Peer Pressure: Distributed Recovery from Attacks in Peer-to-Peer Systems

Pedram Keyani **Brian Larson** **Muthukumar Senthil**

Computer Science Department
Stanford University
Stanford, CA 94305

{pkeyani, balarson, msenthil}@stanford.edu

Keywords: Peer-to-peer systems, overlay networks, scale-free networks, fault recovery, malicious attack
Abstract: Peer-to-peer systems such as Gnutella are resilient to failures at a single point in the network because of their decentralized nature. However an attack resulting in the removal of a small percentage of highly connected nodes could cripple such systems. We believe that distributed attack recovery is not simply a reactive process but requires proactive measures by the nodes in the system. We propose a distributed recovery method where clients proactively detect attacks by monitoring the rate at which their first and second-degree neighbors leave the network, and reconfigure themselves to form a topology that is more resilient to attacks when one has been detected. This topology is created and maintained through a new type of node discovery mechanism that is used during normal network operations. The recovery method is able to reconnect the network and deal with any ongoing attacks once one has started.

1. Introduction

Gnutella is one of the most widely used peer-to-peer (P2P) protocols for file sharing on the Internet. It is a simple protocol for communication between peers (*servants* in Gnutella) to form an overlay network on top of the Internet topology. The Gnutella protocol does not specify the behavior of individual clients; it only prescribes a format for them to communicate [7, 9].

The current topology of Gnutella adheres to the power law where most nodes have very few connections while a small portion of the nodes have many connections and hold the entire network together. It is this property that makes Gnutella particularly susceptible to malicious attacks resulting in the removal of highly connected nodes [12]. By merely removing a small portion of these nodes it is possible to fragment the entire network into many isolated pieces. In order to combat this weakness each node must be capable of detecting such an attack and being able to respond quickly.

Our proposed recovery method has two tasks that each node in the systems must perform. First, each node needs to discover backup connections during normal network operations and maintain a list of them in case of an attack. Our proposed technique for this requires the addition of a *random discovery ping (RDP)* to the Gnutella protocol. Second, nodes must be able to detect attacks quickly and react to them. Our method for detecting attacks requires nodes to keep track of the rate at which their first-degree (direct neighbors) and second-degree neighbors (neighbors of neighbors) die. Once an attack is detected failed neighbors are replaced with those discovered using the random *node discovery ping* during normal operations. In section 2 we describe the background to the problem more fully. In section 3 we describe the model of Gnutella that we used in arriving at our solution. We describe our proposed solution in Section 4. In Section 5 we describe our simulator and the experiments that we ran. Finally, in Section 6 we conclude and talk about future work.

2. Problem Background

There are currently a wide variety of decentralized P2P schemes that are highly resilient to failures of random components [15]. Gnutella is a well-known P2P file sharing system that has this resilience property, but its overlay network is vulnerable to malicious attacks. Several programs exist that "crawl" the Gnutella network and report its topology, allowing a person to find the most highly connected nodes. A party that wished to attack a scale-free network then targets these nodes to bring down [2]. In order to perform an attack of this magnitude, an attacker would likely attempt a distributed denial of service attack

in order to remove these nodes [13]. It has been shown that if 4% of the most highly connected nodes are removed from Gnutella, the network will severely fragment, rendering it useless [12]. Gnutella's robustness to random failure and vulnerability to malicious attack is not unique. Indeed, the Internet has similar characteristics; an attack on 5% of hosts would result in the total collapse of the Internet [14].

Many real-world networks fall into a class of inhomogeneous networks called scale-free networks where a few nodes have many connections, but most nodes have only a few connections. Scale-free networks abide by the power-law relationship $P(k) \sim k^{-a}$, where $P(k)$ gives the probability that a node is connected to *k* other nodes [2]. It has been shown that networks such as the Internet, the WWW, and Gnutella tend to self-organize into scale-free topologies that abide by the power law [1, 3, 8, 10, 12].

There are two mechanisms that cause the formation of scale-free topologies. First, networks expand continuously by the addition of new vertices, and second, new vertices attach preferentially to vertices that are already well connected [3]. In Gnutella, the first mechanism can be seen by the fact that new nodes are continuously entering and leaving the system, meaning the topology is undergoing constant change and growth. The second mechanism can be seen by the fact that there are only a few hosts that clients initially connect to, due to the way bootstrapping is handled (further discussed in section 3.1). Furthermore, clients preferentially attach to stable, high bandwidth nodes. Hence, Gnutella is scale-free because of its adherence to these two mechanisms.

