12-2010

Dynamic probe positioning within peer-to-peer networks for mining contraband file exchanges.

Derek Reese
University of Louisville

Recommended Citation
Reese, Derek, "Dynamic probe positioning within peer-to-peer networks for mining contraband file exchanges." (2010). *Electronic Theses and Dissertations.* Paper 1194.
https://doi.org/10.18297/etd/1194

DYNAMIC PROBE POSITIONING WITHIN PEER-TO-PEER NETWORKS FOR MINING CONTRABAND FILE EXCHANGES

By

Derek Reese

B.S., University of Louisville, 2001

A Thesis

Submitted to the Faculty of the

University of Louisville

J.B. Speed School of Engineering

as Partial Fulfillment of the Requirements

for the Professional Degree

MASTER OF ENGINEERING

Department of Computer Engineering and Computer Science

December 2010

DYNAMIC PROBE POSITIONING WITHIN PEER-TO-PEER NETWORKS FOR MINING CONTRABAND FILE EXCHANGES

Submitted by:________________________________

Derek Reese

A Thesis Approved On

____

(Date)

by the Following Reading and Examination Committee:

____

Dr. Olfa Nasraoui, Thesis Director

____

Dr. Adel Elmaghraby

____

Dr. Michael Losavio

____

Dr. Ayman El-Baz

ACKNOWLEDGEMENTS

It has been my privilege to work with Dr. Nasraoui, Dr. Losavio and Nick Miles on this research grant. Dr. Nasrauoi and Dr. Losavio encouragement and guidance kept me focused on completing my portion of this work. I would also like to thank my friends and family for their encouragement during my studies. Finally I must thank my employer and most especially my manager, Jeff King, for their support and flexibility in the past 2 years while I completed this thesis.

ABSTRACT

Peer-to-peer networks have been growing in popularity over the past decade. There have been many new innovations that greatly improve access to a wide variety of content. This expanded capability combined with a strong sense of anonymity has given rise to increased proliferation of illicit content. In particular the increase in child pornography has been a growing concern in the United States and other countries. Thus law enforcement is motivated to find improved means for finding those sharing this material online.

Due to the dynamic and expansive nature of peer-to-peer networks, there is a need to develop methods that allow law enforcement to monitor with a high degree of confidence that a large percent of perpetrators can be identified. Thus a study of the current state of peer-to-peer networks with an analysis of how best to identify clients sharing contraband files on the network is needed to monitor these criminal elements.

Contents

Nomenclature

direct peer: a node that is connected by a single hop to another node

node: any client in the peer-to-peer network

peer: a node in the peer-to-peer network

probe: a node in the network that is controlled by a known operator that is exploring the network

TTL: TTL is the time-to-live value. In Gnutella this is used to indicate how many hops a message can travel before it should be removed from the network

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1. Motivation

Peer-to-peer networks have been growing in popularity over the past decade. There have been many new innovations that greatly improve access to a wide variety of content. This expanded capability combined with a strong sense of anonymity has given rise to increased proliferation of illicit content. In particular the increase in child pornography has been a growing concern in the United States and other countries. Thus law enforcement is motivated to find improved means for finding those sharing this material online.

Due to the dynamic and expansive nature of peer-to-peer networks, there is a need to develop methods that allow law enforcement to monitor with a high degree of confidence that a large percent of perpetrators can be identified. Thus a study of the current state of peer-to-peer networks with an analysis of how best to identify clients sharing contraband files on the network is needed to monitor these criminal elements.

1.2. Objectives

This thesis will propose the implementation of a specialized client for a Gnutella peer-to-peer network intended to monitor the network for illegal activity. The client should be able to start from any random point in the network and then, using the native features of the Gnutella protocol, dynamically position itself to monitor more illicit activity.

Adapting an existing node ranking technique for use in the dynamic node positioning method may allow clients to get improved query results. These query results will in turn increase identification of users sharing illegal content on the network. If formulated correctly, these probes could continually collect data, while changing their positions as new content is confirmed to be contraband or as nodes enter or leave the network. This thesis will consider both the Gnutella v0.4 and v0.6 versions of the network.

