
ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



Semester Thesis
Attacks on Peer-to-Peer Networks

Baptiste Prêtre
bap@student.ethz.ch

Prof. Dr. Roger Wattenhofer
Distributed Computing Group

Advisors: Stefan Schmid

Dept. of Computer Science
Swiss Federal Institute of Technology (ETH) Zurich
Autumn 2005

Abstract

In this thesis, we collect information about known attacks on P2P networks. We try to classify them as well as study the different possible defense mechanisms. As a case study, we take Freenet, a third generation P2P system, which we deeply analyze, including simulating possible behaviors and reactions. Finally, we draw several conclusions about what should be avoided when designing P2P applications and give a new possible approach to making a P2P application as resilient as possible to malicious users.

Contents

1	Introduction	3
1.1	Peer-to-Peer Network Definition	3
1.2	Historical	3
1.3	Future and Vulnerability	4
1.4	Thesis Organisation	4
2	General Attacks and Defences	6
2.1	DOS Attacks	6
2.1.1	Defenses	6
2.2	Man-in-the-middle Attack	7
2.2.1	Defenses	8
2.3	Worm Propagation	8
2.3.1	Defenses	9
2.4	The Human Factor	9
3	Specific P2P Attacks and Defenses	11
3.1	Rational Attacks	11
3.2	File Poisoning	12
3.2.1	Defenses	12
3.3	Sybil Attack	13
3.3.1	Defenses	13
3.4	Eclipse Attack	14
3.4.1	Defenses	15
4	First conclusions	16
4.1	Only Pure P2P!	16
4.2	Reputation-based Systems	16
4.3	Randomization	17
5	Case study: Freenet	18
5.1	Design	18
5.2	Protocol Overview	18
5.3	Protocol Details	19
5.3.1	Keys	19

5.3.2	Retrieving Data	20
5.3.3	Storing Data	21
5.3.4	Managing Data	21
5.3.5	Adding Nodes	22
5.4	Facts	22
5.5	Attacks	22
5.5.1	DOS Attack I	22
5.5.2	Malice and Eavesdropping	23
5.5.3	Anonymity	24
5.5.4	More-than-just-routing Attacks	24
5.5.5	Simulation	25
5.5.6	DOS Attack II	27
5.6	Future	27
6	Final Conclusions	29
6.1	Concluding Weaknesses	29
6.2	Our Solution	30
6.2.1	Observations	30
6.2.2	PGP, Web of Trust and Darknets	30
6.2.3	P2GP	31
6.3	Conclusion	32

Chapter 1

Introduction

1.1 Peer-to-Peer Network Definition

Throughout this thesis we will study peer-to-peer networks, henceforth we will use the acronym P2P. A P2P network is a network that relies on computing power of its clients rather than in the network itself [1]. This means the clients (peers) will do the necessary operations to keep the network going rather than a central server. Of course, there are different levels of peer-to-peer networking:

- **Hybrid P2P:** There is a central server which keeps information about the network. The peers are responsible for storing the information. If they want to contact another peer, they query the server for the address.
- **Pure P2P:** There is absolutely no central server or router. Each peer acts as client and server at the same time. This is also sometimes referred to as “serverless” P2P.
- **Mixed P2P:** Between “hybrid” and “pure” P2P networks. An example of such a network is Gnutella which has no central server but clusters its nodes around so-called “supernodes”.

1.2 Historical

Although P2P networking has existed for quite some time, it has only been popularized recently and will probably be subject to even bigger revolutions in the near future.

Napster was the first P2P application which really took off. The way it worked was quite simple: a server indexed all the files each user had. When a client queried Napster for a file, the central server would answer with a list of all indexed clients who already possessed the file.

Napster-like networks are known now as **first generation** networks. Such networks didn't have a complicated implementation and often relied on a central

server (hybrid P2P). The central server model makes sense for many reasons: it is an efficient way to handle searches and allows to retain control over the network. However, it also means there is a single point of failure. When lawyers decided Napster should be shut down, all they had to do was to disconnect the server.

Gnutella was the second major P2P network. After Napster's demise, the creators of Gnutella wanted to create a decentralized network, one that could not be shut down by simply turning off a server. At first the model did not scale because of bottlenecks created whilst searching for files. FastTrack solved this problem by rendering some nodes more capable than others. Such networks are now known as **second generation** networks and are the most widely used nowadays [1].

Third generation networks are the new emerging P2P networks. They are a response to the legal attention P2P networks have been receiving for a few years and have built-in anonymity features. They have not yet reached the mass usage main second generation networks currently endure but this could change shortly. Freenet is a good example of a third generation P2P network, that is the reason why we will study it more deeply during this thesis.

1.3 Future and Vulnerability

Some futurists believe P2P networks will trigger a revolution in the near future. The ease of use, the huge choice and finally the low price (often free) have been the main reason for the explosion of file-sharing applications over the past years. Add to this the fact that internet connection speeds are steadily increasing, the arrival of newer faster algorithms (Caltech's FAST algorithm was clocked 6,000 times faster than the internet's current protocol) as well as the incapacity to control or monitor such networks. This P2P revolution simply means huge quantities of data will be available almost instantly to anybody for free.

This, of course, is disturbing news for many industries (music, movie, game...) as P2P networks provide an alternative way of acquiring many copyrighted products. These industries have very actively been waging war against "digital piracy" for a decade soon. The results of this war are controversial but as P2P networks have never stopped growing during this period of time, it is acceptable to think that they will steadily grow on and gain even more importance in the future. I encourage interested readers to consult Scott Jensen's white paper [3] on the subject.

1.4 Thesis Organisation

This thesis will now be organised in 4 main sections:

- 1 First, we will look at several vulnerabilities or attacks found in general networks.

2 We will then look at more specific attacks specially designed for P2P networks.

After these two analysis, we will try to draw some first conclusions. We will then proceed to our case study: Freenet.

3 We will thoroughly describe the Freenet structure.

4 Finally, we will try to find potential weaknesses in Freenet and ways to improve them.

After this we will draw our final conclusions and explore possible new directions.

Chapter 2

General Attacks and Defences

2.1 DOS Attacks

A Denial-Of-Service attack is an attack on a computer or a network that causes the loss of a service [1]. There exist many forms or methods to perpetrate a DOS attack. In the case of P2P networks, the most common form of a DOS attack is an attempt to flood the network with bogus packets, thereby preventing legitimate network traffic. Another method is to drown the victim in fastidious computation so that it is too busy to do answer any other queries.

DOS attacks are far more efficient if multiple hosts are involved in the attack, we then speak of a DDOS attack (distributed denial-of-service) [14]. In a DDOS attack, the attacking computers are often personal computers with broadband connections that have been compromised by a virus or trojan. The perpetrator can then remotely control these machines (qualified as zombies or slaves) and direct an attack at any host or network.

Finally, a DDOS attack can be even further amplified by using uncompromised hosts as amplifiers. The zombies send requests to the uncompromised hosts and spoof the zombies' IP addresses to the victim's IP. When the uncompromised hosts respond, they will send the answering packets to the victim. This is known as a reflection attack.

