Analysis of Peer-to-Peer Network Security using Gnutella

Dimitri DeFigueiredo±, Antonio Garcia+, and Bill Kramer*

±Department of Computer Science, University of California at Davis: defigueiredo@ucdavis.edu
+Department of Physics, University of California at Berkeley: agm@SIMS.Berkeley.EDU
*Department of Computing Sciences, University of California at Berkeley and the National Energy Research Scientific Computing Center, Lawrence Berkeley National Laboratory: kramer@nersc.gov

Abstract

Peer-to-peer (P2P) networks have emerged over the past several years as new and effective ways for distributed resources to communicate and cooperate. "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 the security issues raised by P2P networks by studying an example on an extreme point on the design spectrum: Gnutella [Gnu]. Our primary focus will be the Gnutella network, which is the *de facto* standard for large, loosely structured, P2P networks. The Grid is a very large-scale, heterogeneous, formally structured P2P network that spans many organizations to make "virtual systems" of resources. [OGSA] It is quickly becoming the standard for distributed resource allocation for high-end computer and instrumentation systems. This paper demonstrates that for P2P networks with *ad hoc* structure, significant security concerns persist.

What are Peer-to-peer Networks

"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."[P2PWG] A broad definition of P2P includes the client server mode of computing, as well as exchange directly amongst clients or amongst servers. However, P2P now also is used to describe some new uses of computers and networking. In particular, it is becoming more common for systems to play both the server and client roles simultaneously. P2P networking now is being used to present new services and functions. P2P is more than just the universal file-sharing model popularized by Napster. According to the Peer-to-peer working group, business applications for P2P computing fall into a handful of scenarios.

- Collaboration: Geographic distributed individuals and teams create and manage real-time and off-line collaboration areas in a variety of ways. The goals are

typically increased productivity and decreased costs.

- Edge services. Edge services move data closer to the point at which it is actually consumed acting as a network caching mechanism. This helps deliver services and capabilities more efficiently across diverse geographic boundaries. A current example is Akamai for an enterprise
- Distributed computing and resources. Using networks and computers, P2P technology can use idle CPU power and disk space, allowing businesses to distribute large computational tasks and data across multiple computers. Results can be shared directly between participating peers. Prioritized use of the resources, even if they are not idle, is possible. Examples here range from the seti@home to the Distributed Teragrid.
- Intelligent agents. Provides ways for computing networks to dynamically work together using intelligent agents. Agents reside on peer computers and communicate various kinds of information back and forth. Agents may also initiate tasks on behalf of other peer systems.

Formally and Loosely Structured Peer-to-peer Networks

The most commonly known P2P networks are those associated with music sharing. First made popular by Napster, a centrally managed P2P network, and now represented by Gnutella and Kazaa, these P2P networks are designed to be loose structures and highly dynamic. Gnutella, Kazaa and others are designed intentionally to have no central control or authority so they are entirely self-organizing. The loose federation of continuing dynamic organization of these networks presents very challenging security issues, especially as the network expands. The focus on decentralization represents a current trend in P2P systems and many see it as the stepping-stone to the extended functionality these systems may provide in the future. This is one of the main reasons this work focuses on the Gnutella network, Gnutella's distributed non-homogeneous architecture make it a suitable test bed to observe how new ideas may affect future P2P networks. However, to put Gnutella's architecture into perspective we must be aware of what other approaches have been taken.

The Grid

There are many networks emerging for e-commerce and scientific efforts that have a more formal structure. One excellent example of this type of P2P network is what is commonly called "the Grid". Grid technology is a collection of tools and services that facilitate the building and managing of "virtual" systems that integrate distributed, heterogeneous, multi-organizational resources on demand. [Grid] These resources might include the different computing and data systems operated by a supercomputer center like NERSC, as well as a diverse collection of user-controlled computing and data systems and scientific instruments. The "Grid" is a research effort (~10 years in all) whose principle initiators are Ian Foster (ANL) and Carl Kesselman (Cal Tech). The initial implementations centered on Globus and did demonstrations (1995-96) of single applications running on geographically distributed, large parallel machines– essentially co-scheduling CPUs by human agreement and management. The concepts and software have evolved dramatically since and has expanded to supported large scale data movement, collaborative work tools, and much more. There are several "Grids" moving for experimental to "production" status as other projects now use grid tools as reliable infrastructure for their science and

engineering. Examples of production or near production grids are the NSF Teragrid, NASA Information Power Grid, DOE Science Grid, Grids for Physics and the EuroGrid. There is an organization that is like the IETF for setting grid standards called the Global Grid Forum. GGF has brought corporate and research communities together to work on Grid implementations. This effort was given a major critical mass when IBM, ANL and CalTech jointly proposed the Open Grid Service Architecture (OGSA) about 8 months ago. The joint effort (with about 50 developers) is aimed at integrating the best features of Globus (and associated tools) with IBM's Websphere technology and is turning the grid effort from one of creating virtual organizations with resources to one of creating a distributed service system that is for "modern enterprise and interorganizational computing environments" [OGSA]. The end goal for industry is to create move to providing "on-demand computing" services rather than computing hardware. Sam Palmisano – CEO of IBM – is quoted as saying grid computing is the "is the most important imitative IBM has undertaken since the Internet"[BW].