1.3. Contributions

The proposed techniques described herein will offer a new approach for a more aggressive and dynamic neighbor selection strategy in peer-to-peer networks that can improve query results for diffuse low-prevalence content. This should enable tracking down and identifying nodes that share illicit content.

1.4. Thesis Organization

This thesis is broken down into four subsequent chapters. Chapter 2 starts with a brief overview of the history of the topic including the technologies and legal issues that give rise to the problem that this thesis addresses. Once the problem is understood, Chapter 3 discusses the methodology used to offer one solution to the problem. Chapter 3 will discuss all the tools and algorithms used throughout the exploration of this topic. Chapter 4 then presents the results of the various experiments and explains the statistical analysis used to evaluate the effectiveness of the proposed solution. Finally, Chapter 5 offers a brief assessment of the work with recommendations for the next steps.

CHAPTER 2

Background and Literature Review

Before discussing how it may be possible to improve network searches for contraband, it is necessary to understand some of the existing work and background in this field. In this chapter, a brief history of how these networks came to be will be discussed; followed with how criminal elements began to disseminate illicit content. Understanding the type of content and its quantity on the network will be needed to successfully simulate the network and test possible improvements. The synthetic network must also be built using techniques found in a real network and thus, a review of surveys taken on the existing networks will shape the methodology used to build the networks tested in this thesis. Finally, there has been considerable work already done on how to improve search results in these networks and this chapter will conclude with a review of a few of those approaches.

2.1. History of Peer-to-Peer Networks

Since the inception of the Internet in the 1970s, it has been used to greatly expand the availability of knowledge for those with access to it. While it started out primarily as a research tool, its introduction to the public at large has irrevocably changed how information is exchanged. As the majority of Internet users switched from researchers to the general public, there was a large-scale increase in the material available, including files with illicit content.

There have been numerous tools created to improve the searching and sharing of information over the Internet. Napster, which had been introduced in 1999, popularized the concept of peer-to-peer networking as a way to independently share files with other users without posting that data on a remote server. Prior to this, users largely depended on tools like USENET and IRC to post files they wanted to share. Later, Yahoo and AltaVista allowed these networks, in addition to the World Wide Web, to be searched for content posted by others. Napster fundamentally changed this by allowing those that used the software to share files amongst each other without another server to host their content.

Napster however was not an unstructured [**4**] peer-to-peer network. The client would connect to a central server that would then coordinate the querying and retrieval of music among the numerous participants in the network. This kind of structured network had numerous benefits as it could coordinate the connections of its users to allow greater availability of files. However, to achieve this, Napster required a central control point that could reliably direct the connections and searches of peers in the network. The servers that enabled a more efficient network were also a potential point of failure that could be exploited to shutdown the network. The Napster network was in fact shutdown when Napster Inc. failed to fulfill its obligations to monitor user behavior for illegal activity per court order at the close of A&M Records Inc. versus Napster Inc [**3**]. Avoiding a central-control mechanism was a driving force in the popularity of the Gnutella protocol, which grew as users joined the network with no external direction given to peers as they participated in the network.

Gnutella and unstructured [**4**] peer-to-peer networks in general have a number of properties that must be considered when analyzing the participants or content in the network. Foremost amongst these properties, are the “loose rules” [**4**] imposed on peers that seek to join the network. Gnutella, in particular, has a short set of protocols that must be implemented for a node to join the network. Once a node becomes part of the Gnutella network, it is trusted to pass messages as prescribed in the protocol with no built-in mechanism for verification. This lack of centrally imposed organization on the network has a primary benefit of not having a single point of failure. Research has shown that certain attacks could be used against these networks, and render the network worthless to a large majority of users [**7**]. However there have been no documented cases of a widespread disruptive attack against the Gnutella network. Data persistence in the network is not addressed by an unstructured protocol such as Gnutella either. Thus the network does not guarantee the availability or reachability of any file on the network which makes tracking of file movement very difficult.

2.2. Child Pornography on Peer-to-Peer Networks

As described above, peer-to-peer networks offer significant opportunities for clients to share in a very democratic environment that maximizes user freedom to exchange files. The decentralized nature of these networks poses a growing challenge to law enforcement's ability to identify purveyors of child pornography. The sense of anonymity on these networks combined with their dynamic nature has attracted criminal elements.