Failures in P2P systems are viewed as nodes suddenly and unexpectedly dropping out of the network. Scale-free networks such as Gnutella are very robust to failures of random nodes. This robustness is rooted in the inhomogeneous nature of the network. Nodes with few connections will be selected with much greater probability than nodes with many connections [2]. Unfortunately, the same inhomogeneous nature of scale-free networks that makes them robust to random failures makes them vulnerable to an attack resulting in the removal of the most highly connected nodes. As mentioned before, it is the case that removing a small percentage of these highly connected nodes from the Gnutella topology can completely fragment the network [12].

One network topology that is more resilient to attacks is an exponential network [2]. An exponential network is a homogenous network where each node has roughly the same number of links k. Instead of following the power law, $P(k)$ peaks at k and decays exponentially, following the Poisson distribution [2]. Exponential networks are built by connecting random neighbors together with no preference of one node over another. In this configuration it is improbable that any one node is holding the network together. The homogeneous nature of exponential networks makes them much less vulnerable to attacks then scale-free networks.

We would like to be able to reconfigure the topology of the network from scale-free to exponential in order to survive an attack, but this requires measures by the nodes themselves to detect that an attack is in progress. Also, reconfiguration of the topology to exponential requires nodes to discover other nodes randomly, but this must be done before the attack, as the network will become fragmented. It is therefore our position that to combat an attack nodes must proactively detect attacks and maintain backup connections selected in a non-preferential way. We describe our recovery method that uses this idea in Section 4.

3. Our Model of Gnutella

Before we propose a solution to making the protocol more robust, we feel that it is necessary to describe some assumptions we make about Gnutella and the corresponding models that we built. As it stands, Gnutella is a very free protocol in the sense that it only captures the messages that should be sent between nodes and leaves the remainder up to the client. Some of the open properties of Gnutella include bootstrapping, the number of neighbors maintained be each node, the up-time distributions of nodes, and the initial topology. We feel that the most reasonable way to model the first two properties (bootstrapping and number of neighbors maintained by each node) is to adopt the method used by the most common Gnutella client, Bearshare [4]. For the up and down time distributions we use a power law relationship fitted to measurements by 12, and for the initial topology we use data collected by Steven Gribble at the University of Washington.

3.1 Bootstrapping

There are many ways that Gnutella clients handle bootstrapping. This used to be done by connecting to a well-known point, Gnutellahosts.com, which maintained a list of servants with high availabilities that were likely to accept incoming connections [12]. On a connection from a newcomer, gnutellahosts.com returned one of these available servers. Bearshare works similarly in the sense that clients connect to a well-known Bearshare server (public.bearshare.net) and receive a list of servants from it [4]. Clients then directly connect to these servants. This preferential connection mechanism is responsible for the construction of a scale-free topology.

3.2 Number of Connections

The number of connections that a node maintains is not specified by the Gnutella protocol, so it is entirely left up to the specific client. Also finding out which client a node is running is not a trivial problem. We believe that a reasonable thing to do here is adopt the default value used by one of the most popular clients, BearShare. This client is set with a default max of 10 and min of 3 neighbors to maintain. We use these max and min values in modeling our nodes in our experiments.

3.3 Up and Down Time Distributions

As a model for the up and down time distributions, we use a power law function. A study of Gnutella shows that there are somewhere between 10,000 and 30,000 nodes that are alive at any given time slice [6]. The data that Steven Gribble's group has given us shows that there are 3,000 nodes that are alive during the course of a 12-hour period. We model the uptime distribution using the power law $P(t) = t^{-a}$, where $P(t)$ is the probability that a node is up for time t. We feel that a power law is an accurate model to use, because the majority of Gnutella nodes are up momentarily while fewer and fewer are up progressively longer. We conservatively assume that the shortest period of time a node can be up is 1 minute, since this represents the time required to sign on and off from the system.

3.4 Initial Topology

All of our measurements are based on a Gnutella topology we have received from Steven Gribble's group at the University of Washington that contains nodes that have been up for a 12-hour period. This data contains roughly 3,000 active nodes and the edges between them. We built an initial topology by adding 17,000 nodes to this base topology using the bootstrapping and uptime models discussed above, resulting in a network with roughly 20,000 nodes alive at any given time. In Figure 1, we show the number of nodes vs. the number of connections, plotted on a log-log scale. These results are very similar to the results of the Clip2 study.

