F networks. A system could be employed to where if a node is detected as

malicious, its referring node suffers. The blame on introducing malicious nodes can recursively trek its way to a large degree of referrers to punish nodes of allowing malicious nodes to infect the system.

5.7 Prototype:

To determine the feasibility of SARP Net, a small prototype of its highlighted features was implemented. File queries, requests, and their responses were all designed for this prototype. Interaction receipt exchange and queries, as well as reputation generation and requests were left out to focus on the transfer of anonymous files and the system's node join/leave management. The prototype was designed in Java 5 for portability and consisted of separate client and bootstrap server applications.

The client application allows users to join groups, share files, make queries, and interact with other users securely and anonymously. The bootstrap server provides each client with a single starting entry point to the system. This server is used for nodes that lack any knowledge of other active nodes on the network. Clients can bypass the bootstrap server by connecting to an active node's IP and port if are known. For the sake of testing purposes, the bootstrap server automatically assigns incoming nodes to groups and which role they should portray. For nodes who bypass the bootstrap server, if the incoming node belongs to the group of the active node, the active node will send information about its group-leaders so a connection can be made between the incoming node and the group-leaders. Here, group-leaders will assign the incoming node as another group-leader or user-node.

The bootstrap server has the additional task of providing users who are not running the client software to query and download files through a website hosted by the bootstrap. With this feature, users can connect to the network to download files from any computer in the world without the need to install the client software. To deter users from only using this web feature, usernames and passwords are required to track the number of downloads. Also, users of the website are not returned group ratings with their file queries nor are they allowed to rate groups based on their interactions.

All SARP Net key pairs were generated with the RSA algorithm to provide the necessary asymmetric keys. Key pair generation was performed by each node upon joining the network for the first time. After being assigned to a group, a node was given the group's key pair and generates its own identification key pair as described earlier. The node's identification public key was then sent to the bootstrap server for storage. Any node that needs a public key of any group or any user can request one from this server. Once all of the needed keys have been exchanged or generated, a node could participate in the network securely by decrypting incoming data and encrypting its outgoing messages as explained in the routing protocol.

Search - Mozilla Firefox
 http://localhost:8080/?find=pdf

pdf Search

Results:

Filename	Track Title	Artist	Album	Track #	Duration	Bit Rate	File Size	Sources	Rating	
!!!!TK3-K5d-Peer2PeerPrint - Presentation.pdf	N/A	N/A	N/A	N/A	N/A	N/A	1212	1	1	[download]
3LS - A Peer-to-Peer Network Simulator.pdf	N/A	N/A	N/A	N/A	N/A	N/A	235	1	1	[download]
A Definition of Peer-to-Peer Networking for the Classification of Peer-to-.pdf	N/A	N/A	N/A	N/A	N/A	N/A	163	1	1	[download]
!!!!TK3-K5d-Peer2PeerPrint - Presentation.pdf	N/A	N/A	N/A	N/A	N/A	N/A	1212	1	1	[download]
3LS - A Peer-to-Peer Network Simulator.pdf	N/A	N/A	N/A	N/A	N/A	N/A	235	1	1	[download]
A Definition of Peer-to-Peer Networking for the Classification of Peer-to-.pdf	N/A	N/A	N/A	N/A	N/A	N/A	163	1	1	[download]
A First Look at Peer-to-Peer Worms - Threats and Defenses.pdf	N/A	N/A	N/A	N/A	N/A	N/A	65	1	1	[download]
A Generic Peer-to-Peer Network Simulator.pdf	N/A	N/A	N/A	N/A	N/A	N/A	816	1	1	[download]
A Measurement Study of P2P File Sharing Systems.pdf	N/A	N/A	N/A	N/A	N/A	N/A	510	1	1	[download]
A First Look at Peer-to-Peer Worms - Threats and Defenses.pdf	N/A	N/A	N/A	N/A	N/A	N/A	65	1	1	[download]
A Generic Peer-to-Peer Network Simulator.pdf	N/A	N/A	N/A	N/A	N/A	N/A	816	1	1	[download]
A Measurement Study of P2P File Sharing	N/A	N/A	N/A	N/A	N/A	N/A	510	1	1	[download]

Figure 5: SARP Net Prototype: File Query Web Results

During prototype testing, one bootstrap server and 7 client instances were executed on the 3 different computers. The resulting network topology was 2 groups with 3 members each, one leader-of-leader, and one bootstrap server. Figure 5 displays a screen shot of the results returned to a web user after a file query with the search string of “pdf”. The green hyperlinks to the far left of the web browser are the download links for the returned files. These links points to the hit’s middle-man’s computer and begins the file request process. Queries and requests originating from web users slightly deviated from the routing protocol described in section 5.3. These users connect directly from to the middle-man during file requests and forward file queries to the bootstrap server. Obviously, file queries channeled through the bootstrap server are not anonymous and can be traced; this being another incentive for users to install and use the client software.

The prototype showed that data could easily flow from group-to-group under small and favorable conditions without any difficulties. It also showed that the system is feasible in node management and data transfers. The system proved to be able to handle nodes joining and leaving the system gracefully with the aid of group-members constantly updating each others statuses. File transfers with both the use of hops for anonymous routing and with the use of the web feature, proved reasonably successful with files making their way from a source to the query author.

CHAPTER SIX: DATA ANALYSIS

The following section describes the experiments that were performed to analyze the proposed network and reputation scheme. Section 6.1 is an overview of the P2P simulator that was developed. Section 6.2 describes the first experiment performed with the simulator. Sections 6.3 and 6.4 discuss experiments 2 and 3 which extend the first experiment with the goal of increasing transfer speeds.