There has been growing prosecution of these offenders since the adoption of the Child Pornography Prevention Act of 1996. The number of cases has increased from 113 in 1996 to 2,500 in 2009, an increase of 2,050% [**6**]. The United States General Accountability Office (GAO) also found child pornography to be easily accessible on peer-to-peer networks using a total of 15 keywords to identify 692 out of 1,627 total images as child pornography [**5**]. However it has been shown that child pornography falls far short of other activity on the network. A survey of peer-to-peer networks for pornographic videos found that of 507 video files found, 3.7% involved child pornography [**5**]. Another survey in 2006 found that only 1.6% of searches and 2.4% of responses included illegal pornographic material [**15**] as defined in the United Kingdom, which includes video or photographs that sexually exploit children. This material appears to only be shared by a small community of users. The 2006 survey found that 57% of those sharing illegal material exclusively shared such files, while only 17% that shared such content had greater than 50% legal files.

2.3. Existing Studies of Peer-to-Peer Networks

In order to enhance the position of a small set of the nodes in a network that we can control, it is necessary to understand the current state of the Gnutella network. Numerous

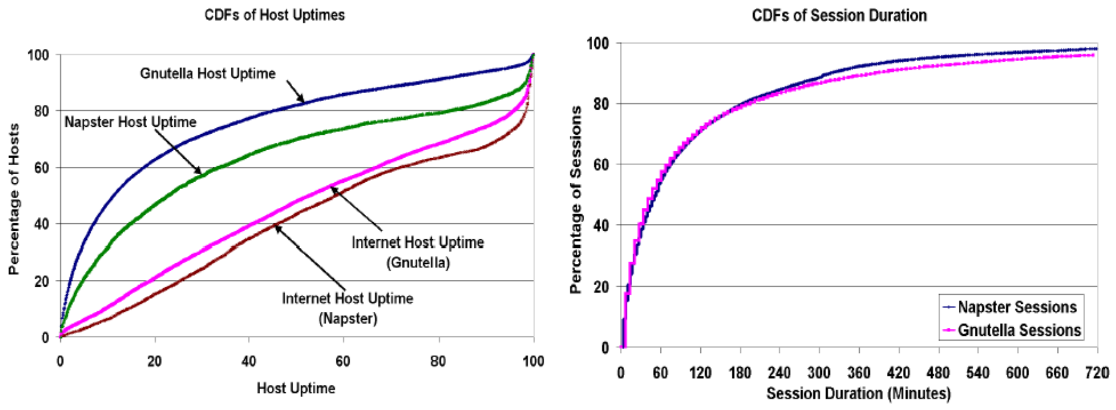


FIGURE 1. Left: IP and application level up-time of peers. Right: Distribution of session durations. [**8**]

surveys of this network have produced a variety of data reports over the years. A measurement done in 2001 by Saroiu et al. [**7**] measured the Gnutella network over the course of eight days. In that period, their tool reported 1,239,487 unique Gnutella peers at 1,180,205 unique IP addresses in the network. After identifying the scale of the network, active measurements were taken to assess a peer's availability and resources. 17,125 Gnutella peers were monitored over 60 hours for latency, up-time, and files shared. It was found that a full 25% of nodes in the Gnutella network shared no files. Additionally, the application up-time and session length of monitored peers were documented. These results give a practical view of the amount of network churn that occurs in the Gnutella network. Saroiu et al. [**7**] discussed the distribution of files available on the network, but only categorized files based on their total size rather than on content. As this project is concerned with improving network position relative to file content, the size of files is not germane to the problem.

Further analysis of the Gnutella network has shown that it has certain properties that can be used to enhance the simulation of those networks. Jovanović [**8**] showed that the Gnutella network, like other large self-organizing networks, can be approximated with four Power Law characteristics (i.e. of the form $y = x^a$). Jovanović [**8**] also collected data from the Gnutella network to examine its clustering coefficient and diameter. The diameter of a network describes the average distance in node to node hops needed to reach a "'sufficiently large' portion of a network."