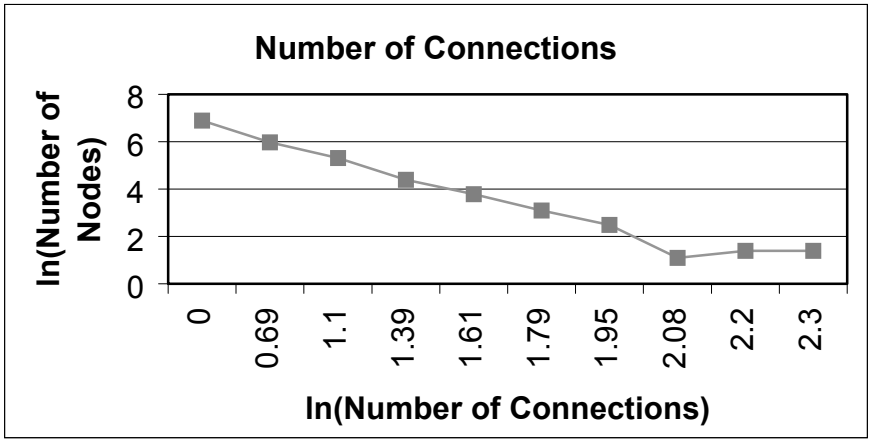

Figure 1. Distribution of connections in base Gnutella topology

4. Recovery Method

We would like to have a recovery method that prevents the network from becoming fragmented by an attack and prevents the decrease in querying effectiveness during the attack. By querying effectiveness we mean the number of neighbors that can be reached with a query. Most Gnutella clients maintain a "cache" of host names that they have encountered traffic from during operations. This data could be useful in connecting fragmented components during an attack, but clients do not make any attempt to detect an attack. Furthermore, if clients did detect an attack in progress and were to connect to hosts in the cache as a response, hosts that most recently sent queries through the network would appear in thousands of caches and could be overwhelmed by connection requests, or worse become targets of the attack themselves. It is therefore our position that to combat an attack nodes must proactively detect attacks and maintain backup connections selected in a non-preferential way.

Exponential networks fair much better than scale free networks under attack [2], so our solution is to maintain a virtual exponential overlay network in addition to the active scale free overlay network used by the system. We refer to the network as virtual because connections between nodes on the network will not be made, and no query traffic will be sent over the network. Each node will maintain a list of virtual neighbors in addition to the node's active neighbors, and they will be selected using a process for random node discovery that we have developed. Because there are no preferential connections, these random connections will constitute an exponential network.

The virtual exponential network will be used in place of the active scale-free network in light of attacks, which each node in the network is responsible for detecting. When a node detects an attack, the node will begin replacing its dead neighbors with nodes from its virtual neighbor list, thus making the exponential network active as the scale free network disintegrates as a result of the attack. By combining these two types of network topologies, we will have the robustness of scale-free networks to failure coupled with the robustness of exponential networks to attack.

4.1 Random Node Discovery

Our recovery mechanism requires that nodes be able to discover random nodes in the system with little or no preference in order to construct an exponential network. An optimal solution would allow for any node in the network to be chosen with equal probability, a task that could be accomplished by a centralized name authority. In a decentralized system such as Gnutella, maintaining a database of all active nodes in the system would be difficult if not impossible.

Our solution is for nodes to forward a message randomly through the network for a certain number of hops. This message, which we will call a *random discovery ping*, will be similar to a Gnutella ping. An RDP contains an origin node and a hop count, and is forwarded from node to node through the network. The final node that receives the RDP, determined when the hop count is decremented to one, will respond with a pong and is thus discovered. This node will then be added to the virtual neighbor list maintained by the origin node.

While standard Gnutella pings are forwarded by a node to all of its neighbors, an RDP is only forwarded to one of the node's neighbors. This will allow us to use a much larger time to live (TTL), thus getting deeper into the network without overwhelming the network with ping traffic. In our implementation, random discovery pings are created with a TTL of 20. This number is a good approximation of the diameter of the Gnutella network [6], so this will enable a ping to reach any node in the network. Acknowledgment messages are used to ensure that an RDP is not lost.

Initially a node forwards an RDP to a neighbor N selected randomly with probability scaling linearly with the number of neighbors that N has. This means that a node with 6 neighbors is 3 times as likely to have the ping forwarded to it than a node with 2 neighbors. This strategy is used for the first 10 hops in order to get the ping as far away from the originating node as possible. If this strategy is not used, in most cases the ping is only forwarded within the immediate vicinity of the originating node, passing through the originating node's most connected neighbor many times.

For the next 10 hops, the opposite strategy is used. The ping is forwarded to nodes with lower number of neighbors, meaning that a node with 6 neighbors is 1/3 times as likely to have the ping forwarded to it than a node with 2 neighbors. If this strategy is not used, then the nodes with the most neighbors will be selected with a frequency scaling with their connectivity. This is not what we desire because it replicates the preferential attachment property of the scale-free active network.

4.2 Maintenance of Virtual Exponential Network

The process by which each node discovers and maintains a certain number of virtual neighbors globally maintains an exponential network. Nodes are discovered by creating and forwarding an RDP as outlined by section 3.1. Although this is not specified in the Gnutella protocol, all clients have a range of neighbors <X, Y> and must maintain at least X neighbors. As mentioned in section 3, Bearshare uses the range <3,10>. Each node in the system is responsible for maintaining X virtual neighbors.

