Analysis and Design Peer-to-Peer Gnutella Network and Threat Solutions

Dejan Chandra Gope dejan.gope23@gmail.com

Dr. Md. Nasim Akhtar drnasim@duet.ac.bd

Department of Computer Science and Engineering
Dhaka University of Engineering and Technology (DUET)
Gazipur-1700, Dhaka, Bangladesh.

ABSTRACT: Gnutella is a file sharing network that allows users to send and receive files over the Internet. The Gnutella network is a peer-to-peer (P2P) network, which allows users on different networks to share files. Peer-to-peer (P2P) networks have emerged over the past several years as new and effective ways for distributed resources to communicate and co-operate. Peer-to-peer computing is the sharing of computer resources and services by direct exchange between systems. These resources and services include the exchange of information, processing cycles, cache storage, and disk storage for files. P2P networking has the potential to greatly expand the usefulness of the network be it for sharing music and video, privately contracting for services or for coordinating the use of expensive scientific instruments and computers. Some of the networks, such as Napster and Gnutella are created in an ad hoc manner with little or no centralized control. Other P2P networks such as computational and data grids are being designed and implemented in a very structured manner. P2P networks are presenting new challenges to computer security and privacy in a number of ways. This project will explore Analysis and Design Peer-to-Peer Gnutella Network and Threat Solutions. Our primary focus will be analysis the Gnutella network, threat creation and solutions of threats.

Keywords: Peer-to-Peer (P2P) Network, Gnutella, Napster, Morpheus , FreeNet, Distributed Search Solutions (DSS), Gnutella Crawler, Distributed Hash Table (DHT).

1 INTRODUCTION

Gnutella is a file sharing network that allows users to send and receive files over the Internet. The first part of its name comes from the GNU General Public License, which originally allowed the source of the program to be made available to the public. The second part of the name comes from Nutella, a chocolate hazelnut spread, which apparently the developers ate a lot of while working on the project. The Gnutella network is a peer-to-peer (P2P) network, which allows users on different networks to share files. However, each user still must connect to an "ultrapeer," which is a server that lists files shared by connected users. This makes it possible to search for files across hundreds or even thousands of other computers connected to the network. Gnutella is a network protocol, not an actual program.

Although traditional network file systems like NFS provide a reliable way for users on a LAN to pool and share data, Internet-wide file sharing is still in its infancy. Software developers and researchers are struggling to find new ways to reliably, efficiently and securely share data across wide area networks that are plagued by high latency, bottlenecks, and unreliable or malicious nodes. This paper will focus on decentralized file sharing networks that allow free

Internet-wide participation with generic content. A decentralized network has no central authority, which means that it can operate with freely running nodes alone (peer-to-peer, or P2P). Much of the current research into file sharing focuses on such systems, after the repeated failure of commercially-oriented networks such as Napster and Morpheus demonstrated that centralized and purely multimedia-based systems were unsuitable for long-term use by the general Internet public. Building a useful decentralized file sharing network is no small feat, but an effective system will be a remarkable accomplishment for the modern Internet. A robust, massive content distribution network will have a multitude of uses. The huge amount of accessible data (already into hundreds of terabytes on existing networks) and enormous transfer capacity at little or no cost to individual participants demonstrates the value of such a system, which may soon become a core Internet technology akin to the World Wide Web. Such large, anonymous networks seem quite natural for the Internet, as they demonstrate the epitome of pooling resources for the mutual benefit of all users. The first major project to delve into decentralized file sharing was Gnutella, developed by the Nullsoft team (also developers of the Winamp media player). After being posted in 2000, the software was quickly removed from the web site by Nullsoft's owners, America Online Inc., and the plans to release the specification of the protocol were abandoned. Nevertheless, other developers were able to reverse engineer the protocol and publicly released the specification. The publication of a well defined protocol specification (currently Gnutella v0.6) proved to be extremely useful, as different developers were able to contribute their own Gnutella-compliant software that could inter-operate.

2 RELATED WORK

Peer-to-peer systems (P2P) have emerged as a significant social and technical phenomenon over the last year. They provide infrastructure for communities that share CPU cycles (e.g., SETI@Home, Entropia) and/or storage space (e.g., Napster, FreeNet, Gnutella), or that support collaborative environments (Groove). Two factors have fostered the recent explosive growth of such systems: first, the low cost and high availability of large numbers of computing and storage resources, and second, increased network connectivity. As these trends continue, the P2P paradigm is bound to become more popular.