The four power-law attributes that Jovanović [**8**] outlined were the “rank exponent,” “out-degree exponent,” “hop-plot exponent” and the “eigen exponent.” The first power-law defines the degree, d_v, of each node (v) which is proportional to the rank (r) raised to the power R (i.e. $a = R$). The rank of a node is the index of the node in a list of nodes ordered by decreasing degree. Power-law 2 says that the frequency of the out-degree, f_d, is proportional to the out-degree value (d) raised to the power O (i.e. $a = O$). Power-law 3 says that the “total number of pairs of nodes P(h) within h hops is proportional to the number of hops to the power of a constant H” (i.e. $a = H$). The final power-law finds that the eigenvalues, e_i, of the network are proportional to the order, i, raised to the power of E (i.e. $a = E$). The analysis of the collected data for power-laws 2 and 3 are shown below. However the author omitted the results for 1 and 4, but reported that the Gnutella network did have these properties.

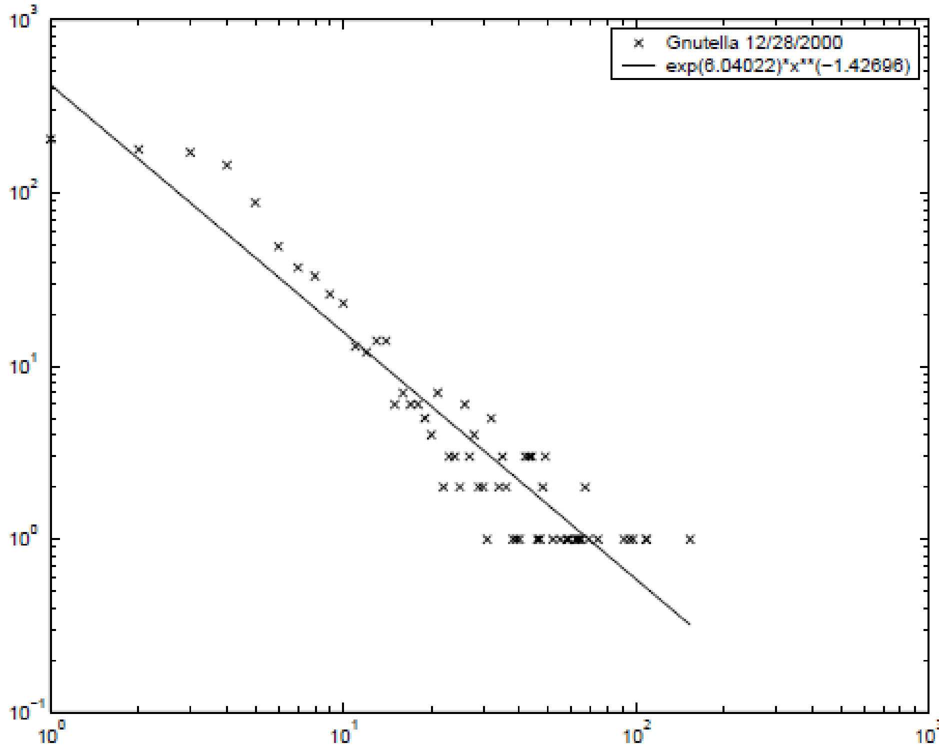
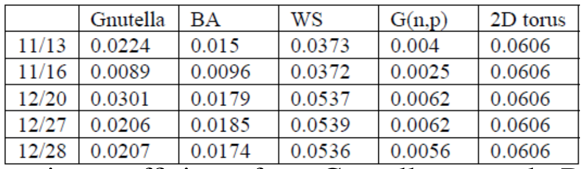


FIGURE 2. Log-log plot of frequency versus degree for 1 snapshot of the Gnutella network (Power Law 2)[**9**]



	Gnutella	BA	WS	G(n,p)	2D torus
11/13	0.0224	0.015	0.0373	0.004	0.0606
11/16	0.0089	0.0096	0.0372	0.0025	0.0606
12/20	0.0301	0.0179	0.0537	0.0062	0.0606
12/27	0.0206	0.0185	0.0539	0.0062	0.0606
12/28	0.0207	0.0174	0.0536	0.0056	0.0606