6.1 Simulator Overview:

A number of simulators were considered to evaluate the performance of SARP Net under several different configurations, including 3LS [24], P2P Sim [10], and PeerSim [14]. Due to the desire to simulate the unique features the system possesses in detail, such as the reputation scheme and interaction receipts, the simulator used in PGS Net [19] was used for its ability to simulate a network structure similar to SARP Net. This simulator was modified to support the proposed architecture, its routing protocol, and security mechanisms. The simulator's design was heavily influenced by [26] for system comparison detailed later. The simulator was used to monitor the network behavior and feasibility of the system. The simulator, renamed to SARP-Sim, is a discrete event simulator fully implemented in Java. Similar to [26], encryption is not performed at any point in the simulation and nodes joining and leaving the network are not simulated. The simulator pre-establishes the network with node placement and their web of interconnectivity.

To examine the different network structures SARP Net can be given, the simulator allows for the specification of different group sizes and assignments of the three node roles based on

configured inputs. The characteristics that heavily affect a node's functionality in the network are the different bandwidth limitations and number of shared files. Both of these attributes are randomly selected based on a specified distribution to give nodes the ability to accurately simulate real user machines on the internet.

Each simulated node is represented by additional real world characteristics such as bandwidth, listing of shared files, malicious intent, and cache size. Nodes are given a number of shared files that they will offer to other nodes to download. To simulate this feature, the randomized number of shared files per node is used to represent the list of files to share. This is a list of integer numbers ranging between 0 to the number of possible file name combinations specified by simulator input. For instance, if the number of combinations is 200,000 and node A has 1,000 files to share, that node has a list of 1,000 randomized numbers between 0 and 200,000 without any duplicates. When another node, node B, queries A for a file, B will randomly select a number between 0 and 200,000 (call it `random_file`) and ask A for that file. Node A will respond with file query hits for every instance of files in the list that are in the range of $(\text{random_file}-100, \text{random_file}+100)$ to simulate if B queries non-specific files. An example of this would be if B is looking for all files that have filenames that contain the word "document"; a single node may share a lot of files with the phrase "document" in its filename and should reply with them all.

6.2 Experiment 1:

SARP Net was simulated using SARP-Sim for 10 runs to determine the average system statistics. 10 runs were performed since there are a number of random variations per simulation run. Each run simulated the P2P network for 3600 seconds (1 hour) with 4,000 nodes processing messages and interacting with each other. Of these 4,000 nodes, 200 groups were formed with 20 group members each. A 5:1 ration of user-nodes to group-leader was used. In addition, each leader-group was assigned to oversee 3 groups where each leader-group was comprised of the 3 fastest group-members. As recommended by the protocol, leader-of-leaders are the only nodes prohibited to make file requests and share files. This is to help ease the large workload of message forwarding and group-governing expected of these leaders.

The method of assigning bandwidth and number of shared files for each node were comprised of the findings in [23], [26], [17]. 70% of all nodes had high speed internet connections. Of those nodes, 90% had cable or DSL connections and 10% had T1 or T3 connections. Of the 30% dial-up nodes, 60% possessed 33.6-56kbps connections, 15% possessed 64-128kbps connections, and 25% used 14.4-28.8kbps connections.

The number of shared files was determined by the bandwidth, as influenced by the results in [23]. The dial-up nodes had a 20% chance of sharing no files, 60% probability of sharing between 0 and 100 files, 14% chance of sharing between 100 and 1,000 files, and a 6% chance of sharing 1,000 to 10,000 files. The high-speed nodes had a 10% chance of sharing no files, 60% chance of sharing 0 to 100 files, 24% chance of sharing 100 to 1,000 files, and 6% chance of sharing 1,000 to 10,000 files.

2,500 file queries were processed randomly throughout the simulated time. Each node was given an equally-likely chance of making any of the 2,500 file queries at any given time. Each query's HTL was set to 5. For 20 percent of all query hits returned, the query author produced a reputation query for the hit source's group and waited 60 seconds to receive the group's rating. If the group's rating is greater than or equal to 0 then the query source would issue a file request for the file returned in the hit. Additional file transfer settings include: all file transfers represent a 1 MB fragment of the queried file, each node has a cache size of 50 entries (50 MBs each), and the forward probability, probability of a node connecting to a specified middle-man or forwarding the request to another group-member, was 50%.

To test SARP Net's ability to locate and punish malicious nodes, all nodes were given a 20% chance of being malicious since this allows for a large proportion of the network to be benevolent while having a large enough visible proportion of the network misbehaving. All malicious nodes transfer malware 80% of the time and it is up to the file requestor to detect if a transferred file is indeed malware. 5% of all files transferred by each innocent node were malware, the rest are benevolent files. With this said, if a hit's source transmits a malicious file, it is assumed that the requestor will be able to detect the malware and will distribute a negative rating for every occurrence. Group-leaders will disable a node's file sharing privileges if it has an overall negative reputation and 3 or more groups have produced ratings on it. This loss of privilege can represent a node being expelled or suspended from the group.

These experiments focus on malware detection and eliminating the spread of malicious files by locating its source. Even though a node refusing to forward messages is a problem this protocol addresses, nodes are not judged on this criterion in the simulation. Instead the load that

interaction receipts placed on the network is monitored for their effect on query response and transfer speeds.

6.2.1 Results:

The experiment yielded many noteworthy results. The following results are averages over the 10 runs of the simulator. All experiment data averages and standard deviations can be found in the appendix.

Search Results: Of the 2,500 file requests, close to 225,422 hits were generated from 404,304 file searches, with about 224,828 of them reaching the file query author. The average time for the first query hit was 7.11 seconds. These query hits spawned 40,260 file requests resulting in 30,935 uploads attempted, 2,729 downloads completed, and 19,175 file fragments transferred including downloads from hops and the file request author.