Current Grids are mostly based on services from the Globus and Condor software packages, and emerging data Grid and Web based portals. Globus services provide a standard way to define and submit jobs, manage the code and data associated with those jobs, and locate and monitor the available resources across geographically and organizationally dispersed sites. They also provide a consistent set of security services based on X.509, PKI, or Kerberos authentication, proxy certificates to carry the user authentication to remote resources, and a set of secure communication primitives based on the IETF GSS-API: secure telnet, remote shell, and secure ftp are provided by using these services. Condor-G provides job management on top of the Globus services, ensuring that one or more associated jobs that might run on remote resources execute once and only once. Both Globus and Condor services provide for communicating with remote jobs, etc.

A sophisticated set of Grid data services is being developed by the NSF GriPhyN and EU DataGrid projects for managing massive data sets in support of the global high energy physics community. Over the next few years these will provide for cataloging, querying, accessing, and managing replication, location, and movement of very large data sets from a worldwide collection of data sources. Grid portal work at half a dozen institutions is defining and building the Web services and primitives that will provide all Grid services though the user's Web browser. Advanced services that will provide for brokering, co-scheduling, advance reservation of CPU capacity, network bandwidth, and tertiary stored data availability are currently being developed. Collaboration services are also being developed that provide for secure distributed collaboration group management, messaging, versioning and authoring, and the definition and management of "virtual organizations." These services are being integrated with the basic Grid services, frequently through Web Grid services.

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 [GNU]. 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. They are necessarily brief but more detail can be found at [Gnu] and [GDF].

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 →B: GNUTELLA CONNECT/ 0.4
B→ 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.

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.

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.

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.

Flooding

In [Yaz], 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 [Das] 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 (supernodes) 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[1].

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 [Das] 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. 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.

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

[1] Perhaps a framework on the lines of [GN2] 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.

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.

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 described in [Yaz]. 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.

Threat Solutions

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 $O(d\ N^{1/d})$ 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] 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 2^{m-1}. Each node maintains a table of pointers, where the i^{th} entry contains the identity of the node at least 2^{i-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.

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 means 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.

Authentication, Encryption and FFTs

It is not clear what benefits could be achieved by the use of authentication or encryption in the current Gnutella framework. Both authentication and encryption primitives assume “trusted” endpoints to establish communication channels, these are simply not present in Gnutella right now. A querying host has *a priori* no idea of who might answer the query. Without the existence of an underlying “trust or reliability framework”, other mechanisms (like sampling a few possibilities) may be more fruitful.

A simple example is to imagine a modification to the Gnutella protocol to try to stop censoring of query replies in the Gnutella network. We do this by having replies be sent straight to the IP address of the querying host end we also establish end-to-end encryption/authentication. Obviously, there is no mutual authentication, as it is impossible to know who will reply. Thus, any malicious intermediate node can still censor the reply by mounting a man-in-the-middle attack. The use of cryptographic primitives relies on the existence of an underlying “trust infrastructure” such as a private key infrastructure.

One way to prevent content alteration is to have a trusted third party verify the content through mechanisms such as digital signatures. This is a proven mechanism and can be implemented in a straightforward manner. However, there are limitations to this approach. First, this is a much more centralized approach. Every resource or item of content would have to have a digital signature, meaning the overhead for a peer to provide a resource becomes higher. Every down load would have to contact the central authority to get the key for the signature. This approach may be useful for some P2P applications, particularly ones where there is either a high cost associated with the risk of an altered resource. Another instance where

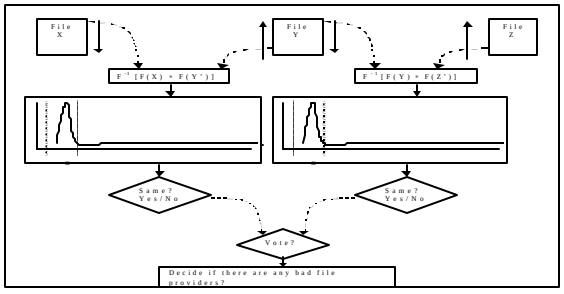
digital signatures may be acceptable is when the P2P network is relative modest or limited in scale, for example when a P2P network is applied to business-to-business transactions. But, for very large, dynamic networks like Gnutella that have tremendous numbers of participants, it may not be feasible nor desirable to have any part of such a network rely on a centralized resource.