Unlike traditional distributed systems, P2P networks aim to aggregate large numbers of computers that join and leave the network frequently and that might not have permanent network (IP) addresses. In pure P2P systems, individual computers communicate directly with each other and share information and resources without using dedicated servers. A common characteristic of this new breed of systems is that they build, at the application level, a virtual network with its own routing mechanisms. The topology of the virtual network and the routing mechanisms used have a significant impact on application properties such as performance, reliability, and, in some cases, anonymity. The virtual topology also determines the communication costs associated with running the P2P application, both at individual hosts and in the aggregate. Note that the decentralized nature of pure P2P systems means that these properties are emergent properties, determined by entirely local decisions made by individual resources, based only on local information: we are dealing with a self-organized network of independent entities.

These considerations have motivated us to conduct a detailed study of the topology and protocols of a popular P2P system: Gnutella. In this study, we benefited from Gnutella’s large existing user base and open architecture, and in effect use the public Gnutella network as a large-scale, if uncontrolled testbed.

Our measurements and analysis of the Gnutella network are driven by two primary questions. The first concerns its connectivity structure. Recent research shows that networks as diverse as natural networks formed by molecules in a cell, networks of people in a social group, or the Internet, organize themselves so that most nodes have few links while a tiny number of nodes, called hubs, have a large number of links. Finds that networks following this organizational

pattern (power-law networks) display an unexpected degree of robustness: the ability of their nodes to communicate is unaffected even by extremely high failure rates. However, error tolerance comes at a high price: these networks are vulnerable to attacks, i.e., to the selection and removal of a few nodes that provide most of the network's connectivity. We show here that, although Gnutella is not a pure power-law network, it preserves good fault tolerance characteristics while being less dependent than a pure power-law network on highly connected nodes that are easy to single out (and attack).
The second question concerns how well (if at all) Gnutella virtual network topology maps to the physical Internet infrastructure. There are multiple reasons to analyze this issue. First, it is a question of crucial importance for Internet Service Providers (ISP): if the virtual topology does not follow the physical infrastructure, then the additional stress on the infrastructure and, consequently, the costs for ISPs, are immense. This point has been raised on various occasions but, as far as we know, we are the first to provide a quantitative evaluation on P2P application and Internet topology (mis)matching. Second, the scalability of any P2P application is ultimately determined by its efficient use of underlying resources.
We are not the first to analyze the Gnutella network. In particular, the Distributed Search Solutions (DSS) group has published results of their Gnutella surveys and others have used their data to analyze Gnutella users' behavior and to analyze search protocols for power-law networks. However, our network crawling and analysis technology (developed independently of this work) goes significantly further in terms of scale (both spatial and temporal) and sophistication. While DSS presents only raw facts about the network, we analyze the generated network traffic, find patterns in network organization, and investigate its efficiency in using the underlying network infrastructure.

3 DESIGN GOALS OF GNUTELLA

Like most P2P file sharing applications, Gnutella was designed to meet the following goals:
● Dynamic network which allows users to join and leave continuously. This goal has been achieved satisfactorily.
● Scalability this as we have seen above poses a challenge. Considering the fact that this is a basic requirement in any p2p network, this is a serious problem.
● Reliability in the face of external attacks from viruses etc.
● Anonymity which is a basic privacy requirement in any p2p Network

4 GNUTELLA OVERVIEW OF THE PROTOCOL AND ARCHITECTURE

The Gnutella protocol is a peer-to-peer (P2P) overlay network designed for resource sharing across the global Internet. The network is built completely at the application layer, and nodes interact via client programs running on their local machines, irrespective of the underlying physical network. As originally conceived, connectivity, routing, and resource searching are handled in a wholly distributed way, with every node nominally equal to every other (recent upgrades changed this slightly). Any differences in a node's ability stem solely from their own computational, memory, or network bandwidth relative to other nodes.
The Gnutella protocol was originally a very simple protocol, completely specified in a sparse 6-page document. The protocol eventually grew in complexity as the popularity and function exceeded its very simple initial design. Currently, Gnutella developers refer to the original protocol (with minor modifications) as version 0.4, and the next generation protocol, which has absorbed a slew of new features and even wholesale protocol additions, as version 0.6
The sections that follow briefly describe Gnutella in its original incarnation, followed by a description of the relevant parts of the later version.

4.1 Gnutella v0.4 Protocol

The Gnutella protocol consists of five types of messages: ping, pong, query, query hit (the reply to a query message), and push. A ping message is used to discover new nodes on the network. A pong message is sent as a reply to a ping and provides information about a network node, including IP address, port number, and number of files shared. A query message is used to search for files shared by other nodes on the network. It contains a query string and a minimum requested link speed. A query-hit message contains a list of one or more files which match a given query, the size of each file, and the link speed of the responding node Push is used to upload a file to clients behind a firewall who cannot download files themselves.