File Transfer and Bandwidth Statistics: Even though the average bandwidth for each node was 78.05 kB/s, the average download speed was 7.06 kB/s. This download speed was the average transfer rate from the last hop to the file request author. It took an average of 701.19 seconds for a node to receive a file after issuing a file request. This delay is directly related to the number of hops from the file request author to the file's source. It is the time needed for each hop to receive the file plus the time for locating the file's source. Finally, the system used less than 21% of its bandwidth to process requests, responses and transfer files over the hour.

Malicious Activity Detection: The system performed relatively well in detecting malicious activity. Of the close to 800 averaged malicious nodes in the network, just over 112 of them were correctly detected within an hour. Unexpectedly, an average of 3 benevolent nodes was detected

malicious per run. This false-positive rate is due to the low probability of 5% that each benevolent node had to share an unknown version of malware. These nodes beat the odds and generated overall negative ratings with more than 2 groups. The system was able warn approximately 4,078 returned hits of being malicious from the hit's group behavior.

Figure 5 shows the percentage of free and used bandwidth of all nodes over the course of the simulated hour. This graph proves that as time progresses in the simulation, nodes begin to issue more and more queries, transmit files, and send reputation and interaction receipt messages. The decrease in free bandwidth percentage in the first 300 seconds is due to the network increasing the amount of messages and network activity. Around the 300 second mark, the bandwidth demand begins to stabilize around 79% free. Towards the end of the simulation, the free bandwidth percentage begins to increase again since most of the file requests have been completed.

Figure 6 shows that the increased number of reputations created and the number of malicious nodes are related since malicious node detection relies on the spreading of reputation messages. As time progresses, more reputations are made since the number of completed downloads are increasing. The slower ascent of malicious node detects to number of reputations in the system is due to the amount of confirmation group-leaders need before declaring a node malicious.

Figure 7 displays the percentage of network activity from the total bandwidth utilization. The most bandwidth usage comes from file transfers, making up 42% of all bandwidth utilization. This large portion is due to the data hopping caused by file requests. With nodes providing cover for file sources and query authors, the number of hops requires nodes to

continuously forward file fragments. Interactions make up 38% of the bandwidth utilization since receipts are spawned from file and reputation queries, requests, hits, and transfers.