Another more generalized approach is to acquire several copies of a file from different sources. Gnutella already provides multiple sources for the requestor to choose from if more than one source has the file. Once multiple copies are acquired, various types of checks can be made to determine if they are the same. Once there is the ability to compare the contents of multiple files, it becomes possible to implement a voting or selection scheme. Depending on the confidence needed to assure altered files is detected; a requestor can use well known methods such as straight voting. For example, if two out of three files are judged to be equivalent, then the minority file is considered bogus. A more robust selection method could be implemented by following an algorithm such as described as the "Byzantine Generals Problem" [Lam] which will guarantee the correct outcome. The basic algorithm works if there is a small number of treacherous nodes relative to the overall number of nodes. If the number of treacherous nodes might be large, then the query originator would have to have multiple rounds of communication with all the participants to determine which are truthful. There are multiple implementations to solve the problem, each with increasing cost (in both messages and computation) based on the needs of the query originator. In either case, if the files are not identical, then the consumer has options ranging from rejecting the dissimilar file to ignoring all other responses from the resource that sent the bad file. Once it is possible to determine truthful and untruthful hosts, it is also possible to send a message to the reputation framework discussed above to either enhance or detract from a node's reputation.

A more fundamental question is how to compare files in order to have some substance to select from. In some cases, a straight checksum may be viable. A checksum requires two issues that possibly are non-optimal. The first issue is that K (with K> 1) copies of the file have to be transferred to the receiver, using multiple times the bandwidth and storage. Either the entire file has to be transferred K times, or a large part of it. If only parts of the file are used, more problems come into play in at least the Gnutella case. Consider two music files from two different sources. While containing effectively the same music, it is highly unlikely they are bit-wise identical since they were created on two different CDs, maybe at two different times, from two different manufactures and with two difference manufacturing processes. All this will cause checksums to be different. The CDs may have different errors and timing. There may be different lengths of idle time before the music or after the music. Thus to use partial file checksums, firm conventions would have to be put into place for all files that may require the cooperation of the originators of the file. The second and more critical issue is in order for checksums to be the same; the files have to be bit for bit identical. Some P2P applications do want absolutely identical files bit for bit. However, a number, including Gnutella would not be able to use a checksum because equivalent files are not bit for bit identical. A more flexible solution is to use Fourier Transforms to compare the files. The approach is to take any two files X and Y out of the set of M files, and perform a Fourier Transform on each. Given the Fourier Transform F, one can calculate F(X), and F(Y'), where Y' is the file Y, written in reverse order. Then taking the inverse Fourier transform of the product of F(X) and F(Y') provides a signal profile whose characteristics are like Figure 1. The sharpness and height of the peak of the signal will indicate how likely it is the files

agree. The location of the peak along the x-axis indicates when the files start agreeing. The sooner the peak appears, the earlier the files agree. For equivalent files, a very steep, high peak near the x origin would be expected. The concept is to perform pair-wise comparisons on N copies of the file, where N will vary based on the needs of the consumer.

Figure 1 – A schematic diagram of using Fourier Transforms and a selection mechanism to decide if a files has been altered even if the files are not bit wise identical.



The FFT scheme has several advantages. First, it can be used in all cases – whether the files are bit wise identical or not. Second, the entire file does not have to be present. FFTs can be blocked – operating on segments of the file – and then blocks combined if it is necessary to test the entire file. It is likely for a network like Gnutella only the first parts of the file are needed. Using part of the file has the dual advantage of saving bandwidth and computation. That savings in bandwidth could be avoided all together or used to acquire more copies of the file to improve the accuracy of the checking and to prevent an attacker from overwhelming the number of good files by sending many copies of bad files. These choices could be tuned based on the network characteristics of the receiver. A variant of the above scheme is to take the file content and pass it through a hash function to provide a value. Then, based on the hash function and the size of the hash space, two files whose value is identical may beconsidered equivalent.

Thus, for a cost of bandwidth and/or computation, it is possible to verify the authenticity of a file's contents and to detect and eliminate untrustworthy peers to an arbitrary degree. While not foolproof – an attacker can in theory launch so many responses that it would have high probability the consumer would select more bogus files than correct files, this is unlikely.

Proposal for an experiment

In the following section, we propose a measurement experiment on Gnutella network. Since such an experiment, and the attack it hopes to emulate, depend intimately on graph of the Gnutella network, we spend the next sections explaining Gnutella network structure. A description of the experiment itself follows.