TABLE 1. Clustering coefficients for a Gnutella network, Barabasi-Albert, Watts-Strogatz, random graph, and 2D mesh topologies. [**9**]

Jovanović [**8**] defined the network's clustering coefficient for a graph of size k as the average of the clustering coefficients for all the nodes in the network. The clustering coefficient for each node *v* is defined as the number of cross edges in a Breadth First Search (BFS) tree of depth 1 with root node *v* divided by the maximum number of cross edges possible for a graph of size k. The formula for the maximum number of cross edges in a graph of size k is as follows:

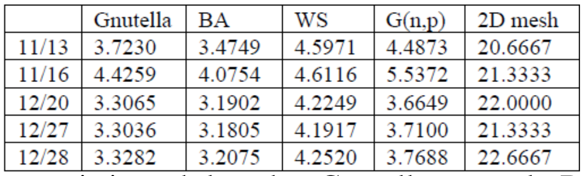
$$k!/((k-2)! * 2) - (k-1)$$

Jovanović [**8**] used a tool to take five snapshots of the topology of Gnutella in November and December of 2000. He found that the average clustering coefficient over that time was 0.02054 with a standard deviation of 0.00759.

Jovanović [**8**] discussed that the effective diameter provides a good way to apply the information learned from the power-law properties. In particular, the hop-plot exponent or power-law 3 is used to calculate the effective diameter, which could in turn be used to better tune the TTL for the network. Searching the network would be more efficient if the effective diameter and the TTL for the network are equal. The effective diameter is defined as follows where N is the number of nodes, E the number of edges, and H is the hop-plot exponent.

$$\delta_{\text{ef}} = (N^2/(N+2*E))^{1/H}$$

While there has been a number of surveys of the Gnutella network over the years, there appears to be a deficiency in the ability to quantify whether the simulated network is



	Gnutella	BA	WS	G(n,p)	2D mesh
11/13	3.7230	3.4749	4.5971	4.4873	20.6667
11/16	4.4259	4.0754	4.6116	5.5372	21.3333
12/20	3.3065	3.1902	4.2249	3.6649	22.0000
12/27	3.3036	3.1805	4.1917	3.7100	21.3333
12/28	3.3282	3.2075	4.2520	3.7688	22.6667

TABLE 2. Characteristic path length a Gnutella network, Barabasi-Albert, Watts-Strogatz, random graph, and 2D mesh topologies.[9]

representative of a real-world network. Most authors made qualified conclusions that their surveys would represent the networks studied as those networks evolved.

2.4. Simulating Peer-to-Peer Networks

There are numerous implementations available for simulating peer-to-peer networks. Most involve using properties observed in the real networks to generate simulated networks for testing at a smaller scale. It has proven difficult to model a real Gnutella network with more than one million nodes. Most simulations take the approach of taking the properties discovered by surveying a Gnutella network and using algorithms that will generate a network that contains those properties [**16**]. Many models attempt to scale the network down to less than 10,000 nodes [**10, 12**].

Another approach proposed was that the network could be built by modeling the actual protocol the network would use when forming connections [**16**]. In this case, the synthetic network was initialized using known data from a real network. Then, once the initial network was built, an algorithm scaled the network up or down. Scaling down targeted specific nodes for removal from the network while maintaining the initial network properties. Scaling up was accomplished by employing the network specific "bootstrapping" technique to find nodes that would be appropriate to connect to in a real network. In the case of the modern Gnutella v0.6 network, the bootstrapping process would find a super peer, also called an ultra peer, to initiate a connection. The new node would continue searching for a super peer to connect to until it was successful or it reached a limit on its allowed number of connections [**17**]. The authors used an inverse Barabasi-Albert approach for selecting super peers by assigning a higher weight to known super peers with the least number of connections. This technique allows for both the use of real data to simulate modifications to the network or to generate entirely new networks that should closely mimic a real network.