Similar to active neighbors, virtual neighbors are maintained through periodic pings, or heartbeats, which must be responded with a pong by the neighbor if it is alive [7, 9]. This allows nodes to detect the liveliness of their virtual neighbors. If a node's virtual neighbor is found to be dead, or a node has less than the minimum number of virtual neighbors, then the node discovers a new random node.

4.3 Attack Detection.

One of our goals in creating an attack detection mechanism was to minimize the amount of coordination between nodes. Our attack detection scheme leverages the information that a node has of its immediate neighbors (1^{st} degree) and their neighbors (2^{nd} degree), and is based on measuring changes in this local topology. From the viewpoint of a node in the network, an attack will most likely result in the removal of its most highly connected neighbors, which contain most of the node's 2^{nd} degree neighbors. This results in a node losing a greater percentage of its 2^{nd} degree neighbors than the percentage of 1^{st} degree neighbors. This is different from random failures, where the percentage of 1^{st} and 2^{nd} degree neighbors lost will be roughly equal.

Our attack detection mechanism requires nodes to maintain the number of 1^{st} and 2^{nd} degree neighbors lost during a specified time period *T*. These losses will be used to calculate the percentage of 1^{st} and 2^{nd} degree neighbors lost during this period. If the percentage of 2^{nd} degree neighbors lost is greater than or equal to the percentage of 1^{st} degree neighbors lost and a threshold *P*, an attack is detected. The threshold *P* is used to filter out false positives where some of a node's neighbors fail at random, but the total percentage of 2^{nd} degree neighbors lost is very small. In our implementation we used a value of 30 seconds for *T*, and a value of 50% for *P*.

We have performed some initial experiments varying *T* and *P* in order to find optimal values for them, but have not seen much variation in behavior surprisingly. We have explored values for *P* ranging from 90% to 10%. Attack detection rates increase as *P* decreases, but the detection rate increase is very slight. Varying *P* anywhere from 80% to 20% makes only about a 5% difference in detection rate, while dropping *P* below 20% results in increasing amounts of false positives. We have also explored values for *T* ranging between 10 seconds and 2 minutes. Increasing *T* from 10 seconds to 1 minute tends to increase attack detection rate a small amount, but increasing *T* above 1 minute makes nodes increasingly "paranoid" and increases the amounts of false positives. Our initial results show that our values for *T* and *P* may not be optimal, but they are within 5% of the best attack detection rates we have been able to generate without seeing false positives.

4.4 Reacting to Attacks

The virtual exponential network will be used in place of the active scale free network during an attack. Upon detecting a malicious attack, a node will begin replacing its dead neighbors with nodes from its virtual neighbor list, thus making the exponential network active as the scale free network disintegrates due to the attack. Also, after detection, the node counts the number of neighbors that have failed recently and replaces each of these with a virtual neighbor. The node creates an active connection with the virtual neighbor and removes it from the list of virtual neighbors. During the attack, the node replaces each

neighbor that dies with another virtual neighbor, until the node determines that the attack is over or it runs out of virtual neighbors. Also, it is important that the node not seek out new virtual neighbors to replace its lost virtual neighbors during an attack, as the extra traffic would increase the stress on the already failing network. Once a node no longer detects an attack, it returns to normal operations using its current list of active neighbors, including those added during the attack. As a part of normal operations it rebuilds a new list of virtual neighbors.

5. Results

The results we use to measure the effectiveness of our recovery method were generated using a general P2P network simulator that we have created. The simulator provides a framework for maintaining peers and delivering messages between them. It includes support for peers joining the network, peers leaving the network, and removing peers suddenly from the network. This last feature allows us to simulate an attack by removing the most highly connected peers. To simulate Gnutella, we created two Gnutella clients compatible with the simulator: one based on Bearshare and one with the additional recovery method built into it. We ran our experiments using the models and topology information described in section 3 to generate the results presented in the following sections.

5.1 Experiments

We ran two versions of our experiment: a control case with no recovery method, and a test case with the recovery method enabled. Each case was run 10 times for statistical soundness and to allow us to average the results and plot them with a 95% confidence interval. In our experiment, we simulated an attack on the network by removing 5% of the mostly highly connected nodes, a threshold greater than the 4% which [12] used to show the attack vulnerability of the network. We removed these nodes over a period of 5 minutes, a period measured as a common length for denial of service attacks by [11]. Experiments were run for 10 minutes, with the attack starting at 2 minutes.