Gnutella topology

It has recently been observed that a multitude of networks in both the natural and human worlds follow a power-law distribution, where the probability of a node having degree k goes as $P(k) \sim k^{-a}$. Here, a generally ranges from 1.2-4, and is a global parameter describing the network. Such disparate sets as World Wide Web pages, authors in science publications, Hollywood movie actors, and human sexual contacts are described by power-law networks of varying a. Gnutella, with its *ad hoc* and chaotic connection architecture, also possesses such a structure (see below figure).

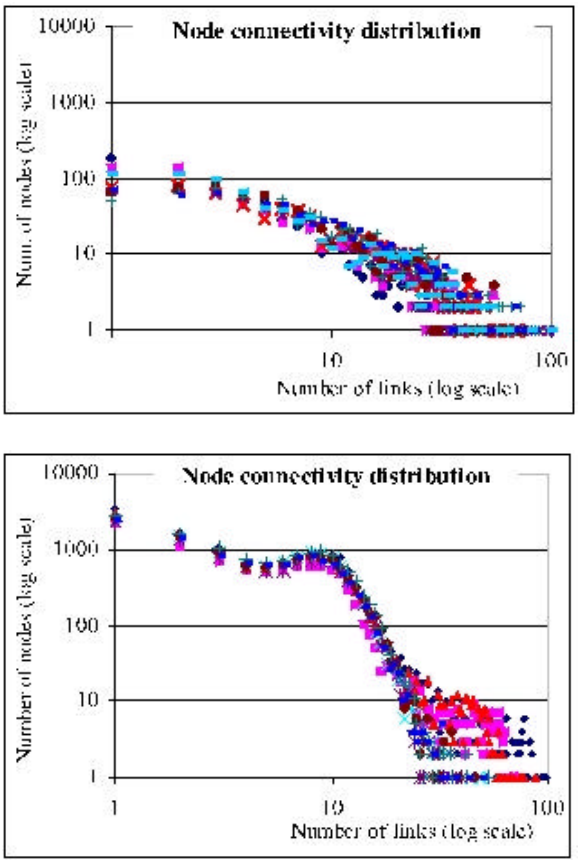


Figure 2 *Two crawls of the Gnutella network, from [Rip]. The first above was taken November 2000, the second below March 2001.As can be seen, the topology of Gnutella is in rapid transition. In between the two crawls, LimeWire and BearShare, two leading Gnutella clients, instituted the two-tier hierarchy of nodes, as described in previous sections. The sudden change around link number 10 reflects the existence of two communities of nodes: leaf nodes connected to one or few supernodes, and the supernodes connected to a brood of leaf nodes as well as any number of other supernodes. Essentially, the Gnutella network is splitting into two superimposed networks, with the supernode graph retaining power-law properties but at a different a constant.*

The reasoning behind why such uncoordinated phenomenon as web page authoring and human sexual behavior show such general scaling stands some enquiry. As Albert and Barabasi [Bara1] showed, large-scale structure is determined by how entering nodes attach to the growing network. Specifically, if nodes attach to a node *i* with probability proportional to $k_i/\Sigma_I k_i$, (where k_i is the degree of the *i*th node) a power-law graph results. The phenomenon is summarized by the phrase "the rich get richer", reflecting how the time rate of change of connectivity is proportional to the instantaneous connectivity, allowing nodes that get slightly ahead of the game to rapidly outpace their less well-connected colleagues.

In Gnutella, nodes join by contacting well-known nodes that in turn provide a new node with the contact information for yet other nodes. While the technical details of node joining are slightly complex (as well as obscure where the protocol specification is ambiguous), it seems safe to say that well-connected nodes, due to their existence being known to many nodes, are probably disproportionately represented in node-join handoffs. This fact, combined with a base of users that occupy a spectrum from occasional users to hardcore Gnutella stalwarts, leads to the variation in connectivity indicated in previous figures. It seems safe to say that Gnutella is now, and will remain for the near future, a power-law network of possibly varying α parameter.

Graph properties

Such networks demonstrate many interesting properties, most notably a small network diameter, or short average path length between nodes. This leads to the well-known 'small-worlds' effect, whereby any two nodes in even large networks are connected by a relatively small number of hops. As previously explained, searches on Gnutella are propagated via undirected floods on a network with uncertain topology and constantly changing node population. Despite these harsh conditions, searches in Gnutella are remarkably successful,

precisely due to this short-range property of the graph. Whether by accident or design, the power-law structure has saved Gnutella from total breakdown, even when the network outgrew its humble beginnings to become a global network moving terabytes of data over tens of thousands of machines. Any proposed changes or improvements to the protocol should preserve the fundamental network properties that have made Gnutella so long-lived.