A node initiates a connection to another via a two-way handshake:

A fiB: GNUTELLA CONNECT/ 0.4 Bfi A: GNUTELLA OK

A and B then exchange Gnutella protocol messages. Each protocol message contains a 23-byte descriptor of the form {id, type, TTL, hops, payload length}. The first field is a 16-byte descriptor number (roughly) unique on the network. TTL is the time-to¬live of the packet on the overlay network, and hops is the number of hops thus far the message has traveled. Each time the message transits a node on the network the hop count is changed. The payload length describes the actual data in the Gnutella packet, if any. This number is crucial, as more than one Gnutella packet can fit in one IP datagram, and there are no breaks in the Gnutella datastream. Lastly, the type describes which of five message types is in the packet, either Ping, Pong, Query, QueryHit, and Push.

Ping messages carry no payload, and are used to explore the network for more neighbors. Upon receiving a ping, a node will decrement (increment) the TTL (hops) field appropriately and pass along the ping to all neighbors except the originating node. That node will also return a pong message containing as payload a 13-byte descriptor
{port, IP address, number of files shared, kBs shared}. Note, pongs are sent back along the network overlay back to the originating node, not directly. Hence, a pinging node ostensibly learns of the existence of all nodes within a radius of one TTL.

When a node decides to find a file or resource, it sends a Gnutella header along with a descriptor {minimum speed, search criteria}. The first field is a two-byte number that lists the minimal connection (in kb/s) of nodes that should respond, followed by the specified search criteria. Nodes who match the criteria respond with a query-hit message of the form {number of hits, port, IP address, speed, result [1…n], host ID}. Each result is a tuple {File Index, File Size, File Name}, with File Index a unique ID issued by the responding node, and File Name some human-readable tag to display on a hit list. The descriptor ID number on the 23-byte Gnutella header must match that of the query packet's header ID number. This allows matching by both the query launcher and all intervening nodes. Hits are sent back via the overlay network to the originating node.

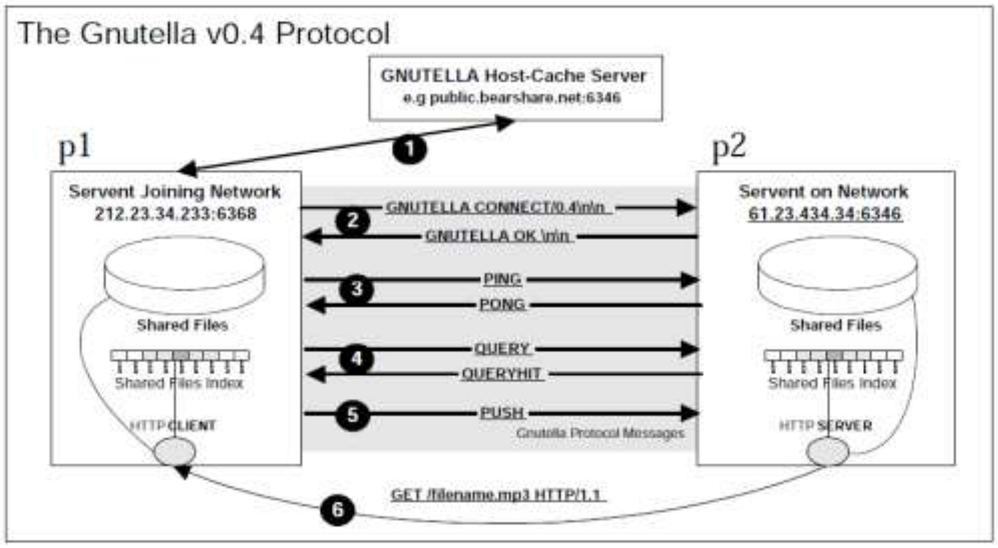

Figure-1: Gnutella v0.4 Protocol

Finally, to access a resource (e.g. a file), the querying node establishes a TCP connection with the responding serving node, and sends an HTTP GET request of the form:
GET/get/<File Index>/<File Name>/HTTP/1.0 \r\n Connection: Keep-Alive \r\n Range: bytes=0-x \r\n User-Agent: Gnutella \r\n
The range field allows for continuing a disrupted download, or parallel downloads from several nodes. Note, the download is outside of the overlay network, and direct between the serving and the downloading nodes.
Finally, in addition to the above four basic message, the protocol supports a PUSH message to allow downloads from firewalled hosts. The query originator, in addition to the Gnutella header, sends {node ID, File Index, IP address, port} to the firewalled node via the network. The node ID is a random unique 16-byte string, and the address and port refer to that of the query origination node. The serving node then starts a TCP connection back to the querying node, along with a string indicating the file in question. With just a standard the TCP connection established (which is allowed by almost every firewall), the querying node then sends the HTTP request and everything proceeds as before.