5.2 Metrics

We chose metrics that would represent the fragmentation of the system and end-to-end performance of the system from the point of view of a user. To measure fragmentation in the system we calculated the percentage of living nodes in the largest connected component. A connected component is a set of nodes where a path exists between all pairs of nodes [5]. This measures whether fragmentation is breaking off small or large clusters of nodes. Another measure of fragmentation is the number of connected components. This measures the difficulty of rebuilding the network after the attack has done its damage. To measure end user performance we calculated the average number of nodes reachable within 6 hops of any node as a percentage of the total number of living nodes. We chose 6 hops because it is roughly the maximum hop count used when performing a search [6]. Because Gnutella is mainly used for searching and downloading files, this metric gives us an end-to-end measurement of the average number of nodes a user will be able reach with a query.

5.3 Connected Component Results

Figure 2 (below) shows the percentage of living nodes that are in the largest connected component. These results show that without our recovery method this percentage drops significantly from nearly 100% to below 30% in less than 30 seconds after the attack begins. With the recovery method in place this metric stays almost constant, initially dipping to just above 98% before quickly returning to the 99.9% range. This initial dip is due to the delay between the beginning of the attack and the detection of the attack by individual nodes. Once nodes start detecting the attack they are able to keep the network from fragmenting and maintain a very high percentage of nodes in the most highly connected component.

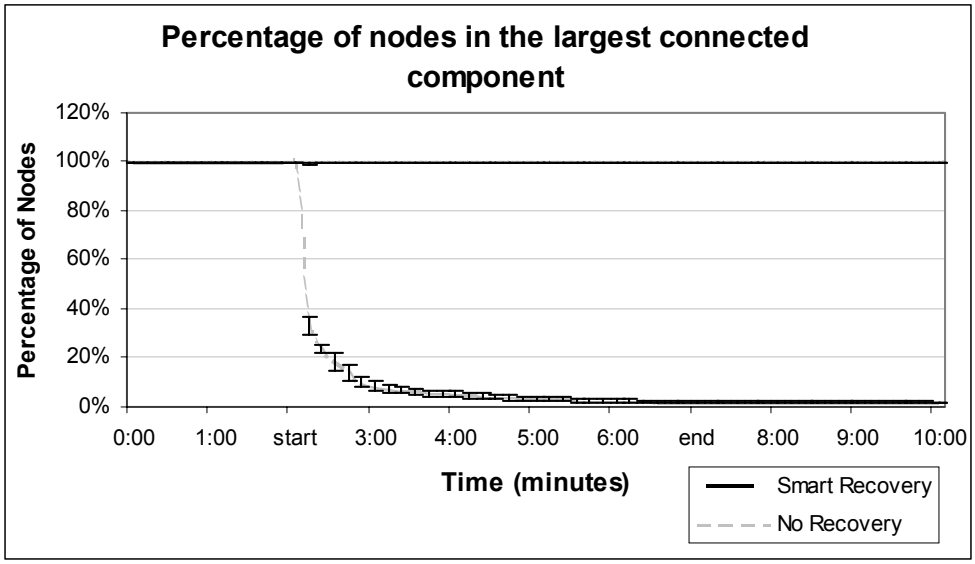

Figure 2. Data averaged out over 10 runs with a 95 % confidence interval

Figure 3 (below) shows the number of connected components in the system over time. The steady state value measured is close to 5 before the attack, counting the largest connected component and a few nodes that have just come online but have not bootstrapped onto the network yet. These results show that without our recovery method, this number increases very rapidly as soon as the attack begins, yielding hundreds of network fragments. With our recovery method, there is an increase in the number of connected components during the attack, but it returns to the steady afterwards.