2.1.1 Defenses

The first problem is detecting a DOS attack as it can be mistaken with a heavy utilization of the machine. DDOS attacks using reflection are extremely hard to block due to the enormous number and diversity of machines a malicious user can involve in the attack (virtually any machine can be turned into a zombie). In addition, as the attacker is often only indirectly involved (he attacks through the zombies and the reflective network), it is often impossible to identify the

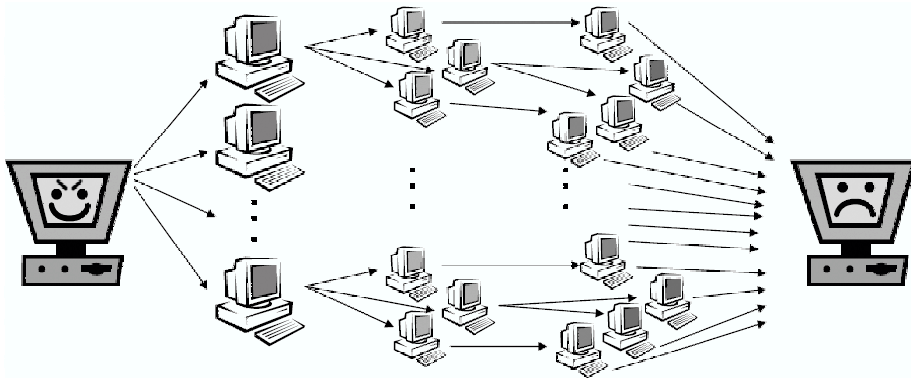


Figure 2.1: A DDoS attack: The attacker sends the order to the computers he personally controls (masters) which then forward it to the zombies, which DOS as many machines as possible and spoof their IP to be the victim's, who will receive all the replies.

source of the attack. Because of these factors, there exists no general way of blocking DOS attacks.

A widely used technique to hinder DOS attacks is “pricing”. The host will submit puzzles to his clients before continuing the requested computation, thus ensuring that the clients go through an equally expensive computation. DOS attacks are most efficient when the attacker consumes most of his victim's resources whilst investing very few resources himself. If each attempt to flood his victim results in him having to solve a puzzle beforehand, it becomes more difficult to launch a successful DOS attack. “Pricing” can be modified so that when the host perceives to be under an attack, it gives out more expensive puzzles, and therefore reduces the effect of the attack. Although this method is effective against a small number of simultaneous attackers, it more or less fails against very distributed attacks. Other drawbacks are that some legitimate clients, such as mobile devices, might perceive puzzles too hard and/or would waste limited battery power to them.

2.2 Man-in-the-middle Attack

In a man-in-the-middle attack, the attacker inserts himself undetected between two nodes. He can then choose to stay undetected and spy on the communication or more actively manipulate the communication. He can achieve this by inserting, dropping or retransmitting previous messages in the data stream. Man-in-the-middle attacks can thus achieve a variety of goals, depending on the protocol. In many cases it is identity spoofing or dispatching false information. Man-in-the-middle attacks are a nightmare in most protocols (especially when there is a form of authentication). Fortunately, they are less interesting in P2P

networks. All the nodes have the same “clearance” and the traffic’s content is shared anyway which makes identity spoofing useless. If the P2P application supports different clearances between nodes, then the implications of man-in-the-middle attacks would depend on the protocol itself. Possible attacks could be spreading polluted files on behalf of trusted entities or broadcasting on behalf of a supernode.

2.2.1 Defenses

Without a central trusted authority, which generally do not exist in P2P networks, it is not possible to detect a man-in-the-middle attack. Nodes have no information about their neighbors and have no way of being able to identify them later with certainty. Fortunately, as man-in-the-middle attacks are mostly useless in P2P networks, this is not very alarming news.

2.3 Worm Propagation

Worms already pose one of the biggest threats to the internet. Currently, worms such as Code Red or Nimda are capable of infecting hundreds of thousands of hosts within hours and no doubt that better engineered worms would be able to infect to reach the same result in a matter of seconds. Worms propagating through P2P applications would be disastrous: it is probably the most serious threat.

There are several factors which make P2P networks attractive for worms [13]:

- P2P networks are composed by computers *all running the same software*. An attacker can thus compromise the entire network by finding only one exploitable security hole.
- P2P nodes tend to interconnect with many different nodes. Indeed a worm running on the P2P application would no longer lose precious time scanning for other victims. It would simply have to fetch the list of the victim’s neighboring nodes and spread on.
- P2P applications are used to transfer large files. Some worms have to limit their size in order to hold in one TCP packet. This problem would not be encountered in P2P worms and they could thus implement more complicated behaviors.
- The protocols are generally not viewed as mainstream and hence receive less attention from intrusion detection systems.
- P2P programs often run on personal computers rather than servers. It is thus more likely for an attacker to have access to sensitive files such as credit card numbers, passwords or address books.
- P2P users often transfer illegal content (copyrighted music, pornography ...) and may be less inclined to report an unusual behavior of the system.

- The final and probably most juicy quality P2P networks possess is *their potentially immense size*.

Once worms finish propagating, their goal is usually to launch massive DDOS attacks (W32/Generic.worm!P2P, W32/SillyP2P, ...) against political or commercial targets (whitehouse.gov, microsoft.com, ...).

2.3.1 Defenses

Before considering any technical defense, there must be a sensitization of P2P users. Leaving a personal computer unattended without a complete firewall and anti-virus on a broadband internet connection is begging for trouble. Blaster, for example, exploited a vulnerability 5 days after it was made public by Microsoft with a “Security Update” that fixed it.

A solution would be for P2P software developers not to write any bugged software! Perhaps that is a far fetched goal, but it would be better to favor strongly typed languages such as Java or C# instead of C or C++, where buffer overflows are much easier to compute.

Another interesting observation is that hybrid P2P systems have a vulnerability pure P2P systems do not. By making some nodes more special than others (for example better connectivity for Gnutella’s supernodes) the attacker has the possibility to target these strategic nodes first in order to spread the worm more efficiently later on. Pure P2P does not offer such targets as all nodes have the same “importance”.

Finally, it is interesting to note the operating system developers are also offering some solutions. OpenBSD’s 3.8 release now returns pseudo-random memory addresses. This makes buffer overflows close to impossible as an attacker cannot know what data segment he should overwrite [15].

2.4 The Human Factor

The human factor should always be a consideration when security is at issue. We previously saw that the upswing P2P applications have experienced is also due to ease of installation and use, the low cost (most of the time free) and its great rewards. Even novice users have little difficulty using such applications to download files that other users shared intentionally or accidentally shared on the P2P network.

This is yet another security problem P2P applications are posing. Empowering a user, especially a novice, to make choices regarding the accessibility of their files is a significant risk. Because of its convenient and familiar look, applications such as Kazaa can cause a user to unwittingly share the contents of his documents or even worst, his whole hard disk.

Unfortunately, novice users do not understand the implications of their inaction with regard to security. Simply closing the application for instance isn’t enough as most of them continue running in the background. Remarkably, millions of P2P peers are left running unattended and vulnerable for large periods of

time. Malicious users with intermediate hacking skills can take advantage of such situations.

Chapter 3

Specific P2P Attacks and Defenses

We will consider two different planes of attack in this section: the data plane and the control plane. Attacking the data plane means attacking the data used by the P2P application itself, for example by poisoning it or rendering it in any way unavailable. On the other hand, attacking the control plane means directly attacking the functionality of the P2P application, trying to render it slower or as inefficient as possible. This is generally done by using weaknesses in the routing protocol. Depending on the attacker's goal, he will choose to attack in one plane or the other, or both.