2.5. Techniques for Improving Search Results

Identifying users that are sharing illicit information on an unstructured peer-to-peer network such as Gnutella can be difficult due to the dynamic and anonymous nature of the network. While child pornography on the Gnutella network is an ongoing problem for law enforcement, this problem still only consists of a small percentage of overall users that download and traffic in illicit material. Thus, finding techniques that allow law enforcement to search a greater percentage of the network with fewer resources over a shorter period will improve deterrence by identifying perpetrators more quickly. As the problem continues to grow, there has not been much advancement in methodologies to identify contraband exchange on peer-to-peer networks. A few articles have proposed to modify the network to enhance information retrieval on peer-to-peer networks ([**9, 10, 11, 12**]). Although these techniques would help law enforcement track down criminals on future networks, they do not address the problem of tracking down offenders using existing protocols. This thesis will explore a technique to improve a node's position in a Gnutella network to enhance suspicious query results.

Much research has been done on how to create a better peer-to-peer network ([**9, 10, 11, 12**]) that would allow a client in the new network to track down information much more quickly. The biggest hurdle to applying these techniques to the existing Gnutella protocol is that they require global knowledge of all peers in the network. Some [**10**] also require knowledge of all the content on the network. The size and dynamic nature of peer-to-peer networks, including Gnutella, make it unlikely that a single client could have global knowledge of the network.

Using a particle swarm optimization approach Liu, Abraham, and Badr [**10**] attempted to maximize the disjointness of the network. They define disjointness as the amount of content that one node can share with another. Their particle swarm algorithm attempts to bring together nodes that can share the most information possible. The swarm technique rewards those that contribute more content to the network by connecting them closely with nodes that are sharing information they do not already have. However this approach would be difficult to apply to an existing network as it optimizes with complete knowledge of all nodes and each node's content in the network. The technique also necessitates that all nodes be configurable in order to move to an optimal solution.

Schmid and Wattenhofer [**11**] take an approach that is more relevant to the problem of improving the position in an existing network like Gnutella. They are also trying to refine the Gnutella protocol to improve overall network performance. Their approach offers more opportunities for application in an existing network as it is formulated with individual nodes acting only with local knowledge. They propose that some peers should take on the role of "beacons," which identify a cluster of nodes with shared interests. The beacons then identify themselves to new nodes attempting to join the network. Beacons are similar to the "super" peers in the v0.6 version of the Gnutella network as they can have a greater edge-degree in the network than other nodes. Clients in the network then tell inquiring peers about beacons that can be reached. Schmid and Wattenhofer utilize a strategy of having clients select new peers that maximize the number of different beacons that can be reached through their direct peers. While the technique is interesting, it requires that the communication protocol of Gnutella be changed and thus can not be used as presented in the existing network.

Li, Yang, Shi and Bai [**12**] also took the approach of applying local knowledge to optimize node positions. Their approach attempts to factor in the "popularity" of a document into their clustering strategy. They assign weights to the documents based on the frequency of their occurrence and then assign a score to peers that are candidates for connection that is a sum of the weights of that peer's documents. One problem with this scoring method is that it is not possible to know all documents that may exist in the network. Thus one cannot assign a score to a document that has not been seen, but that may exist on nodes being evaluated. The optimization is also greedier as each node is attempting to get connected to highly desirable nodes rather than simply connecting to a node that is already connected to a desirable node. They did not seem to address the fact that this may cause those that are not sharing many files to become starved for connections into the larger network. When evaluating the results, the clients were modified to perform a directed walk based on the weights of the peers instead of using Gnutella's flooding method. This would also have to be addressed if attempting to use the results in the existing network as it is not possible to change the behavior of other peers in the network. The ability to assign weights to documents would apply to the problem of illicit content in Gnutella as the goal is to find users sharing a particular kind of file and not just any files on the network. Thus, with that technique, the optimization can be focused on the users sharing specified information.