4.2 Gnutella v0.6 Protocol

With Gnutella's meteoric rise in popularity following the disbanding of Napster, the original protocol soon displayed its inadequacies. Early measurement studies showed that as much as 50% of Gnutella traffic consisted of superfluous pongs flooding the network. Since nodes were regularly looking for neighbors, as well as sending 'keep-alive' pings that signaled their continuing existence to current neighbors, cascades of redundant pongs were choking connections, and impeding searches. In addition, in the early days of custom-written clients, some Gnutella nodes were engaging in anti-social behavior like hammering neighbors with pings or queries, injecting packets with large TTL values, or continuing to forward packets they had already seen. In response, the major developers of clients, plus open source participants, added a series of heuristic modifications to the original protocol, as well as some fairly fundamental changes that are collectively called 'v0.6' (for somewhat obscure reasons). The most significant change concerned network topology is described below.
In an attempt to make Gnutella scalable for mass usage, developers imagined establishing a minimal hierarchy in the Gnutella network. So-called 'supernodes' or 'ultrapeers' would be able to leverage the superior bandwidth of their hardware and handle a larger share of search routing and connectivity, while keeping low-bandwidth nodes from choking traffic via their slow connections. Functionally, each supernode keeps connections open to a set of leaf nodes, and to a number of other supernodes. Leaf nodes themselves keep connections only to their supernode, which handled its traffic to the rest of the network. Hence, the set of high-bandwidth supernodes form a data bus used by the larger network.
By using a header in the handshake messages passed on connection initiation, nodes then negotiate connectivity based on their status: leaf nodes subsume their existence to a supernode, and supernodes collect leaves and inform other supernodes of their existence. In more sophisticated implementations of the idea, leaf nodes pass a file index to their supernode, who then answer all incoming search queries on its leaves' behalf. This reduces supernode-leaf traffic, and shields the leaf nodes from all traffic other than direct download requests, and whatever traffic they themselves generate.

5 GNUTELLA CRAWLER

A Gnutella crawler is a software program used to gather statistic information on the gnutella file sharing network, such as the number of users, the market share of different clients and the geographical distribution of the userbase. Early crawlers used the Ping/Pong messages to discover hosts connected to the network. Although this method is still usable, it is too slow to capture enough data for a topological overview of the gnutella network as it requires initiating full gnutella connections; this involves several roundtrips to perform the header processing. An extension has been added to the gnutella protocol to allow crawlers to quickly access ultrapeers. Right now, there is no public accessible crawler online on the gnutella network, since the last one hosted by Lime Wire LLC has been taken down.
Gnutella2 (G2) also supports crawlers for the gain of statistical data such as the network size or the network composition (clients, versions, usernames and usercountries). Right now, there is only one crawler existing on the G2 network, called g2paranha. It is written and maintained by dcat and licensed under the GPL.

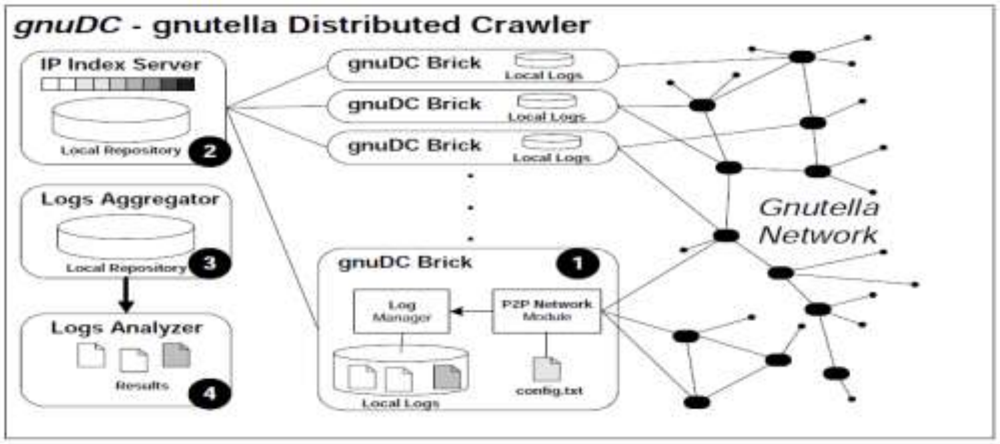

Figure-2: gnuDC - Gnutella Distributed Crawler.