These two planes are not completely independent. For instance by attacking on the data plane and corrupting many files, users will tend to download more instances of a file thus slowing down the traffic which is typically the aim of a control plane attack. Vice versa, eclipse attacks which are in the control plane can render data unaccessible, which is the primary objective of a data plane attack.

The possibilities of attacks are enormous in P2P networks. Now follows an analysis of the most common attacks as well as some appropriate defense mechanisms.

3.1 Rational Attacks

For P2P services to be effective, participating nodes must cooperate, but in most scenarios a node represents a self-interested party and cooperation can neither be expected nor enforced. A reasonable assumption is that a large fraction of P2P nodes are rational and will attempt to maximize their consumption of system resources while minimizing the use of their own.

For example nodes might realize that by not sharing, they save precious upload bandwidth. In the case of copyrighted material, file sharing can have worst outcomes. As it is illegal and quite easy for authorities to find out who is sharing

specific files, it can lead to a very big fine. These are good enough reasons to motivate nodes in becoming “self-interested”. If a large number of nodes are self-interested and refuse to contribute, the system may destabilize. Successful P2P systems must be designed to be robust against this class of failure.

3.2 File Poisoning

File poisoning attacks operate on the data plane and have become extremely commonplace in P2P networks. The goal of this attack is to replace a file in the network by a false one. This polluted file is of course of no use.

It has been reported [7][8][9], that the music industry have massively released false content on P2P networks. Moreover, companies such as Overpeer¹ or Ret-snap² publicly offer their pollution-based services to the entertainment industry as a way for protecting copyrighted materials.

In order to attack by file poisoning, malicious nodes will falsely claim owning a file, and upon a request will answer with a corrupt file. For a certain amount of money, Overpeer or Retsnap will release huge amounts of fake copies of a file on their servers. Moreover, all messages passing through malicious node can be poisoned (similar to a man-in-the-middle attack). These factors may give the poisoned file a high availability, making it more attractive to download the true file.

3.2.1 Defenses

Although file poisoning attacks sound pretty dangerous, we will argue they do not pose a threat to P2P networks [6]. The main problem is that P2P applications are often set in the background. When a polluted file is downloaded by a user, it stays available for a while before being inspected and cleansed. After a period of time, all polluted files are eventually removed and the authentic files become more available than the corrupted ones. The reason file-poisoning attacks are still successful today are due to 3 factors:

- clients are unwilling to share (rational attack).
- corrupted files are not removed from users machines fast enough.
- users give up downloading if the download seemingly stalls.

These 3 factors each give advantage in different ways to the most available file, which probably is the polluted file at the beginning. Simulations show these factors tend to greatly slow down the removal of polluted files on the network. [16]

¹www.overpeer.com

²www.retsnap.info

3.3 Sybil Attack

Sybil attacks are part of the control plane category. The idea behind this attack is that a single malicious identity can present multiple identities, and thus gain control over part of the network.[10]

Once this has been accomplished, the attacker can abuse the protocol in any way possible. For instance he might gain responsibility for certain files and choose to pollute them. If the attacker can position his identities in a strategic way, the damage can be considerable. He might choose to continue in an eclipse attack, or slow down the network by rerouting all queries in a wrong direction.

3.3.1 Defenses

Unfortunately, without a central trusted authority, it is not possible to convincingly stop Sybil attacks [10]. Maybe carefully configured reputation-based systems might be able to slow the attack down, but it will not do much more. Indeed, once the attacker has legally validated a certain amount of identities, he can validate the rest.

A good defense is to render a Sybil attack unattractive by making it impossible to place malicious identities in strategic positions. We have already seen that structured P2P networks are more resilient to worm propagation. For the same reasons it is a good defense mechanism here, as an attacker will not be able to place his identities where he wishes. Randomly dispersed malicious identities are far less dangerous than strategically placed ones, especially if the P2P network is of considerable size.

Another proposition could be to include the node's IP in it's identifier. A malicious node would thus not be able to spoof fake identities as he would be bound to a limited number of IPs and could be noticed and denounced if he created more identities. Yet this solution is far from simple as other attacks are rendered possible, such as generating fake identities for other nodes and then accusing them of being malicious. This is why we will not consider this defense as it adds a layer of complexity to the existing protocol whilst generating other potential weaknesses.

Several papers propose a central trusted authority as a solution, as well as a complicated public-private key based protocol [11]. Each node should sign his messages, and respond to a challenge by the authority every now and then. It is clear that an attacker simulating many identities would need enormous resources in order to be able to answer all the challenges periodically submitted to each of his identities. While this certainly tries to solve the problem, it is unsatisfactory: this solution breaks the P2P model by reintroducing a centralized point of failure, which can easily be attacked.

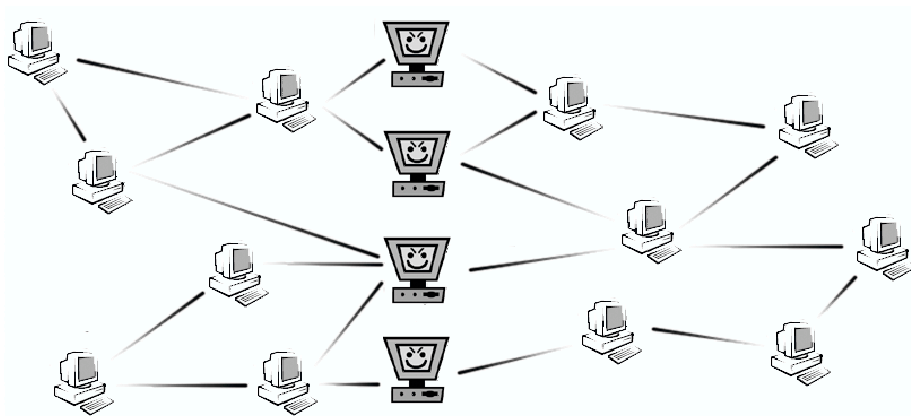


Figure 3.1: An Eclipse Attack: the malicious nodes have separated the network in 2 subnetworks.

3.4 Eclipse Attack

Before an attacker can launch an eclipse attack, he must gain control over a certain amount of nodes along strategic routing paths. Once he has achieved this, he can then *separate* the network in different subnetworks. Thus, if a node wants to communicate with a node from the other subnetwork, his message must at a certain point be routed through one of the attacker's nodes. The attacker thus "eclipses" each subnetwork from the other. In a way, eclipse attacks are high-scale man-in-the-middle attacks.

An Eclipse attack can be the continuation of a Sybil attack. In this case, the attacker will try to place his nodes on the strategic routing paths. We argued before, that man-in-the-middle attacks don't pose a great threat to P2P networks. However, such a high scale attack involving strategic targeting is very serious. The attacker can completely control a subnetwork from the other subnetwork's point of view.

If an attacker manages an Eclipse attack (it is not an easy attack), can attack the network in a much more efficient manner.

- He can attack the control plane by inefficiently rerouting each message.
- He can decide to drop all messages he receives, thus completely separating both subnetworks.
- He can attack the data plane by injecting polluted files or requesting polluted files on behalf of a innocent nodes and hoping, these files are cached or copied along the way.

3.4.1 Defenses

As against man-in-the-middle attacks, very carefully chosen cryptographic protocols may be a good attempt to stop such an attack. Pricing could also help against the Sybil attack version. The problem with such solutions is that they constitute a serious slow-down and harm the scalability of the network.