Finally [**9**] addressed improving node placement in a peer-to-peer network by proposing equations that could be applied locally. These equations had the advantage that they could be applied to specified nodes in the network, while leaving other nodes to function according to the current protocol. [**9**] suggested that responding nodes could be assigned a fitness or importance based on the results of the queries that were returned. They proposed modifying all nodes in the network to use these formulas in order to have each node move independently towards other nodes with shared interests. While it is not possible for one node to force a modification on another in the existing network, it is possible to request a connection between the node that has been modified and an unmodified node. Thus, the controlled nodes can move closer to other nodes they are interested in even if those nodes are not moving in a similar fashion. The fitness value of each node was composed of two factors:

(1) The percentage of all results that it had received from that particular node
(2) The distance to that node.

Both of those factors can be found in the existing Gnutella protocol. Nodes sharing files must identify themselves to those that are attempting to download files. In addition, the protocol passes information regarding the number of nodes that were passed through as the hops value in the node descriptor packet [**14**]. Clients on the network can use a variety of techniques to mask their true identity. In many cases, there would be methods available to law enforcement to use what data is provided for an actual download to track down the actual computer sending a file. The following formulas are used to calculate the fitness of each node that a client can identify. The $QueryHits_{p,q}(s)$ is the number of matching query results sent from node s to node p through node p's direct peer q. $QueryHits_p(s)$ is the total number of query results sent from node so to node p through all of p's direct peers. $nHops_p(s)$ is the number of hops between p and s. α and β assign a weight to previous values, thus preventing previously valuable nodes from being removed too quickly.

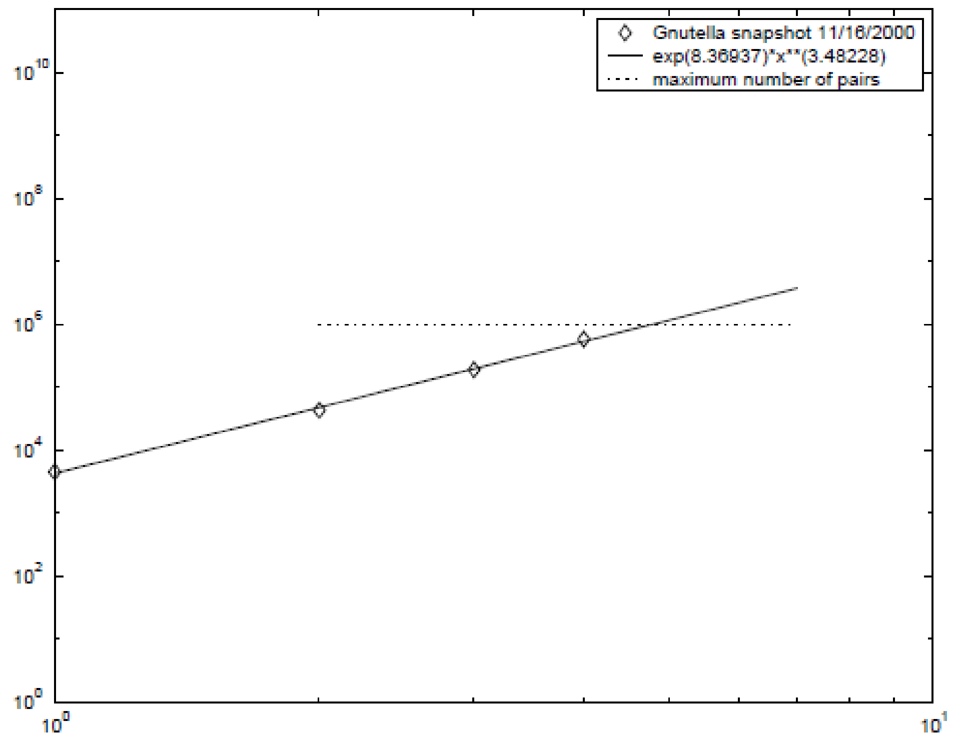
$$averNHops_p(q) = \sum_s (QueryHits_{p,q}(s) * nHops_p(s)) / \sum_s \sum_q QueryHits_{p,q}(s) \tag{2.5.1}$$

$$percQueryHits_p(s) = QueryHits_p(s) / \sum_r \sum_q QueryHits_{p,q}(r) \tag{2.5.2}$$

$$Imp_p(q,t) = \alpha * percQueryHits_p(q) / averNHops_p(q) + \beta * Imp_p(q,t-1) \tag{2.5.3}$$