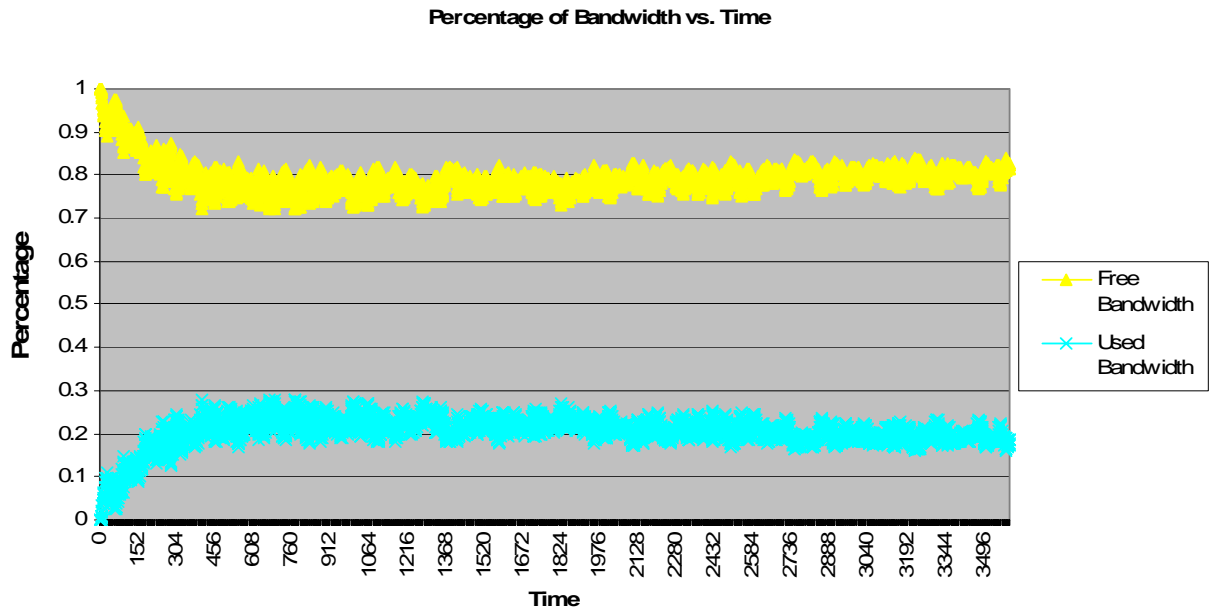


Figure 6: Utilized Bandwidth Percentage vs. Time for Experiment 1

Exp.1: Percentage of Bandwidth Utilization

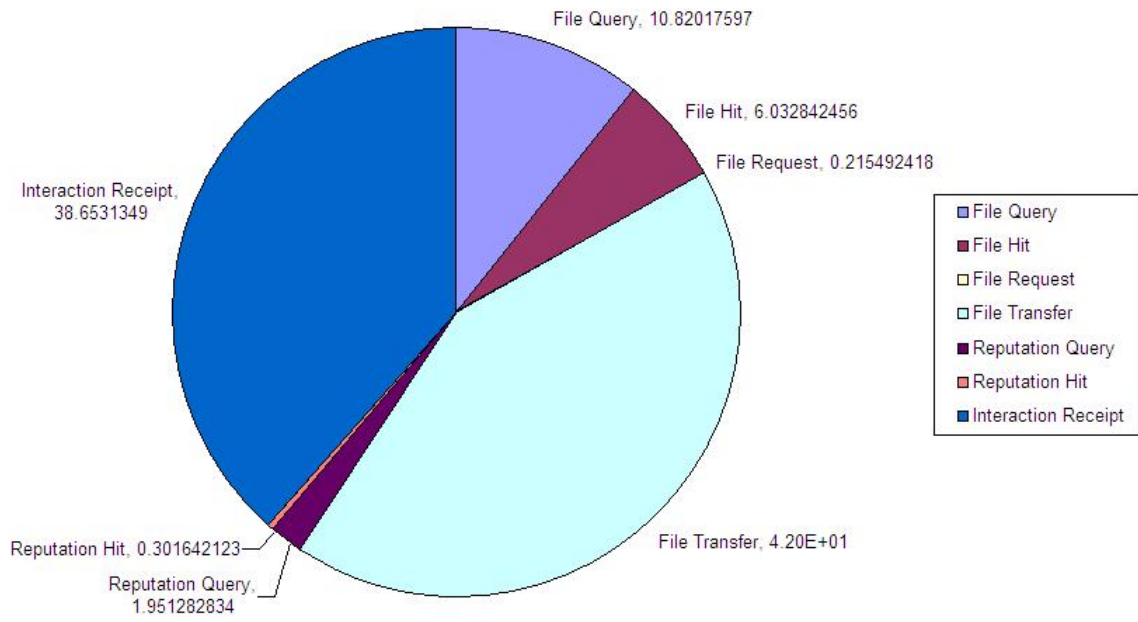


Figure 7: Percentage of Bandwidth Utilization for Experiment 1

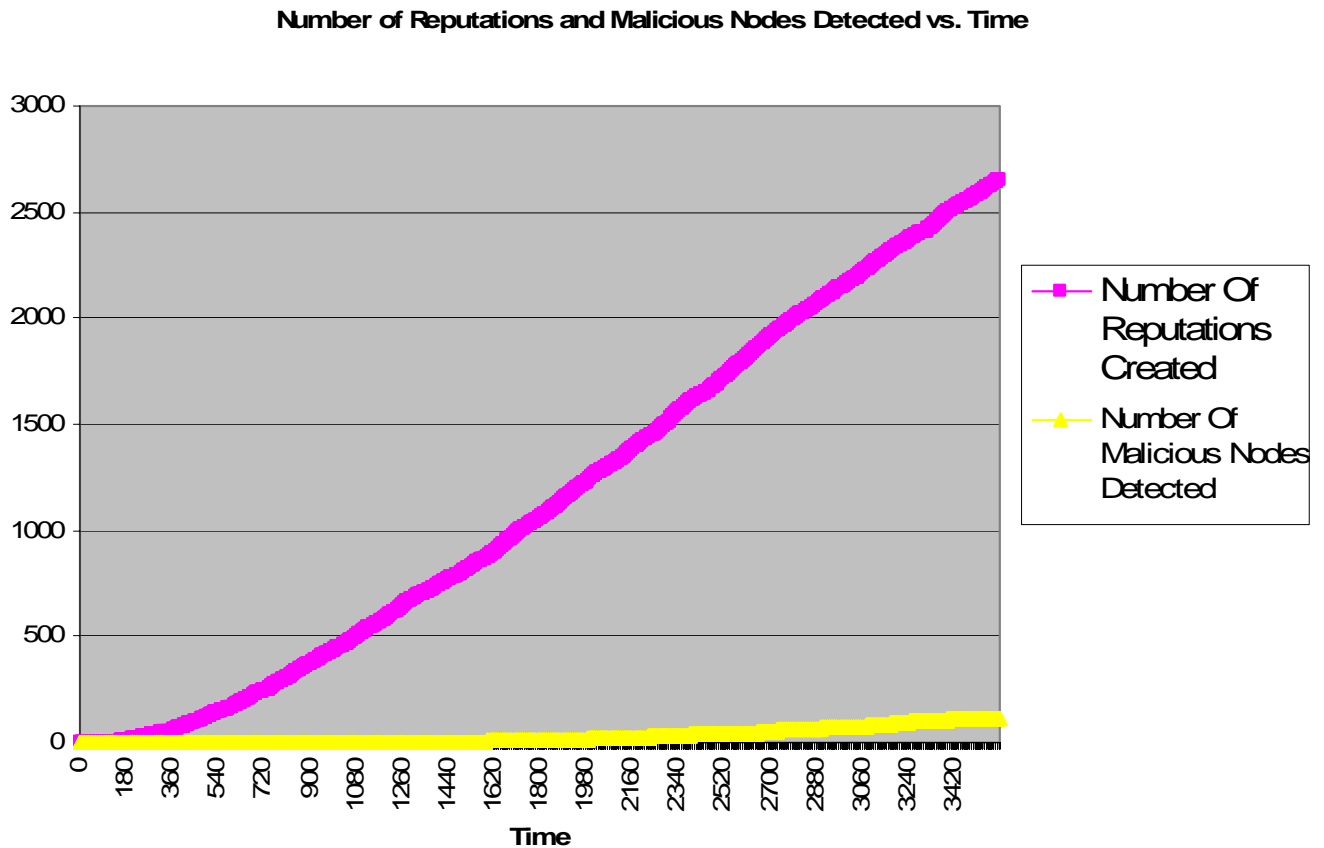


Figure 8: Reputations Created and Malicious Node Detection vs. Time (Experiment 1)

6.3 Experiment 2:

After noticing the delay the hops place on file transfers, another experiment was executed, this time decreasing the size of groups from 20 to 10 nodes, setting the number of groups to 400 to still simulate 4,000 nodes in the network. An additional change included the assignment of a group-leader for every three user-nodes. All other simulation settings were kept the same as the first experiment.