The main defense against Eclipse attacks is simply to use a pure P2P network model. An even better solution would be to additionally use a randomization algorithm to determine the nodes' location (as for example in Freenet). If the nodes in a pure P2P network are randomly distributed, then there are no strategic positions and an attacker can't control his nodes' positions. It would be nearly impossible to separate two subnetworks from one another in such conditions.

Chapter 4

First conclusions

4.1 Only Pure P2P!

We have now been introduced to P2P networks and have observed most possible attacks. So what are the first conclusions we can make at this point?

First of all, when designing a P2P network, it is of utmost importance not to use a mixed P2P model. As soon as we enter any kind of notion of hierarchy, we automatically present a target. If a node is more important, more trusted or better connected than other nodes, then an attacker can use this to his advantage. This permits malicious users to attack the the network strategically, which is far more dangerous. If there is absolutely no hierarchical structure, then the network presents no strategic targets because of it's uniformity.

Paper [4] studies for example the effects of super-nodes have on worm propagation in Gnutella. In Gnutella, normal nodes connect to supernodes which are in turn connected to each other, acting as a kind of "highway". It is shown [19] they play a significant role in the worm propagation in the network, even without being specifically targeted at the beginning. What better target to launch a Sybil attack then such supernodes? Of course, pure P2P is much harder to implement and also slower than the hierarchical approach: the implementation of node querying is easy if all nodes sign in on a central server.

4.2 Reputation-based Systems

This condemnation of all hierarchical structures also makes us reject reputation-based systems. Nodes in such systems have a "reputation" determined by all other nodes [17]. Typically, each node will publish a list of the nodes it trusts, making it impossible for a node to change its own reputation by itself. Before initiating a download, a node will first check the reputation of the node it wants to download from and then decide whether to pursue or not. In a sense, the higher the reputation, the more importance a node has.

While this might seem like a good direction, we will argue that, as it introduces

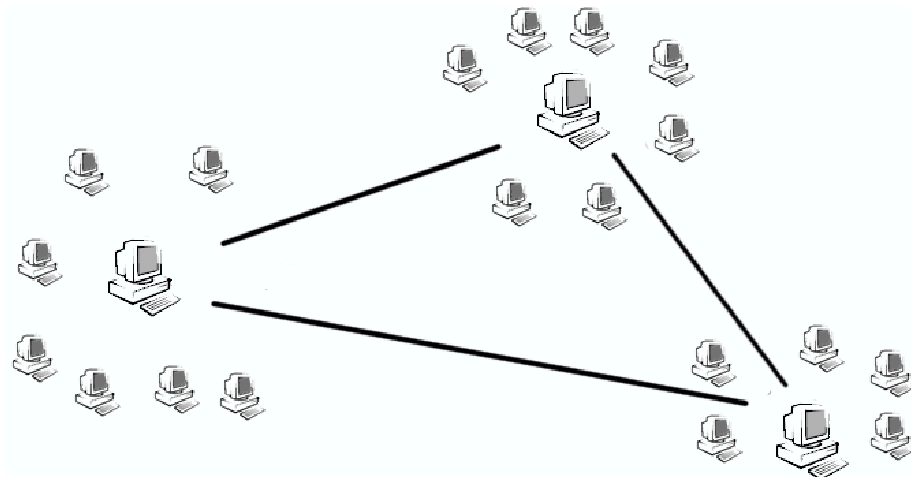


Figure 4.1: Gnutella supernodes.

a notion of hierarchy, this approach constitutes a weakness. The problem is that nodes with a bigger reputation have more powers than other nodes. Other nodes will tend to trust them more and they are able to influence other nodes' reputation more effectively. An attacker simply needs a little patience and wait for one of his nodes to gain sufficient trust in order to launch his attack. If the attacker deploys many malicious nodes as it is often the case, they can give each other a high reputation making them all trustworthy. Finally, other famous nodes constitute strategic targets as they will be able to spread the attack for efficiently.

4.3 Randomization

A last interesting observation is randomization (for example in the routing protocol or in the nodes' location). Indeed, studies show that randomization has a big impact on attacks as an attacker cannot deterministically attack the network any longer. Unfortunately, it also considerably slows the network down. P2P networks often have scalability problems and anything which slows performance down is generally avoided. This is probably the main reason why randomization is avoided in P2P networks.

Chapter 5

Case study: Freenet

5.1 Design

Freenet is an enhanced open source implementation of the system described by Ian Clarke's July 1999 paper "A distributed decentralized information storage and retrieval system" [2] and is classified as a third generation P2P application. A first version was released in March 2000.

Freenet was designed to answer privacy and availability problems second generation applications currently experience. It was built in order to achieve following 5 requirements:

- Anonymity for both producers and consumers of information.
- Deniability for storers of information.
- Resistance to attempts of third parties to deny access to information.
- Efficient dynamic storage and routing of information.
- Decentralization of all network functions.

We will now study it in more detail to see how it manages to meet such requirements.

5.2 Protocol Overview

Intuitively, Freenet can be seen as a "chained" network. Like a link in a chain, each node can only communicate with its direct neighbors. When a node wants to query a file, it sends the message to the most promising neighbor, which will in turn also forward it to its most promising neighbor. Once a message is sent, a node has no way of finding out what will happen to it. It cannot tell to which node the neighbor will in turn forward the message to, or even whether the message is directly answered by the neighbor itself. What's more, a node

receiving a message cannot tell if this message originates from this neighbor or if it is merely forwarding a message received by a previous neighbor. This is the cornerstone idea of the Freenet architecture which guarantees anonymity.

5.3 Protocol Details

A Freenet transaction begins with a Request.Handshake message from one node to another, specifying the desired return address of the sending node. If the remote node is active and responding to requests, it will reply with a Reply.Handshake specifying the protocol version number that it understands. Handshakes are remembered for a few hours, and subsequent transactions between the same nodes during this time may omit this step.

All messages contain a randomly-generated 64-bit transaction ID, a hops-to-live limit, and a depth counter. Although the ID cannot be guaranteed to be unique, the likelihood of a collision occurring during the transaction lifetime among the limited set of nodes that it sees is extremely low. Hops-to-live is set by the originator of a message and is decremented at each hop to prevent messages being forwarded indefinitely. To reduce the information that an attacker can obtain from the hops-to-live value, messages do not automatically terminate after the hops-to-live expires but are forwarded on with finite probability. Depth is incremented at each hop and is used by a replying node to set hops-to-live high enough to reach a requester. Requesters should initialize it to a small random value to obscure their location. As with hops-to-live, a depth of 1 is not automatically incremented but is passed unchanged with finite probability. [2]

5.3.1 Keys

There are two main varieties of keys in use on Freenet, the Content Hash Key (CHK) and the Signed Subspace Key (SSK).

A CHK is an SHA-1 hash of a document and a node can thus check that the document returned is correct by hashing it and checking the digest against the key. This key contains the meat of the data on Freenet. It carries all the binary data building blocks for the content to be delivered to the client for reassembly and decryption. The CHK is unique by nature and provides tamperproof content. A hostile node altering the data under a CHK will immediately be detected by the next node or the client. CHKs also reduce the redundancy of data since the same data will have the same CHK.