6 THREAT CATEGORIES

A paradigm shift is needed when considering the security of peer-2-peer networks and the associated threat models. In the standard client-server architecture, services are provided by a particular host (or a small group of hosts). Thus, by attacking a specific machine an attacker can subvert, modify or make a service become unavailable. For example, if Ebay's website is successfully attacked, no one will be able to have that service (i.e. take part in auctions) they will have to use some other website. In these cases, services are linked to hosts, attacking a service means attacking a host. With decentralized P2P networks that is no longer true. One can still attack specific hosts in the network, but because the services provided are not (usually) provided a small number of hosts, it is not clear what that would achieve. For example, by attacking a single supernode in the Gnutella network one would not be able to make any single file become unavailable. On the other hand, attacks mounted against the whole network, may try to disrupt a single service while leaving others unaffected. Decentralized P2P networks decouple services from hosts. A similar decoupling is needed when analyzing the security of such systems and different threat models emerge. Threat models are the focus here.

6. 1 Flooding

In many flooding attacks that rely on the interactions of the Gnutella protocol with the TCP/IP protocol stack are considered and discussed at length. The author also implements an attack on his own webserver through the generation of fake query-hit messages. The work presented in also considers the security aspects of the Gnutella network, this time problems with the Gnutella protocol itself independently of any underlying protocols are considered. However, the main focus is the development of a traffic model to deal specifically with query-flood DoS attacks. The simplicity of version 0.4 of the Gnutella protocol leads to some inefficiencies. Perhaps, the most notable of which is the need to broadcast query messages when trying to locate resources. This causes an exponential growth in the number of messages in the network. Version 0.6 of the Gnutella protocol addresses many of these issues, but it also introduces another level of complexity when compared to the very basic 4 messages seen in version 0.4. One of the most significant changes is the introduction of a two-tier system where high-bandwidth nodes help decrease traffic by caching query routing information. However, query communication between supernodes in version 0.6 is essentially the same as in version 0.4, i.e. queries are made by broadcast on the neighboring network graph.

Both the connectivity (ping/pong) and the querying (query/query-hit) functionalities of the Gnutella protocol lay the burden of multiplexing/demutiplexing messages on intervening nodes and hence assume implicit faith in third parties. Nodes are assumed to be well behaved. The design focus is primarily functionality and efficiency, as supposed to security. There has been no attempt to establish in reality how trustworthy these intermediate third parties really are. There are no protocol mechanisms to establish or estimate this and we have been unable to identify any open proposals to use the underlying protocol infrastructure to obtain such information. Attacks consisting of flooding a single type of the 4 basic messages in Gnutella have been considered in the literature. Flooding with reply messages (i.e. pong and query-hit) is thought to be unfruitful as replies are dropped unless a previous matching ping or query was sent over the same network connection previously. Any malicious nodes immediate neighbors would, just by following of the protocol, curtail the efficiency of any such attack. The situation was very different with ping messages because these were propagated through the network. Precisely due to the large amount of traffic ping messages can generate the new ping caching techniques introduced in version 0.6 largely make ping flooding a thing of the past, by seriously limiting if not completely preventing ping messages from propagating beyond immediate neighbors. Query flooding still presents a major threat and is addressed in depth in and can only be minimized not prevented by load balancing. Attacks consisting of a mix of messages have not to our knowledge been considered in depth in the open literature. Whether or not such attacks can be more effective than the simple ones already considered is still unclear. We suggest that the reason query-flooding attacks still present a major concern is a fundamental one. The main functionality provided by the Gnutella protocol is distributed file searching. In a distributed file search the work is spread so that many hosts look for the same files. Thus, the query message requests a service from the network and requires a certain amount of work to be performed. When correctly followed the Gnutella protocol ensures that the load is collaboratively spread amongst the nodes.