With the new change in topology, a few statistics were improved while others saw the opposite. The noteworthy changes include the transfer statistics and malicious activity detection. Over the 10 runs, experiment 2 produced a faster average file transfer speed of 9.30 kB/s, compared to experiment 1's 7.06 kB/s. Also, the average time needed from file request to file being transferred back to the request author was 664.17 seconds, 37.02 seconds faster than experiment 1's. Experiment 2 saw the system only detect slightly over 67 malicious nodes correctly and an average 1.4 benevolent nodes detected as malicious.

6.4 Experiment 3:

Experiment 2 showed that file transfer speeds could be increased with smaller groups since there are less nodes to provide cover. These increased speeds were at the cost of fewer nodes receiving queries due to the HTL set to 5 and also showed a decrease in malicious node detection. To remedy this, a third experiment was devised, this time increasing the HTL to 6. The goal of increasing the HTL to 6 was allow more query hits to be generated and increase node interaction. All other settings were left the same as experiment 2.

Of the 514, 797 file searches from the 2,500 file queries, there were 275, 511 hits. On average, 7.26 seconds were needed for the first query hit to make its way back to the query author. The query authors saw an average download speed of 11.3 kB/s and waited 183.79 seconds for each file. Malicious node detection was at its lowest in this experiment. Only 16 malicious nodes were detected of the 796 in the network.

6.5 Discussion:

After seeing the results the experiments yielded, it is clear that SARP Net can accurately detect malicious activity with an extremely low false-positive rate. The level of accuracy is high since the system labeled nodes that transmitted malware as malicious regardless if the node was initially flagged as benevolent or not. This is because of the probability all nodes had of transmitting malware. With the high percentage of malicious distribution for malicious users and the extremely low percentage for benevolent users, benevolent nodes should typically produce positive ratings. Nodes initially set as malicious should almost always produce negative ratings.

Even though none of the experiments correctly identified all of the malicious nodes and the simulator assumes every node has an extremely accurate anti-virus, SARP Net's level of detection is commendable. Since the system's rating depends on the number of uploads nodes perform, the simulation would have to guarantee that each node makes a large number of uploads to multiple groups to be detected. The reputation scheme works well with detecting nodes that consistently act maliciously. The true detection takes place at the group level, where groups must incorporate a set of detection rules described earlier.

It is difficult to compare SARP Net with other anonymous P2P file-sharing systems due to the system's goals, design, and functionality. Regardless, SARP-Sim was designed to produce measurements to compare SARP Net with the experiment results of Freenet described in [26] and its simulator Eos. Since Freenet and SARP Net are different in functionality (Freenet is an anonymous distributed data storage and SARP Net is a distributed file-sharing network where nodes know what files they are sharing and has caching capabilities), file transfers and bandwidth utilization are the most comparable metrics of the two systems. [26] presented

modifications to the Freenet design in order to increase file transfer speeds. The paper leaves doubt of whether the system still protects user identities, so only the results for the original Freenet protocol will be compared. Freenet's average download speed was simulated to be between 9.7 kB/s for the nodes with fast internet connects and 6.2 kB/s for slow, dial-up nodes. With an average download speed of 7.1 kB/s in experiment 1, 9.3 kB/s in experiment 2, and 11.29 kB/s in experiment 3, SARP Net's file transfer speeds are very much comparable with Freenet's. With about 70% of all nodes in the simulation of Freenet having high speed internet connections, one can deduce that an average file transfer speed of both fast and slow nodes is 8.65 kB/s. Compared to this average, SARP Net outperforms Freenet by 30.5%. The same could not be said for total bandwidth utilization. Freenet's bandwidth utilization for both download and upload combined averaged 4.98% usage, where SARP Net's nodes used 20% in experiment 1, 14% in experiment 2, and 17% in experiment 3.

It is gathered that SARP Net's load on bandwidth utilization does not necessarily mean the existence of network inefficiency but that nodes are trying to processes more system functions such as the addition of interaction receipts and reputations than Freenet. One can conclude that SARP Net's bandwidth utilization is a tradeoff for its improved security measures in an anonymous environment.