Another large-scale property of power-law networks is that of robustness under node failure. As modeled by [Bara2], power-law networks exhibit different behavior under the effects of either random or targeted node loss, particularly compared to random graphs. Cohen et. al. [Coh] found that the proportion of nodes p that could fail, without splitting a power-law graph of parameter a into disconnected components was limited by:

$$p \leq 1 + (1 - m^{a-2} K^{3-a} \frac{a-2}{3-a})^{-1}$$

Here is p is the proportion of failing nodes, m is the lowest possible degree, and K the maximum possible degree.
Taking α=2.4 from earlier (and perhaps now dated) measurement studies, and setting m=1, K~30, we arrive at a critical proportion p_c of about .7. Hence, upwards of 70% of Gnutella nodes can fail and the graph remains topologically connected.
This tremendous robustness is necessary in the face of occasional physical network failures, and far more frequent node disappearance, where users simply turn off after downloading. Statistics show [Sar] that 50% of Gnutella nodes spend no more than 60 minutes online at any given time, indicating a constant churn of online hosts.
Against more targeted node failure however, a power-law network shows considerable disadvantages. Since a minority of nodes constitutes a majority of the connectivity, removing those few nodes has dreadful consequences on the network.

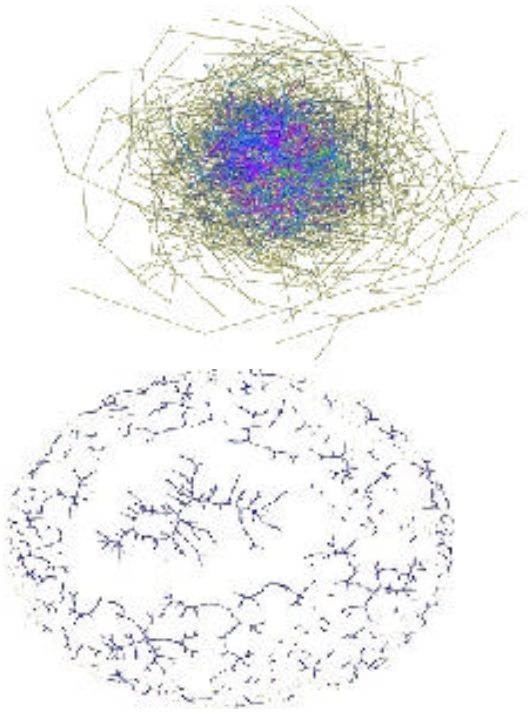


Figure 3 *Two network plots of the Gnutella network, from February 16, 2001, spanning 1771 nodes [Sar]. The first contains all nodes and their accompanying connections. In the second, the top 4% connected nodes have been removed. The resulting network is fragmented into multiple pieces.*

A final interesting property of power-law networks refers to their epidemic properties. Networks are often used as models of disease-propagation among a community of carriers, with nodes representing potential infected organisms and edges modes of transmission. Quantitative epidemiology has developed a classic SIR model (for susceptible/infected/removed, the three states nodes can be in) describing the extent and speed of outbreaks given certain disease transmissibility. Recently several researchers have extended the model, examining the effect of underlying topology on outbreak strength. Pastor-Satorras and Vespignani [Ves] have claimed to have analytically proven that scale-free power-law networks allow diseases to reach epidemic proportions even with low transmission rates between nodes. Newman

[New], using both analysis and simulations, shows that power-law graphs of certain a exhibit epidemic behavior no matter how low the transmissibility between nodes. The arguments of both authors are lengthy and complex, but an intuitive insight can be had by analogy to searches on the graph. Power-law searches work so well as queries very quickly reach a well-connected node, which in turn propagates the query far and wide. Likewise, even one infection on a power-law graph likely reaches a well-connected node, which spreads it to everyone else.

Experimental design

It was our intention to design a measurement experiment that simulated a denial-of-service (DOS) attack against the Gnutella network. As seen above, the topology of the Gnutella network imbues the graph with several interesting properties, some of which might be weapons in the hands of attackers.

For example, the susceptibility of power-law graphs to targeted node attack would open the door for a determined adversary to shutdown the backbone of high-degree users that make Gnutella work. Such an adversary could employ network-level attacks against such hosts and cut off their network access, or simply emulate such nodes himself, pulling out at the last second after achieving a massive connectivity.

It would be difficult to perform such attacks in an academic context, as they may be called morally (if not legally) into question. A better strategy may well be to use the epidemic qualities of power-law graphs to distribute unwanted content, and that way undermine Gnutella's fantastic file-sharing potential. Such unwanted content could take the form of random or garbage files, or more pointed content like a virus. Virus spreading on Gnutella has already been documented [Vir], although the virus in question only caused damage when run from the local machine and had no way of taking over the Gnutella client itself. However, such a shortcoming will only last until the first security hole in a well-distributed Gnutella client is discovered.