SSKs are based on public-key cryptography. Documents inserted under SSKs are signed by the inserter, and this signature can be verified by every node to ensure that the data is not tampered with. SSKs can be used to establish a verifiable pseudonymous identity on Freenet, and allow for documents to be updated securely by the person who inserted them. A subtype of the SSK is the Keyword Signed Key, or KSK, in which the key pair is generated in a standard way from a simple human-readable string. Inserting a document using a KSK allows the document to be retrieved and decrypted if and only if the requester

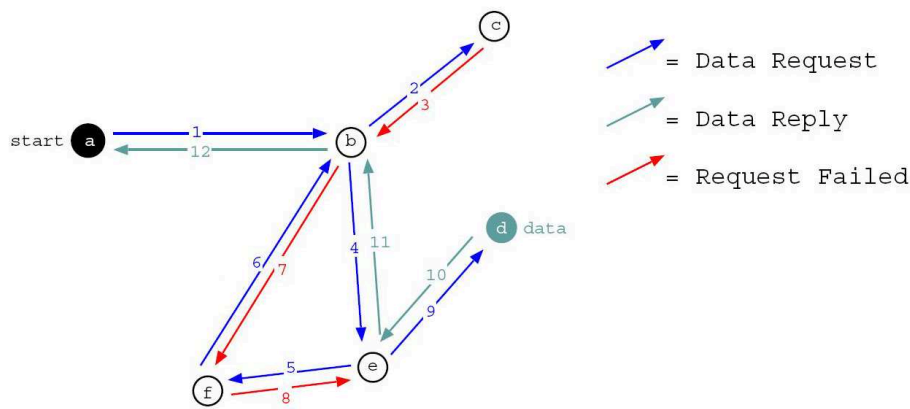


Figure 5.1: A typical request operation: note that request 6 fails because a node will refuse a data request which has already been seen.

knows the human-readable string; this allows for more convenient (but less secure) URIs for users to refer to.

All files are then encrypted using the descriptive string, which the user must publish in order for other clients to be able to decrypt the file. This is done for legal or political reasons alone. It may be preferable for a node not to know what it is storing as all it has is encrypted data and a hash key. This acts as a legal or political immunity.

5.3.2 Retrieving Data

In order to retrieve data, the user must first calculate it's hash. The user then sends a request to his own node specifying the hash and also a TTL for the request.

When a node receives a request, it first checks it's own datastore to see if it already has the data. If not, it looks up the closest key in it's routing table and sends the request to the corresponding node. If that request is ultimately successful and returns with the data, the node will pass the data back to the requester, cache the file in its own datastore and create a new entry in its routing table associating the actual data source with the requested key.

If a node cannot forward the request to its preferred node, it then forwards the request to the second nearest and so on. If a node runs out of candidates to try, it reports failure back.

The request thus operates as a steepest-ascent hill-climbing search with backtracking. If the hops-to-live is exceeded, a failure is propagated back. These mechanisms have a number of effects and should improve routing over time for two reasons. First, it will lead nodes to specialize in locating sets of similar keys. If a node is listed in a routing table under a particular key, it will tend to receive mostly requests for key similar to that key. It should therefore gain "ex-

perience” in answering those queries and become better informed in its routing tables about which other nodes carry those keys. Second, nodes should become similarly specialized in storing clusters of files having similar keys. Because forwarding a request will ultimately lead in the node gaining a copy of the requested file, and most requests will be for similar keys. Taken together, these two effects lead to a specialization of each node for the key field he will be most asked about.

Another effect is that successful nodes will be more inserted in routing tables, and will thus be contacted more often than other less active nodes.

Since the keys are derived from hashes, lexicographic closeness of the files doesn't imply any closeness of the hashes. So yet another effect is that the data should be well distributed among the nodes, lessening the possibility of a node storing all the data on a given subject or philosophy.

Finally, it should be noted that Freenet communicates in a chain-like sort of way. A node receives request only from it's neighbors and forwards such requests again to it's neighbors. It has no clue of the source or destination of the request. Not even the node immediately after the sender can tell whether its predecessor was the messages's originator or was merely forwarding the message of an other node. Similarly, the node immediately before the receiver can't tell whether its successor is the true recipient or will continue to forward it. This arrangement is intended to protect information consumers and producers. By protecting the latter it makes it difficult for an adversary to locate holders of a specific file in order to attack them.

5.3.3 Storing Data

Inserts follow the same strategy as requests. The user first calculates the hash of the file and contact its node with the hash as well as a TTL. The nodes will pass on the “insert request” just as before. If the hops-to-live limit is reached without a key collision being detected, an “all clear” result will be propagated back to the origin inserter. The user then sends the data, which is then propagated along the path established by the initial query. Each node along the path creates a new entry in its routing table with the inserter as data source (although some nodes might choose to select themselves or another node as source).

This mechanism has three effects. First, newly inserted files are placed on nodes already possessing similar keys. This reinforces the clustering of keys established in the request-mechanism. Second, inserts can be a mean to signify the existence of new nodes. Finally, attackers trying to supplant existing files by inserting junk files under existing keys are likely to simply spread the real files further, since the originals are propagated on collision.

5.3.4 Managing Data

Data is managed as an LRU (Least Recently Used), this means that eventually data can no longer be available on the network if all nodes decide to drop it.

In this Freenet differs from other systems (Eternity, Free Haven) which seek to provide lifetime guarantees for files.

5.3.5 Adding Nodes

It would be a security problem if a new node could choose his routing key by himself. This rules out the most straightforward approach.

Once a node connects to the network, it sends a request with a certain TTL and sends it to a neighboring node. Each node receiving such a request computes a random hash key and sends the request to a random node until the TTL is reached. All nodes then commit their hash key. All hashes are then XORed and the result is the new node's hash. This yields a random key which cannot be influenced by a malicious participant. Each node then adds an entry for the new node in its routing table under that key.

5.4 Facts

A P2P network is said to be scalable if the performance of the network does not deteriorate even for very large network sizes. The scalability of Freenet is being evaluated, but similar architectures have been shown to scale logarithmically. An analysis of Freenet files conducted in the year 2000 claims that the top 3 types of files contained in Freenet were text (37%), audio (21%), and images (14%). 59% of all the text files were drug-related, 71% of all audio files were rock music, and 89% of all images were pornographic. It is important to note the fundamental design of Freenet makes accurate analysis of its content difficult. This analysis was done several years ago from within the United States, and the network has been vastly changed and expanded since it was published.

5.5 Attacks

As we have just seen, Freenet seems resilient to many attacks. The files are distributed across the network with little or no control from the users, all data is encrypted and there exists a signing mechanism. Users cannot effectively monitor their neighbors which creates a very strong anonymous structure. Finally the file-request or file-insert queries are done in a very sensible manner, which prevents effective query-flooding DOS attacks against the network (such as in Gnutella).

Nevertheless, we will now analyze a few possible attacks.

5.5.1 DOS Attack I

The most straightforward attack is to fill the storage space with junk data. There are two ways to achieve this, either to request the junk data from another malicious node or directly inserting it in the network. During the attack, the network is heavily used to upload and transfer the data. There are good chances,

that other users will attempt to download parts of this useless data and by doing so, waste time and bandwidth. Finally, if this attack managed to fill all of the network's storage space, no-one would be able to share any new files. Most users are easily discouraged and would quickly leave the P2P application after such problems.