Average File Transfer Speed

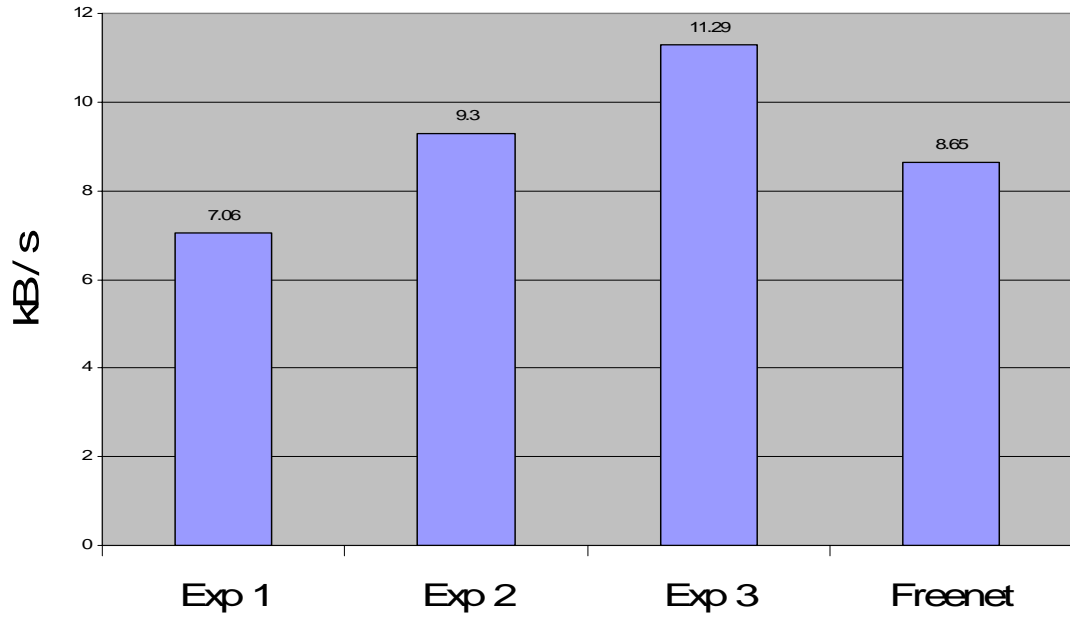


Figure 9: Average File Transfer Speed

Bandwidth Utilization

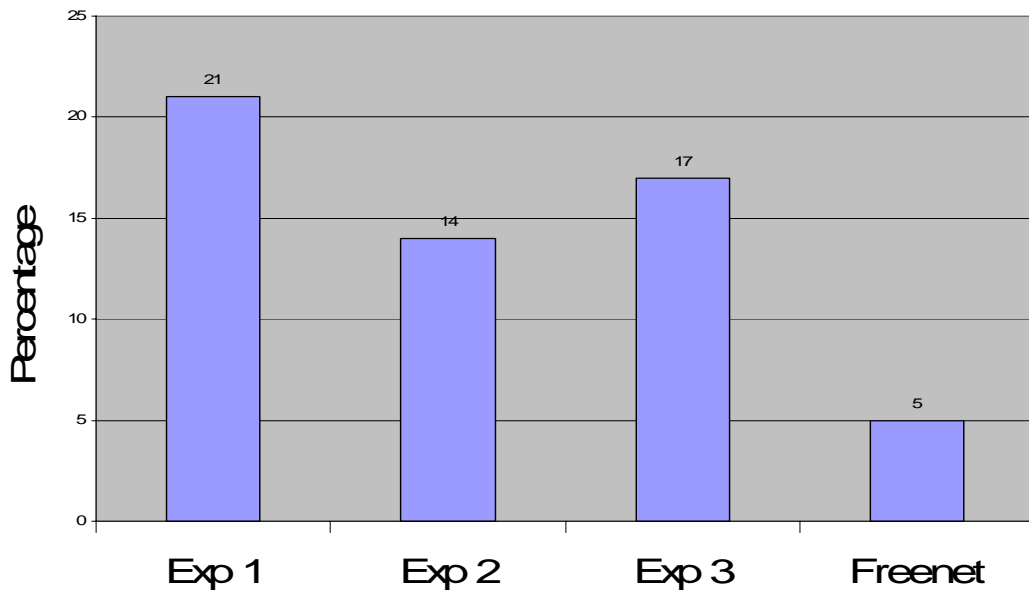


Figure 10: Bandwidth Utilization of SARP Net vs. Freenet

CHAPTER SEVEN: CONCLUSION

7.1 Conclusion:

In this paper, SARP Net, a secure, anonymous, reputation-based, P2P network was described in detail. The protocol is secure due to the necessity of all nodes exchanging encrypted messages and incorporating third-party intrusion detection system and anti-virus software to protect nodes. The anonymity of nodes is guaranteed through the notion of hop making, using nodes in the network as cover to allow for deniability for network activity. The reputation scheme incorporated in the system rates collections of nodes as groups and not based on individual efforts. This sense of group-rating places the emphasis that group overseers (known as group-leaders in this system) are responsible for detecting and filtering out malicious nodes. Groups that do not adequately punish malicious activity will suffer in reputation across the network, limiting its ability to participate in the network fully. The reputation scheme is used to promote a healthy network setting and can be expanded to aid other network features such as broadcasting network alerts and distributing malware detection signatures.

Through simulation, the performance of SARP Net under common real-world file-sharing conditions was evaluated. The simulation showed that the concept of SARP Net is very feasible at handling the network load of file queries, responses, interaction receipts, file transfers, and reputation messages. In comparison to the anonymous distributed data store Freenet, SARP Net's file transfers were faster by 30.5% with small group size configurations, while its bandwidth utilization could be nearly 3 times as high. It is concluded that the additional

bandwidth usage is the acceptable tradeoff for a more secure, anonymous, data sharing environment.

7.2 Future Work:

This paper's focus on file-sharing should not limit the potential of SARP Net. The protocol is so robust that distributed computing, file storage, and data mining are a few of the extensions that can be added to the system with relative ease. With this said, SARP Net is a work in progress. This paper outlines the basic structure and routing protocol of the anonymous system. Several future enhancements will help booster the performance of the network, increase its security, and perform better analysis on the system. One such enhancement would be a completely decentralized method of verifying and distributing encryption keys. Eliminating the dependence for a long-lived, dedicated server would make the system more fault-tolerant and eradicate the notion of having a single point of failure.

A second future extension would be to expand the SARP-Sim simulator to compare the system with a wider range of similar protocols. The analysis comparing SARP Net with Freenet in these experiments is adequate for initial evaluations but to truly see how well SARP Net compares with other systems, the simulator will need to gain the capability of simulating other systems also. This extension would compare SARP Net to other anonymous P2P systems under the exact same environment with the same workload, providing detailed analysis for comparisons.