Perhaps a framework on the lines of would be useful here. Extra book keeping is arguably the major ingredient that is required to produce an estimate of a node’s reliability, the network overhead maybe minimal. However, there is absolutely no guarantee that the protocol has to be followed and thus is open to abuse. For example, a malicious vendor could sell a Gnutella client that relays “difficult but unyielding” queries only to its competitors, keeping its own clients from that load. In query flooding the abuse is simply to cause too much work, another possibility would be supply erroneous replies to all queries. However, if the network is to provide any functionality the query mechanism must be able to request service. This is an indication that some feedback control should be present on any mechanism that can cause work to be performed. But is not present in the Gnutella protocol. Some proposals to achieve this have been presented (such as hash-cash) and will be discussed later.

6.2 Content Authentication

Whenever a file is downloaded, there needs to be confidence that the contents of the file are what is expected and advertised. In the best case, not only is the file what is expected, but also it is only what is expected. In other words, there is no additional information or capabilities in the file that the user does not know about and probably does not want. Current practice in cooperative P2P networks such as Gnutella relies heavily on good faith trust is trust between the consumer and the provider that the file label corresponds to the unaltered file content. In other words, if you request a particular song title, what is transferred is that exact song and only the song. Unfortunately, there is nothing but good will assuring that “what you see is what you get”. Currently the only way to tell if the file delivered is the file expected is to listen to it and see if it sounds correct. There is little overhead for a user listening to a file, and discard it if it is not what is expected. However, this is not guarantee that the file is intact or unaltered. Besides substituting the expected content with other content, but even more sinister things can occur. Content may be added in ways that are not detectable through listening. An alteration might be harmless, but it may also be able to introduce virus code, messages, and other subterfuge.

6.3 Hijacking Queries

Because of its trust in intermediate third parties, Gnutella is highly susceptible to malicious behavior, as has been demonstrated by the numerous attacks. However, if one considers the new paradigm of attacks where services are targeted as opposed to hosts there has been little work done. In the Gnutella protocol, intermediate parties are able to see a significant share of the queries from all servers within their local subgraph.

What damage could be inflicted upon the network if a node misbehaves and uses query information? Every supernode has the ability to see a large proportion of the queries. Gnutella has a 7 hop query protocol so the query will proceed through as many as 6 supernode hops. If every supernode knows of 4 others, then it is conceivable that almost 1,300 supernodes will see the query. So, essentially, the Gnutella is the logical equivalent of a broadcast Ethernet with more than 1,300 nodes able to listen to and respond any query. It may be unclear how much damage can be inflicted on the network by a node’s ability to listen to much of the query traffic and to response as it wants, but it is certainly clear that anonymity is compromised. Furthermore, even in the best case, this opens the P2P network to other attacks.

7 THREAT SOLUTIONS

7.1 Distributed Hash Tables

Recently, a number of DHT-based P2P networks have been proposed as alternatives to the unbounded searches and anarchic topology of networks like Gnutella and Napster. Almost all such networks depend on a global hash function to map node and file ID's to some logical space, with accompanying connectivity and search procedures to channel queries, provide bounds on queries, deal with node failures, etc.

One of the more intuitive proposals is that of Content Addressable Networks (CAN). The global hash maps to a d-dimensional hypercube, with each node being responsible for some chunk of the hash space. With d=2, our space becomes a sheet and, after several nodes have joined, the network resembles a rough checkerboard, with each square controlled by one node that stores all files whose key (e.g. title or metadata) hashes to its square. The routing information goes only as O(d), but lookup cost is O(d N1/d) where N is the number of nodes in the network. In addition, routing is not very robust to node failure, as the ungraceful departure of neighboring nodes leaves a querying node clueless to its surroundings, and unable to complete it's query. To date, CAN implementations have not made it all the way into a real-world system. The Chord project “aims to build scalable, robust distributed systems using peer-to-peer ideas. The basis for much of our work is the Chord distributed