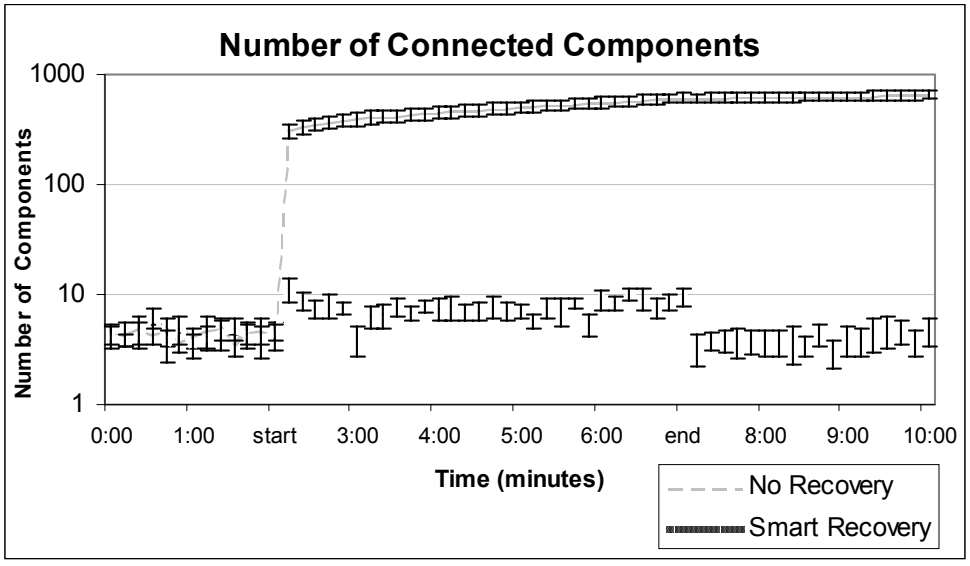

Figure 3. Data averaged out over 10 runs with a 95 % confidence interval

5.4 Average Percentage of Neighbors Reachable in 6 Hops

Figure 4 shows the average percentage of nodes reachable within 6 hops in the network. Again, the results show that without our recovery method this percentage drops off significantly from roughly 25% before the attack to under 1% in just 30 seconds. With our recovery method, this drops below 4% initially, but begins to recover immediately. The initial drop is still severe as a result of the most

connected parts of the network being removed, but the recovery method is able to increase this to 12% over the length of the attack. Our method shows a return to roughly half of the pre-attack performance of the system in terms of query effectiveness.

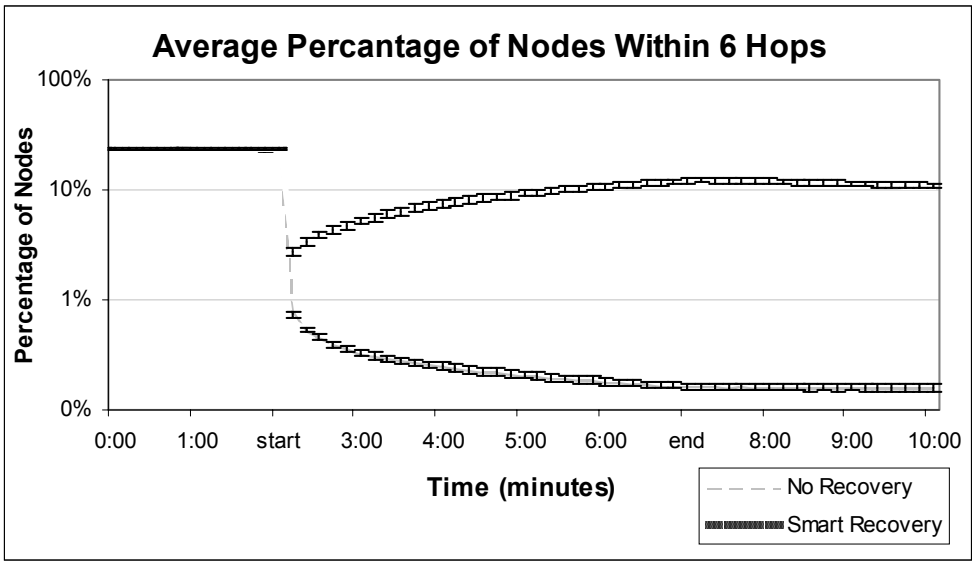

Figure 4. Data averaged out over 10 runs with a 95 % confidence interval

5.5 Random Node Discovery

Figure 5 represents the effectiveness of random discovery pings in discovering nodes in a non-preferential way. The number of times a node is selected, or frequency, is plotted against the percentage of nodes having this frequency. The binomial distribution, showing the results for node selection by random independent trials, is plotted against the distribution measured using RDPs. These results show that our node discovery technique succeeds in selecting nodes in a non-preferential way in order to form an exponential network.