APPENDIX: EXPERIMENT RESULTS

Experiment 1's Average Results Over 10 Runs		
	Average	Standard Deviation
Number of initial file queries:	2500.0	0.0
Number of nodes:	4000.0	0.0
Number of file searches performed:	404304.2	5534.481471646644
Number of hits:	225421.8	6013.063492097852
Number of hits returned:	224827.8	6113.911216234662
Average time for first query hit:	7.1145200000000015	0.07365238353237458
Number of repeated messages ignored:	312642.6	12971.762472385934
Number of file requests:	40260.2	1146.696019004165
Number of uploads:	30934.9	886.9357868526898
Number of downloads:	2729.1	172.95863667362784
Number of file fragments transferred:	19174.8	722.4813907638037
Number of reputations created:	2729.1	172.95863667362784
Malicious Activity		
Number of potentially malicious files avoided:	4077.7	435.2344310828361
Number of malicious files transferred:	523.2	52.17815634918505
Number of pending downloads awaiting reputations:	650.2	176.84897511718862
Number of malicious nodes in the network:	799.5	23.070543990118654
Total nodes detected malicious:	115.4	26.306653150866612
Number of malicious nodes detected malicious:	112.4	24.747525128788133
Number of benevolent nodes detected malicious:	3.0	2.0
Transfer Statistics		
Average bandwidth (kB/sec) :	78.0461317626953	1.3858429796927754
Average file transfer speed (kB/sec) :	7.062409066180429	0.26563754380302523
Average time from file request to download complete (sec) :	701.1927336489005	16.678891706989166
Average free bandwidth percentage :	0.793335419448687	0.003682867553909222
Average used bandwidth percentage :	0.2066645805513136	0.003682867553909413

Table 1: Simulation Results Averaged Over 10 Runs of Simulator for Experiment 1.

Experiment 2's Average Results Over 10 Runs		
	Average	Standard Deviation
Number of initial file queries:	2500.0	0.0
Number of nodes:	4000.0	0.0
Number of file searches performed:	246855.8	2648.7307073388943
Number of hits:	130138.7	1399.781557958241
Number of hits returned:	129828.8	1374.8142274503855
Average time for first query hit:	6.62388	0.06584467784111334
Number of repeated messages ignored:	51676.1	2093.5280007680813
Number of file requests:	24006.2	304.74047975285464
Number of uploads:	15265.2	252.31837031813598
Number of downloads:	1938.4	154.00792187416855
Number of file fragments transferred:	7238.3	433.55969600505995
Number of reputations created:	1938.4	154.00792187416855
Malicious Activity		
Number of potentially malicious files avoided:	1517.0	168.7601848778319
Number of malicious files transferred:	352.2	35.66174420860539
Number of pending downloads awaiting reputations:	434.9	79.32395602842814
Number of malicious nodes in the network:	786.8	24.457309745759037
Total nodes detected malicious:	68.8	14.281456508353763
Number of malicious nodes detected malicious:	67.4	13.676256797823008
Number of benevolent nodes detected malicious:	1.4	1.7435595774162698
Transfer Statistics		
Average bandwidth (kB/sec) :	78.65115505371095	2.0156175707428714
Average file transfer speed (kB/sec) :	9.301385378457507	0.48468675630299246
Average time from file request to download complete (sec) :	664.1707861042386	23.97132430102859
Average free bandwidth percentage :	0.8572273556297519	0.003398113633826803
Average used bandwidth percentage :	0.1427726443702491	0.0033981136338265263

Table 2: Simulation Results Averaged Over 10 Runs of Simulator for Experiment 2.

Experiment 3's Average Results Over 10 Runs		
	Average	Standard Deviation
Number of initial file queries:	2500.0	0.0
Number of nodes:	4000.0	0.0
Number of file searches performed:	514797.1	5297.848118812014
Number of hits:	275510.9	5088.118482307581
Number of hits returned:	274562.9	5112.563240684657
Average time for first query hit:	7.2587600000000005	0.07780045244084366
Number of repeated messages ignored:	116688.9	6953.88257666176
Number of file requests:	51414.6	1116.5897366535303
Number of uploads:	32819.4	675.1056509910135
Number of downloads:	711.1	49.50242418306401
Number of file fragments transferred:	8807.0	452.70409761785896
Number of reputations created:	711.1	49.50242418306401
Malicious Activity		
Number of potentially malicious files avoided:	2477.5	262.90311903817343
Number of malicious files transferred:	131.1	14.3
Number of pending downloads awaiting reputations:	913.6	153.19477797888544
Number of malicious nodes in the network:	795.8	28.683793333518494
Total nodes detected malicious:	16.6	4.004996878900157
Number of malicious nodes detected malicious:	16.4	3.878143885933063
Number of benevolent nodes detected malicious:	0.2	0.4000000000000001
Transfer Statistics		
Average bandwidth (kB/sec) :	78.28282055664063	1.124108825079359
Average file transfer speed (kB/sec) :	11.29465568974031	0.4547570667552502
Average time from file request to download complete (sec) :	183.7886331297139	2.179454822931857
Average free bandwidth percentage :	0.8273814241983551	0.003116775009959704
Average used bandwidth percentage :	0.17261857580164444	0.003116775009959661

Table 3: Simulation Results Averaged Over 10 Runs of Simulator for Experiment 3.

LIST OF REFERENCES

- [1] Azureus, "Azureus: Java BitTorrent Client," [Online Document] 2006, Available at <http://azureus.sourceforge.net/>
- [2] S.M. Bellovin, "Security Aspects of Napster and Gnutella", 9th Usenix Security Symposium Presentation, Denver, Colorado, August 2000.
- [3] M. Bergner, "Improving Performance of Modern Peer-to-Peer Services," [Online document] UMEA University, Department of Computer Science, 2003, Available at <http://www.cs.umu.se/~bergner/thesis/html/thesis.html>
- [4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. "Making Gnutella-like P2P Systems Scalable." In Proceedings SIGCOMM, Aug. 2003. ACM Press, New York, NY.
- [5] I. Clarke, O. Sandberg, B. Wiley, T. W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System, " in Proc. International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability, Berkeley, California, Jan. 2001, p.46-66.
- [6] B. Cohen, "Incentives build robustness in BitTorrent," In Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, USA, Jun. 2003.
- [7] DivX, "DivX.com: The official Site of DivX Video the DivX Player and the DivX Codec" [Online Document] 2006, Available at <http://www.divx.com>
- [8] J. Douceur. "The Sybil Attack" In Proceedings of the 1st International Peer To Peer Systems Workshop (IPTPS), March 2002.