Assuming for the moment clients are safe, one can still imagine virus-like files that would be of interest to the Gnutella hacker. For example, major copyright holders like the Motion Picture Association of America (MPAA) or the Recording Industry Association of America (RIAA) could collude with major makers of mp3 players, like Microsoft's Media Player or RealMedia's RealPlayer, such that players would only play specially marked mp3's a limited number of times (such proposals have already been made in the context of digital-rights management). Copyright holders (and their hired IT guns) would then exploit Gnutella to spread such specially tagged files, making sure any downloaded file was of only very limited utility.

While this seems counterproductive to the cause of the MPAA/RIAA, in that the aggrieved party is itself distributing content, a more circumspect analysis reveals its power. Viruses are successful if they infect many hosts. Infectious agents that are too virulent inevitably cause outbursts that burn themselves out (e.g. the Ebola virus, that incapacitates victims within days and kills them within a week). Viruses like HIV, for example, are much more effective, as they are just as inexorably terminal (essentially all people infected with HIV develop AIDS and die), but they allow their hosts a good 5-10 years (or more, with the use of modern anti-retrovirals) to spread the virus to others. Likewise, the current strategy of the (modest) efforts of anti-piracy agents to spread garbage files when queried for copyrighted content seems ineffective by comparison. Any given Gnutella search returns dozens of hits, and if a download fails due to a false file, the user just picks another uploader. Such tactics will only dissuade the most casual and finicky P2P user.

Much more effective would be to drown the entire namespace of interest (e.g. all queries containing 'The Beatles') with marked and incapacitated files. The user who downloads such a file will not know of its infected state at first, as the file plays normally, until it reaches the play limit imposed by its creators and enforced by the media player. In the meanwhile, other users have downloaded the file themselves, who in turn propagate it, and quickly every file on the system is incapacitated in this way. It seems much more irritating if what is at first a normal and impressive mp3 collection suddenly refuses to play overnight, requiring the slow process of rebuilding it, then simply having to try one or two downloads at first to find a copy of a certain file.

The topology of Gnutella will certainly aid in such an attack, as discussed above. A potential attacker need only spread the file to a relatively small collection of high-connectivity nodes, who will in turn disseminate it to the rest of the network. If wisely done, such an attacker would not even need vast resources to complete his task.

The Experiment

Fortunately, the technology of music compression allows us to model such an attack without need to disrupt network service. The mp3 standard allows an arbitrary amount of data to be appended before the first compressed block of a music file. Such data is typically parsed by mp3 players to provide file metadata like artist name, recording, etc. to the user looking at the player's GUI. Such a data field can be modified at will, and clients uploading and downloading the file (generally) maintain the integrity of prepended tag. To our knowledge, however, no client uses the tag information to perform matches on incoming queries, and the Gnutella protocol provides no direct way of advertising the contents of the tag to outside queriers. Hence, the file's ebb and flow in the network should be quite independent of whatever is written in its tag, but the tag should survive along with the file's contents as the file circulates.

Our experiment thus involves two stages: in the first, we inject files into the network with music files marked with a unique random nonce as their mp3 tag. In the second, we probe the network, and see to what extent our marked file permeated. Our second probing step will be repeated at regular intervals after, and perhaps even during, the injection phase.

Our hope is to use this 'release-and-catch' methodology to quantify the speed and extent of content distribution on Gnutella, something until now unexplored. In addition, this serves as an excellent model of the DOS attack detailed above. Whether Gnutella's future attackers intend to simply distribute false content, or employ the more subtle strategy described here, such an experiment will reveal the magnitude of success one could achieve with a slow-burning virus that gives its hosts plenty of time to spread.

Operationally, this experiment will involve custom coding a Gnutella client to act as tagger and distributor. Such a client will have to achieve high connectivity with the supernode level of Gnutella hosts, getting a rapid picture of the network's most well connected nodes. Prior to client initialization, we will choose a piece of the file namespace as our virus vector. Intuition indicates that the best piece of namespace would be for files in high demand, with large numbers of hosts repeatedly making searches. This way, we will be able to quickly hijack the namespace, pre-empting non-marked files' entry to the network with our own marked version. As a guide, one can use the 'Top Sellers' list on Amazon.com to determine a well-frequented part of the namespace on a daily basis.

Hence, once situated on the network, and with a target namespace in mind, our client selectively responds to queries emanating from selected 'elite' high-connectivity supernodes. The cut-off for inclusion in the elite category can be made dynamically, or simply as a result of an ongoing crawl (e.g. within the current top 5% of connected supernodes). The goal is to limit our upload of files only to a subset of the supernodes, and hence that way parameterizes our later results on file dissemination. That is, on each inject-and-probe cycle we chose to distribute to a greater or lesser fraction of supernodes, and measure the effect of greater or lesser file injection on the file's circulation.