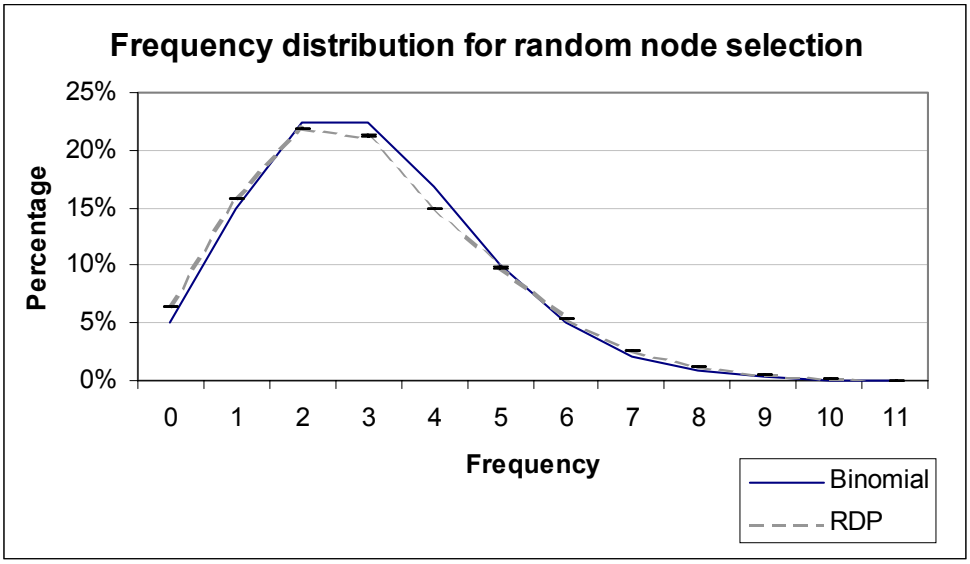

Figure 5. Data averaged out over 10 runs with a 95 % confidence interval

5.6 Attack Detection Mechanism

Figure 6 shows the effectiveness of our attack detection mechanism at differentiating random failures from attacks. In this experiment 5% of the nodes were removed over a period of 5 minutes as discussed in our experimental setup, but in this case the way in which nodes were selected was varied. The gray line shows the failure detection rates if the most highly connected nodes are removed, while the black line shows the rates if random nodes are removed. Less than 1 percent of the nodes detect an attack when removing random nodes, compared to as many as 14% when removing highly connected nodes. Also it is noteworthy to mention that no attack is detected outside of the attack interval, despite nodes entering and leaving the system constantly.

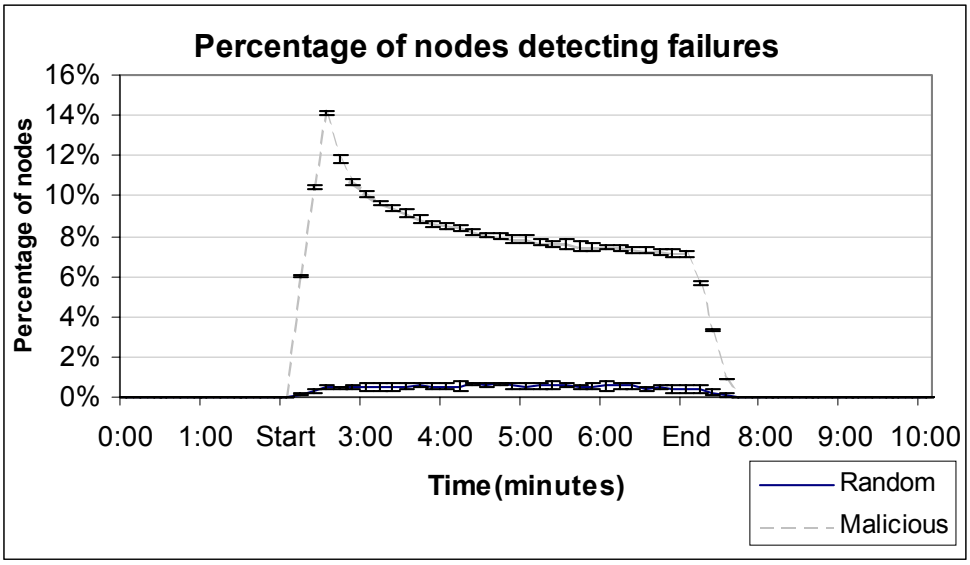

Figure 6. Data averaged out over 10 runs with a 95 % confidence interval

5.7 Messages per Node

Figure 7 represents the average number of messages (pings and pongs) generated per node per second during the experiment. Only messages related to node discovery and connection maintenance are counted, leaving out all query traffic. This graph shows that our recovery mechanism requires only a 20 percent increase in the amount of traffic used to maintain the system during the steady state before the attack. When compared to query traffic, this increase is less significant. The recovery method increases traffic on the network once the attack begins because of nodes connecting to their exponential backups and seeking new backups with RDPs. Once the attack is over, traffic levels return to the steady state. As for the standard client, traffic slowly decreases over the length of the attack as less connections must be maintained.