hash lookup primitive.” Chord arranges its nodes and files on a modular ring, with each node maintaining O(log N) neighbor information on a network of size N. For instance, if m is the number of bits in the node/file identifiers, then the ring extends from 0 to 2m-1.
Each node maintains a table of pointers, where the ith entry contains the identity of the node at least 2i-1 away on the hash ring. Basically, each node possesses a pointer (containing a real IP address) to nodes roughly increasing in powers of 2 away. Individual nodes are responsible for storing files between themselves and the previous node on the ring. Hence, a query begins by consulting its pointer table and tries to find the successor node for the queried identifier on the ring. If no such pointer exists, the query forwards his query to the node closest in the hash space to the identifier. That node will likely have more information concerning nodes in its area, and will either return a correct pointer to the querying node, or forward in turn the query to a closer node. In either case, the distance between the query and the sought-for file or node always decreases by at least a power of two, giving a bound of log N overall for searches.
Such a network, like all P2P networks, is susceptible to the effects of node failure on performance. In the case, of Chord, even fairly catastrophic node failure results in a functioning network, if with a diminished O(n) lookup. Nodes joining the network generate O(log N) traffic to construct their pointer table, as well as a similar communication complexity to update other nodes' tables. Only one transfer and re¬shuffling of files occurs between the entering node and the former successor node for that chunk of hash space. In the background, nodes constantly run a stabilization algorithm that keep them current on routing information and make sure pointers are fresh.
Chord's designers have run simulations modeling its behavior under a variety of conditions. The number of hops required to resolve a query was indeed shown to go as log N, with a mean of 4.5 hops for a network of 1000 nodes (here a hop refers to the number of nodes traversed in searching for an identifier). A simulation of node failure, with nodes failing randomly while queries are underway, showed essentially no network lookup failure. In other words, the percent failed lookups was almost exactly the percent failed nodes, indicating that searches failed due to keys disappearing off the system along with their hosting nodes, and not due to a system failure. Additionally, looking at query failures as a varying function of the rate of node join (and departure), the testers saw a linear dependence of failure on the arrival/departure rate. Specifically, with a large node fail/join rate of 10% per second, only 7% of queries went unanswered, indicating considerable robustness even in the face of dramatic simultaneous failure.

7.2 DHT and Gnutella

While DHT networks can provide an impressive set of features wholly absent from more ad hoc P2P networks, we would not suggest that a DHT-style network should replace Gnutella. It would be unfeasible. The hash assigns file storage based on a random function, not based on what files users already possess. In networks like Napster and Gnutella users share what they have, and it would be impractical to imagine users storing files other than their own. This mean s the host with the hash of the content also has to have the content or be able to get it efficiently. Rather, we would suggest running a DHT style lookup service alongside the going Gnutella v0.6 architecture. Such a scheme would imbue Gnutella with two important properties that it has so far lacked: file permanence and guaranteed lookup.
To realize the importance of these two features, pause and think of security protocols in common computer networks. Most systems either depend on a trusted computing base (e.g. a typical host-client password authentication with an a priori trusted user list), or the ability to read from or write to a global information repository (for example, reputation scores in ebay, or entries in a PKI), as well as the ability to access such memory when needed. Gnutella, as it exists now, possesses none of these building blocks; no nodes are fundamentally trusted, files come and go on the whim of their storing nodes, and findability is determined purely as a tenuous function of topology and network activity at the time of query. Such lack of

functionality limits any security measure to stopgap hacks like HashCash or an inefficient (and probably ineffective) indirection.

With a Chord ring at the supernode level in Gnutella, to take an example, one can imagine leveraging the collective storage to establish a reputation system to rate participating nodes. Thus, nodes known to distribute false or misleading files can be deservedly reported. Search results would then be accompanied with a report on the supplier's reputation, allowing downloader's to avoid disseminators of false files. Likewise, reputation would allow regulation of 'free-loading' on the network, whereby nodes burden the network with searches and downloads, and yet provide no files themselves. Uploaders could preferentially serve users that provide clean reputations, incentivising even casual users to share what they have.

Such an add-on would place an acceptable burden on the supernode layer. Currently, some 90% of supernode traffic stems from queries and query hits, with supernodes having to field their leaf nodes' traffic as well as cross-network traffic for which they are the bridges. Chord lookups on a network with roughly a thousand members (about the size of the supernode graph on Gnutella) take anywhere from 2 to 8 hops. The TTL on most legitimate Gnutella traffic is 7, placing a Chord lookup within the envelope of most going traffic.

Recall also that 7-hop Gnutella queries expand in an exponential flood, while Chord lookups visit only one node per hop, further diminishing Chord's effect with respect to the already existing bandwidth burden on supernodes. Additionally, the increasing efficiency and precision of the Gnutella search architecture following the adoption of yet more v0.6 features, such as the Query Routing Protocol and GUESS (a UDP-based iterative search add-on), will diminish further the rather redundant query traffic at the supernode level, opening up more bandwidth for interactive reputation or authentication protocols.