In the probe phase, one can adopt two philosophies. One is that of an omnipotent and omniscient network god, trying to discover the presence of every relevant file and whether it has been tagged or not. Essentially, probing everyone at the supernode level, downloading upon receiving a hit, and tallying the percentage permeation of tagged files versus non-tagged.

While such a tack might be ideal, given the tremendous bandwidth use that Gnutella produces, it may well be unfeasible, even for fast, high-bandwidth academic machines (see below problems section).

A smarter measurement may be to simply emulate a normal Gnutella user and gauge what effect the tagging 'hack' has on what such a user sees. Thus, following injection, we would re-connect as lowly leaf nodes and search repeatedly for files in the namespace. Downloads on those search hits would reveal their tag, and would allow us to quantify success as degree of permeation on normal user downloads, rather than on the network as a whole.

Progress and Problems

For the past month, development has been underway for just such a custom client. Our initial code base is the Jtella API, a set of Java libraries for network applications on Gnutella. Jtella numbers some 5000 lines, but we have been obliged to change or add another 2000 lines for even basic functionality. Jtella was written almost two years ago, before the introduction of v0.6, and contains the basic, low-level code for Gnutella connectivity in the form of ping/pong messages, persistent TCP connections, etc. The full-blown experimental client will have to contain support for v0.6 features (already done), as well as the high-level algorithm for maintaining placement in the Gnutella graph. This latter feature may prove to be the most difficult. Initial attempts at connectivity reveal the Gnutella network to be a highly dynamic and changing environment. Accurate global knowledge of the instantaneous topology may prove elusive, if not impossible, and our file injection may occur in an opaque fog of network information.

In addition, certain properties of commercially written clients tend to produce fragmentation in an otherwise uniform Gnutella network. For example, it is known that BearShare allows its supernodes to accept only other BearShare leaf nodes (recall that the client vendor is passed along with everything else in the v0.6 handshake), producing a so-called 'BearShare cloud' in the network. Our initial connectivity experiments indeed showed BearShare nodes only responding to handshakes with 'BearShare' in the vendor field. Depending on whether this behavior continues (BearShare claims it will change it) our client may well have to spoof a series of vendor names to achieve connectivity over the entire graph.

Administrative Issues

In order to run the experiment as designed, it was necessary to investigate the regulations relating to the use of copyrighted material and other issues regarding Gnutella. The University of California has at least two governing documents that apply to this. Copyright policy is related to the 1998 Digital Millennium Copyright Act. The university document is "Guidelines for Compliance with the Online Service Provider Provisions of the Digital Millennium Copyright Act" [DMCA]. This policy defines what is possible to do with copyrighted material in the university context, include what UC considered fare use. The second policy is the UC Berkeley IT policy, which controls what activities are appropriate when using campus computers and networks. The IT policy refers to the DCMA guidelines for how copyrighted material should be handled.

The project team approached UC Office of the President's General Counsel and described the experiment and its issues. We also explained why we thought it was appropriate and useful to carry out the experiment. Further issues were discussed regarding the privacy of network users and the fact there would be no obvious impact on the other users of the network.

After approximately 6 weeks of discussions, the University gave approval for the experiment. We do plan to pursue the experiment at a later date.

Findings

Ad hoc peer-to-peer networks, represented in this study by Gnutella, have significant security issues. Previous work focused on v0.4 of the network document many ways the network can be compromised. The changed of v0.6, which are intended to allow growth to a very large scale introduce more vulnerabilities, because v0.6 turned the network into a "power law" system where very few nodes have most of the control and indeed most of the content. While leaving the network unstructured, v0.6 has the affect of making the query structure more hierarchical. Thus attacks focused on the relatively few supernodes make the Gnutella network much more susceptible to compromise.

Gnutella is basically not securable in its current implementation. Several solutions were investigated to improve security on the network, such as point-to-point encryption, authentication and Distributed Hash Tables. Some of these are shown to be untenable in the current Gnutella implementations. It needs additional structure and capability in order to withstand even a modest level of attack. This is important since it is clear the music industry is planning concerted efforts to disrupt Gnutella services.

This paper introduces the idea that security can be view not just from a host or network perspective but also from a "service" perspective. In the Gnutella case, this means looking at attacks on the namespace and file content rather on host per se.

We have introduced several solutions that may be used to increase security and privacy. The use of Fourier Transforms to check content in file that are not bit-wise identical enables would enable the query originator to determine if the content in a file is what is expected. Coupled with a selection mechanism, it becomes feasible to resist flooding and query interception attacks.