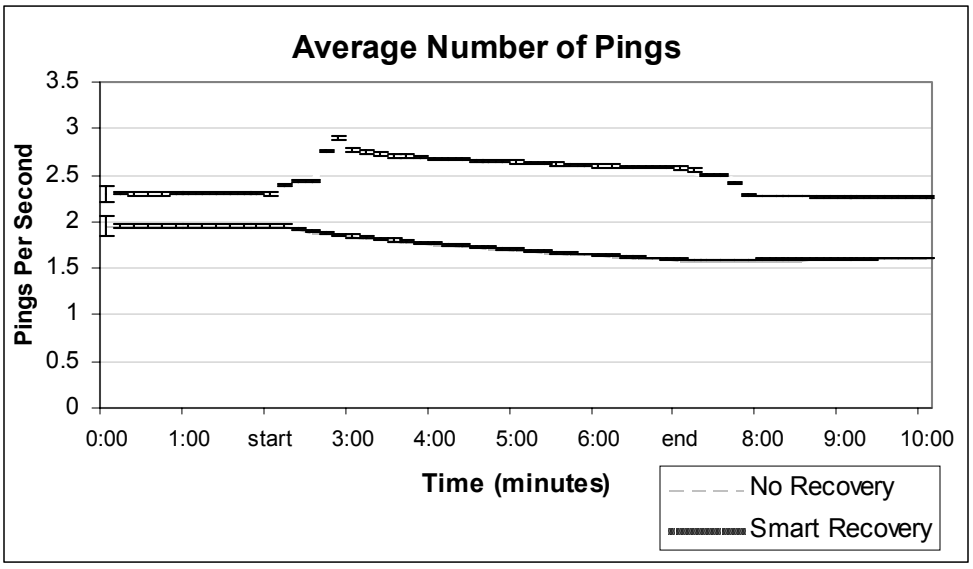

Figure 7. Data averaged out over 10 runs with a 95 % confidence interval

6. Conclusions

Experimental results show that our recovery method vastly improves the robustness of the Gnutella overlay network to attacks. Fragmentation is all but eliminated, and there is improvement in the effectiveness in querying during an attack. Also, the overhead of the method is small. Unfortunately, our recovery method would need to be adopted by a large percentage of users for it to work effectively. Only clients with our method in place would be able to forward the random discovery ping, meaning that a small percentage of clients running the recover method would not be enough to make any improvement. Without random node discovery, the method cannot work.

Our recovery method has an advantage that there is very little incentive to lie, given that "peers tend to deliberately misreport information if there is an incentive to do so" [12]. The only additional information that peers need to report is the number of connections they have. Reporting this as too high would lessen the likelihood of a peer being connected to a backup neighbor during an attack, while reporting this as too low would give the peer too many backups during an attack and potentially make it a target. Also, this information is something that peers can easily measure if an incentive to lie existed.

7. Further Work

We believe there are certain aspects of our recovery method that can be studied in greater detail. For example, we would like to investigate other techniques for random node discovery. Another area that we would like to look into is bringing the network back to a scale-free topology after an attack is over. Our initial premise was that exponential networks are better at surviving attacks, but a scale-free topology is ideal for handling normal operation including query forwarding and surviving random failures. Finally we would like to finish our experiments varying the parameters of the attack detection mechanism in order to find their optimal values.

A potential limitation of our recovery method is that it fails to take into account the "suitability of a given peer for a specific task" [12]. In our method this relates to the suitability of a peer as a randomly selected exponential backup for the task of holding the network together during an attack. Much of the recovery work is done by the least highly connected nodes, which may be dialup users who are unable to handle the added stress during attacks, or users who are unwilling to handle the added stress. This limitation is lessened given the high redundancy of the exponential backups, but is still a potential problem that we did not model.

Acknowledgments

We would like to thank Stefan Saroiu and Steven Gribble for providing us with topological data they collected on Gnutella. Armando Fox and George Candea are our advisors on this project and without their advice our work would have been much harder. And finally we would like to thank Jim Gray and Brendan Murphy for their advice on this project.

References

[1] W. Aiello, F. Chung, and L Lu. "A Random Graph Model for Massive Graphs." Symposium of Theory of Computing, 2000.

[2] R. Albert, H. Jeong, and A. Barabási, "Error and attack tolerance in complex networks," Nature 406

[3] A. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, 286

[4] http://www.bearshare.com/

[5] F. Buckley and F. Harary, *Distance in Graphs*. Addison-Wesley, New York, 1990.

[6] www.clip2.com, Gnutella Measurement Project

[7] http://www.clip2.com/GnutellaProtocol04.pdf

[8] C. Faloutsos, M. Faloutsos and P. Faloutsos, "On power-law relationships of the internet topology," Proc. of ACM SIGCOMM, Aug. 1999.

[9] http://www.gnutelliums.com/linux_unix/gnut/doc/gnutella-prot.html

[10] A. Medina, I. Matta and J. Byers, "On the Origin of Power Laws in Internet Topologies," ACM Computer Communication Review, vol. 30, no. 2 , Apr. 2000

[11] D. Moore, G. Voelker, and S. Savage, "Inferring Internet Denial-of-Service Activity," in Proceedings of the 2001 USENIX Security Symposium.

[12] S. Saroiu, P. Krishna Gummadi, and S. Gribble, "A measurement study of peer-to-peer file sharing systems," Technical Report UW-CSE-01-06-02, University of Washington, June 2001.

[13] http://www.securitystats.com

[14] http://www.trnmag.com/Stories/052301/Five_percent_of_nodes_keep_Net_together_052301.html

[15] B. Yang and H. Garcia-Molina, "Comparing hybrid peer-to-peer systems," In Proceedings of the 27th International Conference on Very Large Databases, September 2001.