CONCLUSION AND FUTURE WORK

The social circumstances that have fostered the success of the Gnutella network might change and the network might diminish in size. P2P, however, “is one of those rare ideas that is simply too good to go away”. Despite recent excitement generated by this paradigm and the surprisingly rapid deployment of some P2P applications, there are few quantitative evaluations of P2P systems behavior. The open architecture, achieved scale, and self-organizing structure of the Gnutella network make it an interesting P2P architecture to study. Our measurement and analysis techniques can be used for other P2P systems to enhance general understanding of design tradeoffs. We see two other directions for improvement. First, as argued in efficient P2P designs should exploit particular distributions of query values and locality in user interests. Various Gnutella studies show that the distribution of Gnutella queries is similar to the distribution of HTTP requests in the Internet: they both follow a Zipf's law (note that, although the Zipf's formulation is widely used, these distributions can also be expressed as power-law distributions). Therefore, the proxy cache mechanism used in the Web context might have useful applications in a P2P context. Moreover, when nodes in a dynamic P2P network are grouped by user interest, a query-caching scheme could bring even larger performance improvements. A second direction of improvement is the replacement of query flooding mechanism with smarter (less expensive in terms of communication costs) routing and/or group communication mechanisms. Several P2P schemes proposed recently fall into the former category: systems like CAN or Tapestry propose a structured application-level topology that allows semantic query routing. We believe, however, that a promising approach is to preserve and benefit from the power-low characteristics that, as shown in this paper, emerge in Gnutella’s ad-hoc network topology. A way to preserve the dynamic, adaptive character of the Gnutella network and still decrease resource (network bandwidth) consumption is to use dissemination schemes (e.g., based on epidemic protocols) mixed with random query forwarding. We have collected a large amount of data on the environment in which Gnutella operates, and plan to use this data in simulation studies of protocol alternatives.

REFERENCES

M. Jovanovic, "Modeling Large-scale Peer-to-Peer Networks and a case study of Gnutella", Master's Thesis, University of Cincinati, April 2001.

M. Jovanovic, F.S. Annexstein, and K.A. Berman. Modeling Peer-to-Peer Network Topologies through "Small- World" Models and Power Laws. In TELFOR, Belgrade, Yugoslavia, November, 2001

G. Pandurangan, P. Raghavan, E. Upfal, "Building P2P Networks with Good Topological Properies", Brown University, 2001

Kelsey Anderson, "Analysis of the Trac on the Gnutella Network", University of California, San Diego CSE222 Final Project, March 2001.

M.Ripeanu, "Peer-to-peer Architecture Case Study: Gnutella Network", Technical Report, University of Chicago, 2001.

N. Daswani, H. Garcia-Molina “Query-Flood DoS Attacks in Gnutella”,9th ACM Conference on Computer and Communications Security, Nov 18-22 2002,Washington DC, USA.

Matei Ripeanu, Ian Foster, and Adriana Iamnitchi,Mapping the Gnutella Network:Macroscopic Properties of Large Peer-to-peer Systems. IEEE Internet Computing, vol. 6, no. 1, Jan-Feb 2002.

Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble, “Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts,” Multimedia Systems Journal, vol. 8, no. 5, Nov.2002.

Subhabrata Sen and Jia Wang, “Analyzing Peer-To-Peer Traffic Across Large Networks,” IEEE/ACM Transactions on Networking, vol. 12, no. 2, pp. 219–232, Apr. 2004.

E. Adar, B. Huberman, Free riding on Gnutella, First Monday, Vol. 5-10, October 2, 2000. The Gnutella protocol specification v4.0.

S. Saroiu, P. Gummadi, S. D. Gribble, A Measurement Study of Peer-to-Peer File Sharing Systems, University of Washington Technical Report UW-CSE, July 2001.

K. Sripanidkulchai, The popularity of Gnutella queries and its implications on scalability, February 2001.

Documentation on v0.4 and the major parts of v0.6 can be found at http://www.limewire.com/index.jsp/deve

I. Gupta, K. Birman, P. Linga, A. Demers, R. van Renesse, \Kelips: building an eÆcient and stable P2P DHT through increased memory and background overhead," in 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), 2002.

The Gnutella protocol specication v 0.4, Document revision 1.2, www.clip2.com, 2003.

Q. He, M. Ammar, G. Riley, H. Raj and R. Fujimoto, \Mapping Peer Behavior to Packet-level Details: A Framework for Packet-level Simulation of Peer-to-Peer Systems," in Proceedings of MASCOTS 2003, 2003.

Dejan Chandra Gope is a student in the Department of Computer Science and Engineering at the University of Dhaka University of Engineering and Technology, Bangladesh. He received Bachelor of Science in Computer Science and Engineering (B.Sc) from the University of Dhaka University of Engineering and Technology in 2012. He is now studying Master of Science in Computer Science and Engineering (M.Sc) at the same University. His research interest lie in studying the role of computer vision and advanced human-computer interaction, Artificial Intelligence and Pattern Recognition.