This junk data would generally not be able to replace the real data as the real data is spread on collision. Nevertheless, by inserting the junk data with low TTLs (i.e. minimizing the collision possibilities) or giving it to offline nodes, the attacker could manage to create more bogus copies than original copies. In which case the junk data could replace the original data if KSKs are used (this is not the case with CHKs or SSKs).

The first partial solution is to separate the recent data from older data. New data should not be able to fill up space reserved for older data. This partially solves the problem as junk data would rarely survive enough to be treated as old data. Yet the junk data still has the potential to eliminate all other new data as well as heavily slow the network down.

Keeping the same model in mind, we can choose to accept insertions only by nodes we trust. The problem here is we only have the IPs to authenticate other nodes. IPs can be spoofed and the attacker would thus insert data on behalf of another node. A more advanced cryptographic solution would require nodes to own public keys and sign any insertions with their key. There is no way to publicize a public key in Freenet yet.

A last possible solution is simply to not (or only sometimes) cache data when it is inserted. This would prevent the DOS attack by pure insertion even if it would also probably hurt performance. The attacker also still has the possibility to request the junk data from strategic places in order for it to be cached across the network, although this is more tedious than pure insertion.

5.5.2 Malice and Eavesdropping

If a node unluckily contacts a malicious user to enter the network, then the attacker can assign him the public key he wishes (using this same system, an attacker can choose any public keys he wishes as long as he is helped by other malicious nodes). An attacker can force a neighboring node to hold illegal material by requesting it through him. As long as the attacker doesn't let any other nodes through to the unlucky node (only malicious ones), then anything this node attempts can be monitored.

The attacker can use dictionary attacks in order to decrypt all messages encrypted with KSKs which pass through him (possible because in KSKs, the key is just a hash of a search string). By keeping track of the published keys (by using web crawlers for example), an attacker could even hope to decrypt more messages.

Finally, by monitoring the traffic, an attacker can gather information on the network by logging which keys are inserted by which nodes, which keys are being requested and what data is sent by whom. As failed requests or timeouts are sent in plain text, the attacker can also monitor these.

5.5.3 Anonymity

By looking at the TTL of the packets which pass through him, a node can gain knowledge on how far it is in the chain of a certain key. Although Freenet sometimes randomly increments the TTLs, this does not make them unusable. For example if an attacker sends a request with a TTL of 1 and receives a response, then he can be pretty certain that his neighbor is detaining the targeted file. He can also check how much time passes before the answer returns and use this information to make further deductions. Finally, as described previously, an attacker can monitor all information contained in the packets passing through him.

All this monitoring can give an attacker a good knowledge of the network surrounding him, yet this is not sufficient to completely break Freenet's anonymity. Any node can be connected to any other node, therefore as soon as a message is passed on to a neighbor, the attacker loses nearly all possible monitoring options. There is only one case where an attacker can be sure that a node is detaining a targeted file; if he has managed to surround the node with malicious entities and doesn't observe any outgoing messages for an ingoing request. Yet this scenario is highly unlikely: an attacker can never be sure he has completely surrounded a certain node unless the node connected to him in order to join the Freenet. In that case however, the attacker already knows most of the files the node is detaining which makes it uninteresting.

In the case that the attacker is more than just a malicious user, perhaps even a government, then he can use additional powers to monitor the traffic across the internet and thus might be able to follow a message to its destination.

We thus conclude that although an attacker can gather quite a lot of information of the network directly surrounding him and in some cases actually manage to break the anonymity, in general Freenet manages to preserve its nodes anonymity. Node anonymity is only broken in the case that a node initially connects to malicious node and doesn't use any alternative ways to connect to a non-malicious node.

5.5.4 More-then-just-routing Attacks

If the data is not encrypted using a CHK, then a malicious user can replace the data with junk data in every message which passes through him. Instead of forwarding file requests to the node with the most similar public key, he can do the contrary and redirect the requests to unadapted nodes. This would also prevent the data from clustering properly at nodes which have a similar public key. Another way to prevent the data from clustering properly is replace the data sources in the return messages with the least appropriate node. This will make other nodes contact this node later on with unadapted nodes. There is also nothing there to prevent the attacker from sending "timeouts" or "request failures" to any data requests he receives.

We have seen before, that successful nodes are contacted more often. In order for a malicious user to attack the routing protocol efficiently, he must

stay as well connected as possible. To achieve this, he can set himself as data source in every message that passes through him. Yet another possibility is to use web crawlers which gather any Freenet clients addresses and initiate handshakes with these. Finally, Freenet uses a PHP script located at: <http://www.octayne.com/inform.php> to publicize available Freenet nodes. As default, when anyone starts a Freenet client, it registers itself automatically to this PHP script. An easy way for an attacker to publicize his nodes is to write a small script which fills “inform.php” with a list of malicious nodes. What’s more, such a list filled with all the IPs of Freenet users constitutes a serious threat to the anonymity of the network.

Finally, if the attacker is interested in a specific file, he can use a dictionary attack to find keywords which generate a similar hash. By inserting these in the network (thus setting himself as data source for these files) he can steer future requests for the specific file he is interested in and do whatever he wishes (answering with junk data, monitoring ...)

5.5.5 Simulation

In order to have a precise idea of which attacks are most efficient against the Freenet structure, we decided to write a simulation of the Freenet network. The nodes’ behavior was programmed in order to be as close to a normal Freenet node as possible. They have limited storage space (40 files maximum), can only connect to a limited number of neighbors and can disconnect from the network during the simulation run. Each node receives at the beginning several files (15 in this case) selected randomly from a global library containing 10000 different files who’s keys were uniformly distributed. What’s more, there is no pre-network structure. Each node inserts himself in the network at a random node and then proceeds to query random files. The network is thus dynamically built during the first 5000 random queries of each run of the simulation, an initialization phase we do not consider for the results. The simulation is then tested for an additional 1000 rounds during which all results are monitored. Messages were given a TTL of 20 hops and could be corrupted by malicious nodes. The simulation was run several times with 2000 good nodes and 20 malicious nodes, the malicious nodes pretending to be good during the initialization phase.

Before discussing the results, we would like to underline the fact that simulations can never perfectly model reality how ever precisely they were implemented. We will therefore not use the results directly but more to get an idea of which attacks would be most effective. Even though, the simulation proved remarkably robust against variable changes (changing the TTLs, the storage space ...) which only mildly affected the final result.