- [9] W.K. Edwards, M.W. Newman, J.Z. Sedivy, T.F. Smith, D. Balfanz, D.K. Smetters, H.C. Wong, and S. Izadi, "Using Speakeasy for Ad Hoc Peer-to-Peer Collaboration," In Proceedings of ACM CSCW 2002, Nov. 16-20, 2002, New Orleans, Louisiana, USA.
- [10] T. Gil, F. Kaashoek, J. Li, R. Morris, and J. Stribling, "p2psim: A Simulator for Peer-to-Peer Protocols," [Online Document] Apr. 18, 2005, Available at <http://pdos.csail.mit.edu/p2psim/>
- [11] GNUnet, "GNUnet: GNU's Decentralized Anonymous and Censorship-resistant P2P Framework," [Online Document] May 5, 2006, Available at <http://gnunet.org/>
- [12] Gnutella, "Gnutella Protocol Development" [Online Document] Mar. 14, 2006, Available at <http://www.the-gdf.org>
- [13] R. Janakiraman, M. Waldvogel, and Q. Zhang, "Indra: A Peer-to-Peer Approach to Network Intrusion Detection and Prevention", in Proc. Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE), 2003.
- [14] M. Jelasity, G. Jesi, A. Montresor, S. Voulgaris, "PeerSim: A Peer-to-Peer Simulator" [Online Document] Dec. 5, 2005, Available at <http://peersim.sourceforge.net/>
- [15] S. Kamvar, M. Schlosser, and H. Garcia-Molina, "The EigenTrust Algorithm for Reputation Management in P2P Networks," in Proceedings of the Twelfth International World Wide Web Conference, Budapest, May 2003.
- [16] H. Kim, "Peer to Peer: Pirating of the 21st Century," [Online document] Oct. 2004, Available at <http://www.cs.rutgers.edu/~rmartin/teaching/fall04/cs552/papers/003.pdf>
- [17] Leichtman Research, "Broadband Internet Access & Services in the Home: Research Study 2006," [Online Document] 2006, Available at <http://www.leichtmanresearch.com/research/bband.html>

- [18] MetaMachine, "eDonkey2000 User's Guide," [Online document] 2003, Available at <http://www.edonkey2000.com/documentation/index.html>
- [19] S. Mondesire, P. Wang, G. Hadlett, "PGS Net, " Term paper, University of Central Florida, Apr. 2006.
- [20] M. Parameswaran, A. Sursala, and A. Winston, "P2P Networking: An Information Sharing Alternative," IEEE Computing, Vol. 34 No. 7, Jul., pp. 31-38, 2001.
- [21] R. Rivest, "The MD5 Message-Digest Algorithm," MIT Laboratory for Computer Science and RSA Data Security, Inc., Apr. 1992.
- [22] M. Roos, J. Willemson, and P. Laud, "Improving the Gnutella Protocol Against Poisoning," in Proc.NORDSEC 2003, Gjøvik, Norway, Oct., 2003.
- [23] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems". In Proceedings of Multimedia Computing and Networking, 2002.
- [24] N. Ting, R. Deters, "3LS - A Peer-to-Peer Network Simulator," Third International Conference on Peer-to-Peer Computing, Linköping, Sweden, pp. 212-213, 2003.
- [25] A. Singh, L. Liu, "TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P Systems", Proceedings of the third IEEE International Conference on P2P Computing, Linköping, Sweden, Sept, 2003.
- [26] H. Skogh, J. Haeggstrom, A. Ghodsi, and R. Ayani, "Fast Freenet: Improving Freenet Performance by Preferential Partition Routing and File Mesh Propagation," Cluster Computing and the Grid Workshops, 2006. Sixth IEEE International Symposium, vol. 2, pp. 9, May. 2006.
- [27] V. Vlachos, S. Androutesellis-Theotokis, D. Spinellis, "Security Applications of Peer-to-Peer Networks," Computer Networks, Vol. 45 No. 2, Jun., pp. 195-205, 2004.

- [28] D.S. Wallach, "A Survey of Peer-to-Peer Security Issues," in International Symposium on Software Security, Tokyo, Japan, Nov. 2002. [22] D. Werthimer, J. Cobb, M. Lebofsky, D. Anderson, and E. Korpela, "SETI@HOME - Massively Distributed Computing for SETI," Communications of the ACM, Vol. 45 No. 11, Nov., pp. 56-61, 2002.
- [29] WASTE Development Team, "Waste: Anonymous, Secure, Encrypted Sharing," [Online document] 2003, Available at <http://waste.sourceforge.net/>
- [30] D. Werthimer, J. Cobb, M. Lebofsky, D. Anderson, and E. Korpela, "SETI@HOME - Massively Distributed Computing for SETI," Communications of the ACM, Vol. 45 No. 11, Nov., pp. 56-61, 2002.
- [31] Wikipedia, "Gnutella", [Online Document], June 22, 2006, Available at <http://en.wikipedia.org/wiki/Gnutella>
- [32] D. Zeinalipour-Yazti, "Exploiting the Security Weakness of the Gnutella Protocol" [Online document] Mar. 2002, Available at <http://www.cs.ucr.edu/~csyiazti/cs260-2.html>