Finally, an experiment was designed and coded to investigate the actual performance of a network and to explore the feasibility of some of the exploits discussed. This experiment could not be performed until legal and administrative issues were resolved within the university pertaining to the investigation of P2P networks that exchanged copyrighted material. We received approval for the experiment, but only two days before the submission

deadline. However, the effort to gain the approval will not be wasted since it will allow the experiment to go forward and be discussed at a later time, and possibly allow other groups to investigate P2P networks with their own experiments.

Summary

Despite the apparent need to make Gnutella more secure for users, the feeling among its open-source developers seems to be one of indifference. What 'state-sponsored' hacking there has been on Gnutella is relatively small, with the network retaining most of its functionality. Also, the pursuit of Gnutella users by the MPAA or RIAA has been limited to relatively high-profile cases of people either sharing enormous numbers of files or sharing clearly pirated copies of copyrighted (and perhaps then unreleased) content. The average Gnutella user sharing and downloading a handful of files has for the moment little to fear from the large lobby groups and their blackmailed ISPs. If the day arrives that file-sharing becomes impossible due to a deluge of false content, or hijacked queries, or when users receive immediate cease-and-desist emails the moment their Gnutella clients blink on, then perhaps Gnutella developers, some of whom have commercial interests in the survival of the network, will start to pay security serious attention. For the moment, most developers seem content with improving Gnutella to the point of competing with currently more successful networks like Kazaa, which enjoys an order of magnitude greater usage statistics than Gnutella.

Finally, it needs to be noted that many of the issues discussed in this document are specific to Gnutella, Kazaa and other very loosely formed networks. Other P2P networks, including the Grid are designed with much more emphasis on security and will in all probability have few, and certainly different, issues.

References

[Bara1] Emergence of Scaling in Random Networks, Albert Barabasi and Reka Albert, *Nature*, **286**, 509.

[Bara2] Error and attack tolerance of complex networks, R. Albert, H. Jeong, and A.L. Barabasi. *Nature*, **406**, 378.

[BW] Ante, Spencer, "Computing Power sold like electricity", Businessweek, November 11, 2002.

[CHORD] http://www.pdos.lcs.mit.edu/chord/

[Coh] R> Cohen, K. Erez, D. ben-Avraham, S. Havlin, Resilience of the Internet to Random Breakdowns. *Phys.Rev.Lett*. **85**, 4626.

[Das] 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.

[DCMA] Guidelines for Compliance with the Online Service Provider Provisions of the Digital Millennium Copyright Act, November 17, 2000 – UCOP General Counsel's Office

[GDF] http://groups.yahoo.com/group/the_gdf/

[GNU] several websites have introductory information, e.g. www.bearshare.com, www.limewire.com.

[Grid] http://www.gridforum.org, or, The Grid: Blueprint for a New Computing Infrastructure by Ian Foster (Editor), Carl Kesselman (Editor)

[Gro] C. Grothoff "GNUnet – An Excess

Based Economy" available at: http://www.gnu.org/software/GNUnet/

[Lam] The Byzantine General Problem, Leslie Lamport, Robert Shostak and Marshall Pease, ACM Tansactions on Programming Languages and Systems, Vol. 4, No. 3, July 1982, Pages 382-401.

[NERSC] http://www.nersc.gov/aboutnersc/pubs/Strategic_Proposal_final.pdf

[New] The spread of epidemic disease on networks, M.E.J. Newman. Sante Fe Institute working paper. http://www.santafe.edu/sfi/publications/Working-Papers/02-04-020.pdf

[P2PWG] http://www.peer-to-peerwg.org/whatis/index.html
[OGSA] http://www.gridforum.org/ogsi-wg/drafts/ogsa_draft2.9_2002-06-22.pdf
[Sari] A Measurement Study of Peer-to-Peer File Sharing Systems, Stefan Saroiu, P. Krishna Gummadi and Steven D. Gribble. Appeared in MMCN 2002.

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

[Sar] A Measurement Study of Peer-to-Peer File Sharing Systems, Stefan Saroiu, P. Krishna Gummadi, Steven D. Gribble. In *Proceedings of the Multimedia Computing and Networking (MMCN)*, San Jose, January, 2002.

[v4] Documentation on v0.4 and the major parts of v0.6 can be found at http://www.limewire.com/index.jsp/developer. Further documentation of new and proposed features is at [GDF].

[Vir]http://www.commandsoftware.com/virus/gnutella.html

[Yaz] D. Zeinalipour-Yazti "Exploiting the Security Weaknesses of the Gnutella Protocol" Course project for "Seminar in Computer Security" at the University of California –Riverside, Computer Science, March, 2002.