Results

The first runs of the simulation were done with only good nodes. The simulation returned an average of 90% successes which is quite plausible. Indeed, it is possible that a node chooses to query data which is not available on the

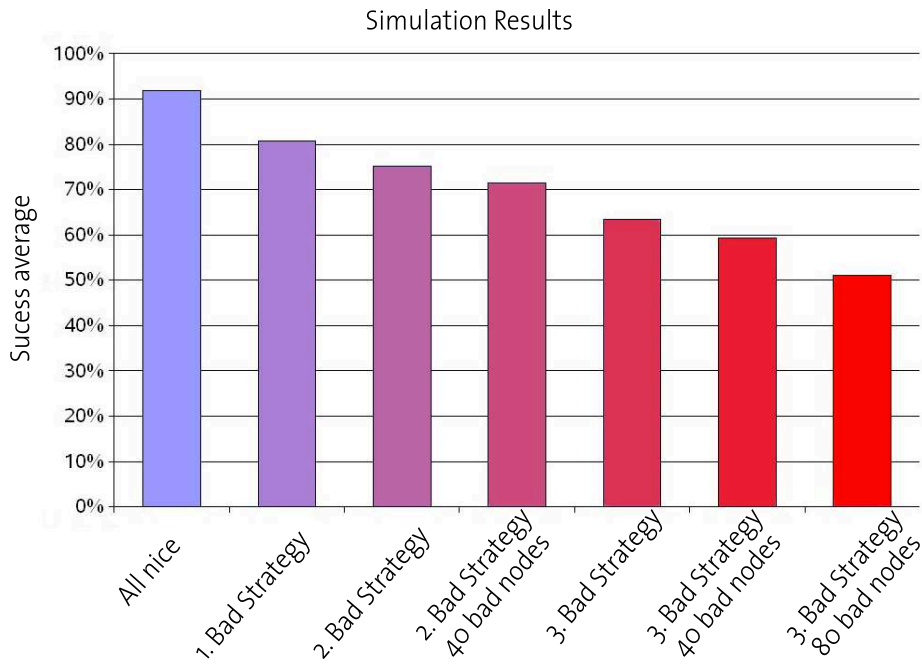


Figure 5.2: The results of the simulation run on the three different malicious strategies.

network. The other failures are due to the TTL which expired before the file was reached.

We then tried out the first attack: all bad nodes should forward each message to the worst possible node instead of the most qualified. This barely had an impact, the simulation showed an average of about 80% successes. This is understandable as forwarding the message to the least qualified node is equivalent to starting a query with a diminished TTL.

The second attack simulated was to make malicious nodes overwrite every data-source with the worst possible data-source. The results were comparable to the first malicious strategy, although a little more effective. The simulator indicated an average success rate of 73%. This success rate barely changed when we doubled, then tripled the number of malicious nodes.

Finally, we tried the last attack: all bad nodes should corrupt each message which passes through them. This time, the simulation showed only 63% success. We then decided to double the number of malicious nodes making them 40. With only 40 malicious nodes and 2000 good nodes, the simulation then showed an average success of only 56%. It came down to a 51% success average when we doubled the number of malicious nodes again.

Conclusion

The simulation clearly shows Freenet is capable to adapt to the first two attacks. Its model is flexible enough to defeat both attacks which aimed to destroy the nodes' specialization. Yet it is very vulnerable to the third kind of attack as 2% malicious nodes can already reduce the success rate to nearly 50%. It is understandable that this attack is most effective. Indeed, the malicious nodes operate as a team, passing each other as data-sources to ensure other nodes will keep connecting to them. This way they can stay connected to the network whilst corrupting any message passing through them. It is always impressive to observe how much an organised few can achieve against a large group of disorganised individuals. This is also the case in our society.

The only way to prevent this attack from being this effective is to develop a mechanism permitting to identify and avoid malicious nodes. Yet it is unclear at the moment how such a mechanism could be achieved in the Freenet paradigm.

5.5.6 DOS Attack II

Using the same idea as before, an attacker can DOS the owner of a specific file. Of course, if the file is already very distributed, then the attack won't have much impact. We consider the case where a few nodes contain a certain file we wish to remove from the network.

The attacker cannot drown these nodes under data request queries, as this would only distribute the file over the network. What he can do is slightly modify the hash and request the data corresponding to this new hash. As the new hash is very similar to the hash of the targeted file, the requests will also be routed to the nodes containing the specific file. The targeted node will probably send a "request failure" back as the data probably doesn't exist. If this is not sufficient, the attacker can use a dictionary attack in order to find the search string of files which hash very near to the targeted file. This has the advantage that the targeted file will not be distributed across the network on the return. An attacker thus has the means to target all nodes containing the specific file, despite Freenet's anonymity. An attacker can thus launch a DDOS attack against these nodes in order to bring them down. [18]

5.6 Future

It seems Freenet is betting on a "Darknet" model in the near future. A Darknet is a private virtual network where users only connect to people they trust. Typically such networks are small, often with fewer than 10 users each. The program is currently undergoing a massive re-write, which changes the fundamental way the routing takes place, deviating massively from Ian's original paper. The project is attempting to model a series of linked Darknets, where users only connect directly to other users they know and trust. This approach emulates the larger small world effect. This is an attempt to eliminate the scaling problems which have plagued the project, and to improve anonymity. We don't have the

data or the protocol details to be able to analyze it. The release is expected for early 2005.

Chapter 6

Final Conclusions

6.1 Concluding Weaknesses

During this work, one general rule has manifested itself all the time. Each time a P2P application has taken an easier path, facilitating the implementation, it has always been used as a weapon during an attack.

Implementing first generation P2P networks (Napster) using a centralized server is the easiest approach possible, as all the routing information is centralized. But this also presents the easiest attack as there is a single point of failure, which should not be the case in a distributed network.

Second generation P2P networks (such as Gnutella) went a little further in the distributed philosophy. They removed the centralized server, but created supernodes which create a kind of highway in order to facilitate the transfer of routing information. But these hierarchy can easily be used by a malicious user in order to spread a virus through the same network much faster.

Finally Freenet, a third generation P2P network, removed all kinds of hierarchy as well as introducing cryptography and other more or less efficient fancy gadgets we saw last chapter. Yet also they lacked the courage to fully implement a distributed network. This PHP script which keeps track of actual Freenet nodes is an obvious vulnerability.

P2P protocols are publicly known, which also means any malicious users can analyze them and model their attack accordingly. Anything other than pure P2P, any model relaxation can be used against the network to make the attack more effective.

One could argue that one could protect the impure functionalities with other security paradigms. Yet it is often the case that these newly introduced paradigms greatly complicate the protocol without really making the protocol secure. Either they only block part of the attack or, even worst, they permit a new kind of attack. We saw such an example in the case of Sybil attacks, where mapping the IP to each node enabled a new kind of attack: creating fake identities for valid nodes and then denouncing them as malicious. In any case, these enhancements

are far from trivial.

In order to make the network as robust as possible, it must be pure P2P. The use of randomization would permit an even more efficient defense against malicious users.

6.2 Our Solution

6.2.1 Observations

We will now give our version of the best possible approach to model a P2P network. In order to do this, we will use our previous experience as well as two further observations.

In the end, malicious nodes are programmed by individuals, so the answer to attacks cannot be purely technical. The flexibility and unpredictability of human interactions can only be countered by other human interactions. This idea is not new and some P2P applications try to include this interaction. But not all kinds of interactions are good: we argued previously that reputation-based networks weren't a good idea because of the introduced hierarchy. Yet the fact that a human interaction is necessary is our first observation.

The second observation is that duplicate behavior in different programs are considered a nuisance by users. If you like chatting, you probably prefer to use only one program rather than having to download ICQ, MSN and Skype in order to correspond with all your friends. This is even more true when using trust-based applications. Trust is a notion which transcends the functionality you wish to use, as long as it has the same critical level. You probably shouldn't trust the person who walks your dog with your lifesavings, but it is reasonable to trust him to feed your cat or water the plants. A user who has managed to obtain a certain degree of trust in a certain functionality, would probably like to keep his level of trust in other resembling functionalities. Indeed, it would be very cumbersome if one had to start over and obtain a separate degree of trust for each different application.

These two observations lead us to the following conclusion: we need some sort of human trust interaction and it would be best if the mechanism could be modularly used elsewhere. There is a paradigm which nearly embodies these two observations: "web of trust" [1].

6.2.2 PGP, Web of Trust and Darknets

"Web of Trust" is implemented by PGP (Pretty Good Privacy) [1]. PGP is an alternative way of authenticating a user. The normal way is to download the certificate of the user and then to check with the certificate authority if the user's certificate is valid. Note the very hierarchical construction of this model which is why it cannot be applied to P2P networks (as there cannot be any central authority).

Instead, PGP removes the central authority: each user must manually sign

which certificates he has authenticated. The interesting point is that he can also assign a certain level of trust to the entity. There are of course, multiple levels of trust for the model to be as accurate as possible. Once a user has decided to trust a certain entity, then all the entity's authenticated relations are also authenticated to a certain extent, depending on whether this entity is also authenticated the importance of the trust levels.

The only problem is that the meaning of "trust" is only very loosely used in this paradigm. PGP worries solely on authenticating: binding individuals to their public keys. The only trust used in this context is to decide to authenticate entities already authenticated by a trusted entity. In our case, we do not care about the physical identity of our relations, but would instead prefer to know if he can be really be trusted (enough to accept files from). This is completely orthogonal to the entity's physical identity. We would like to bind a public key to a certain level of trust.

This newly defined model is not at all the same as a reputation-based system. In reputation-based systems, a node checks the reputation of another node by asking all nodes what they think of this node. It thus takes a decision based on data given by other nodes it doesn't know it can trust in the first place. In our "web of trust", you would only consider nodes which you have manually decided to trust. No number of malicious nodes can alter your decision if you do not trust them which is not the case in reputation-based systems. This model permits users to build a network of more or less trusted entities: so-called Darknets.

6.2.3 P2GP

This newly defined "web of trust" paradigm fits perfectly on P2P networks. Our approach would thus consist of a pure P2P application mounted by an application similar to PGP, hence the name "P2GP". There should not be any big implementation problems as the PGP-like application would be modularly built. There would be numerous advantages of having such a structure.

- Before querying a certain node, the program would first use this PGP-like application to check what degree of trust the node has. According to the degree of trust, the user could then choose to pursue the download or stop. The user could also set a separate trust threshold for each download: downloading a picture of your favorite rock-star is maybe not as critical as downloading an update for your computer.
- The user could encrypt his query using the trusted node's public key, and the node could answer by using the user's public key. Such communication would be very secure and could not be broken by using a dictionary attack as it can be the case in Freenet.
- After connecting to any node (possibly a malicious one) the user could then look for his trusted nodes on the network in order to use them as a direct neighbor. To be sure of the authenticity of a trusted node, the user

could first give it a series of cryptographic challenges only the real owner of the trusted public key could be able to solve (i.e. returning signed messages). This would be impossible for malicious nodes to simulate and could prevent man-in-the-middle and certain eclipse attacks.

- Because nodes cluster in trusted networks, P2P viruses would probably need much more time to propagate through the network. This would also certainly prevent corrupted files or fake copies from spreading efficiently.

Yet this PGP-like application is not yet the answer to all problems, there are a number of unanswered points such as the anonymity issue of a node. Note that we have at this point not set any constraints to the underlying P2P application other than that it should be pure. Any kinds of technical solutions to enhance security or address the unanswered issues could be implemented there which gives our approach even more flexibility.

Performance

The only problem we foresee is a performance one. In the case of file sharing, the power of P2P networks resides in the number of nodes involved. Even if the number of nodes you trust grow exponentially to the number of directly trusted nodes, it is not possible for a user to scan enough nodes in order to reach the many thousand which constitute the network. This performance problem is partially unavoidable as it is price for introducing trust. The fact that each download could receive a different trust threshold should permit to regain part of the performance. It is very interesting to note that this solution also seems to be the direction Freenet is taking.

6.3 Conclusion

We have now finished our analysis of security in P2P networks. As a conclusion we can re-express the fact that only pure P2P stand a chance against attacks, any kind of shortcuts taken in the implementation can be turned around in order to attack the P2P application in a more dangerous manner. We finally observed that it would be interesting for a PGP-like application to exist. This application should not solely worry about authenticating users (binding public keys to physical identities) but also how much trust can be given to a public key. If such an application existed, it could be used by P2P applications as a very efficient protection against malicious attacks.

Bibliography

- [1] <http://www.wikipedia.com>
- [2] Ian Clarke, Brandon Wiley, Theodore W. Hong, Oskar Sandberg
Freenet: A Distributed Anonymous Information Storage and Retrieval System
- [3] Scott Jensen: *The P2P revolution*
peer-to-peer networking and the entertainment industry
<http://www.nonesuch.org/p2prevolution.pdf>
- [4] Lidong Zhou, Lintao Zhang, Frank McSherry, Nicole Immorlica, Manuel Costa, and Steve Chien: *A First Look at Peer-to-Peer Worms: Threats and Defenses*
- [5] Jason E. Bailes, Gary F. Templeton: *Managing P2P Security*
Considering the benefits and trade-offs of file-sharing systems
- [6] D. Dumitriu, E. Knightly, A. Kuzmanovic, I. Stoica and W. Zwaenepoel:
Denial-of-Service Resilience in Peer-to-Peer File Sharing Systems
- [7] J. Liang, R. Kumar, Y. Xi, and K. Ross : *Pollution in p2p file sharing systems*
In IEEE INFOCOM, 2005.
- [8] S. Chartrand: *New way to combat online piracy*
The New York Times, May 17, 2004.
- [9] N. Christin, A. Weigend, and J. Chuang:
Content availability, pollution and poisoning in peer-to-peer file sharing networks.
In ACM E-Commerce Conference, 2005.
- [10] John R. Douceur: *The Sybil Attack*
- [11] Atul Singh, Miguel Castro, Peter Druschel, Antony Rowstron:
Defending against Eclipse attacks on overlay networks
- [12] Arno Wagner, Bernhard Plattner, Thomas Dübendorfer, Roman Hiestand:
Experiences with Worm Propagation Simulations

- [13] Stuart Staniford, Vern Paxson, Nicholas Weaver:
How to Own the Internet in Your Spare Time
- [14] Thomas Dübendorfer, Arno Wagner:
Past and Future Internet Disasters: DDoS attacks
- [15] <http://openbsd.org/security.html>
- [16] Seth James Nielson, Scott A. Crosby, Dan S. Wallach:
A Taxonomy of Rational Attacks
- [17] Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, Pierangela Samarati, Fabio Violante:
A Reputation based Approach for Choosing Reliable Resources in Peer-to-Peer Networks
- [18] Rachna Dhamija: *A Security Analysis of Freenet*
- [19] Neil Daswani, Hector GarciaMolina: *QueryFlood DoS Attacks in Gnutella*
- [20] Fabian Kuhn, Stefan Schmid, Roger Wattenhofer:
A Self-Repairing Peer-to-Peer System Resilient to Dynamic Adversarial Churn
- [21] Jason E. Bailes, Gary F. Templeton: *Managing P2P Security*
- [22] Geetha Ramachandran, Delbert Hart: *A P2P Intrusion Detection System based on Mobile Agents*
- [23] Salman A. Baset, Henning Schulzrinne, September 15, 2004
An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol
- [24] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, Dan S. Wallach:
Secure routing for structured peer-to-peer overlay networks