2.6. Summary

This chapter reviewed the rise of peer-to-peer networks as a technological solution to the need to share content with little ability for other parties, including the government, to interfere. This characteristic fits in with the spirit of an Internet that enabled large scale exchange of information for all those connected to it. It also enabled less savory content to be easily distributed, increasing the interest of law enforcement in finding solutions to that problem. As the properties of these networks have been surveyed and opportunities for improvement were proposed, it became apparent that there is a need for improved methods in the current state of the arts to enable law enforcement to track down illegal file sharers. In the subsequent chapters, one solution will be implemented and its results will be validated using experiments on a simulated peer-to-peer network.



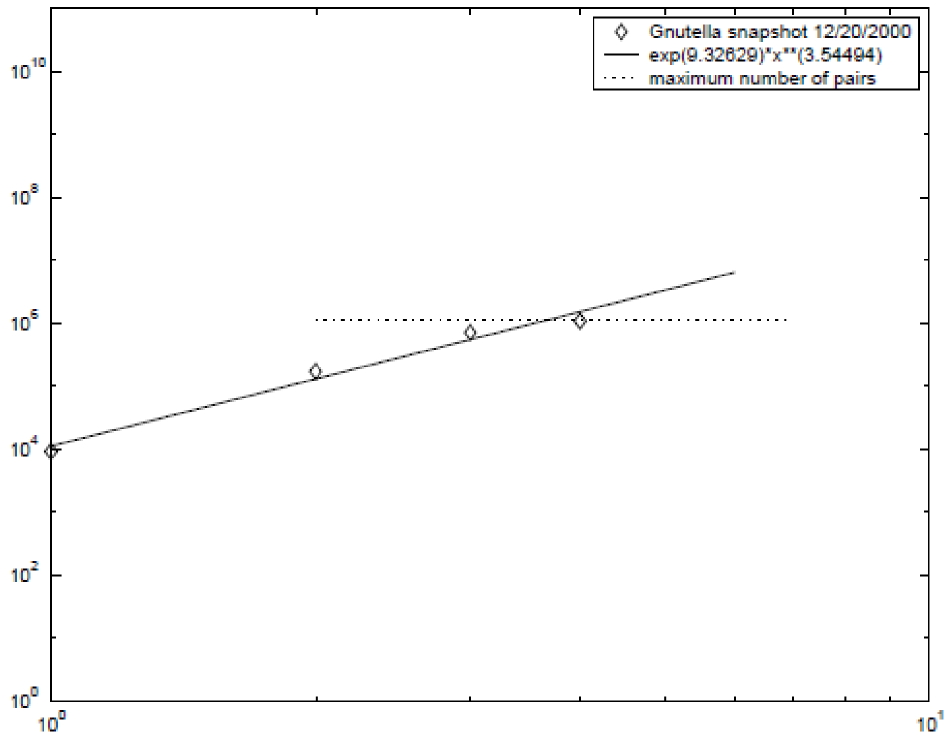


FIGURE 3. Log-log plot the number of pairs versus the number of hops for 2 snapshots (Power Law 3). [**9**]

CHAPTER 3

Methodology

Chapter 3, Methodology, will present the specific tools and algorithms that the team used to simulate a realistic peer-to-peer network and will describe the methodology to be followed in our simulations. If one were to attempt to reproduce the results for oneself, this chapter along with the referenced appendices should allow for the experiments to be repeated.

As discussed previously, there was little work in the existing literature that could be directly applied to the problem of improving search results in an existing network. Creating a new protocol offers many tangible benefits for researchers and law enforcement. Once a new network is created, there is the challenge of transitioning a sufficient portion of the existing users to the new infrastructure. Gnutella only gained popularity after Napster was shutdown and new protocols, such as BitTorrent, have further fractured the peer-to-peer community. Thus, this thesis operates without attempting to modify the Gnutella protocol, but instead creating a single intelligent client that should be indistinguishable from other clients on the network. The client's intelligence cones from its ability to dynamically change its placement within the peer-to-peer network using an algorithm to optimize that position in order to better intercept illicit traffic.

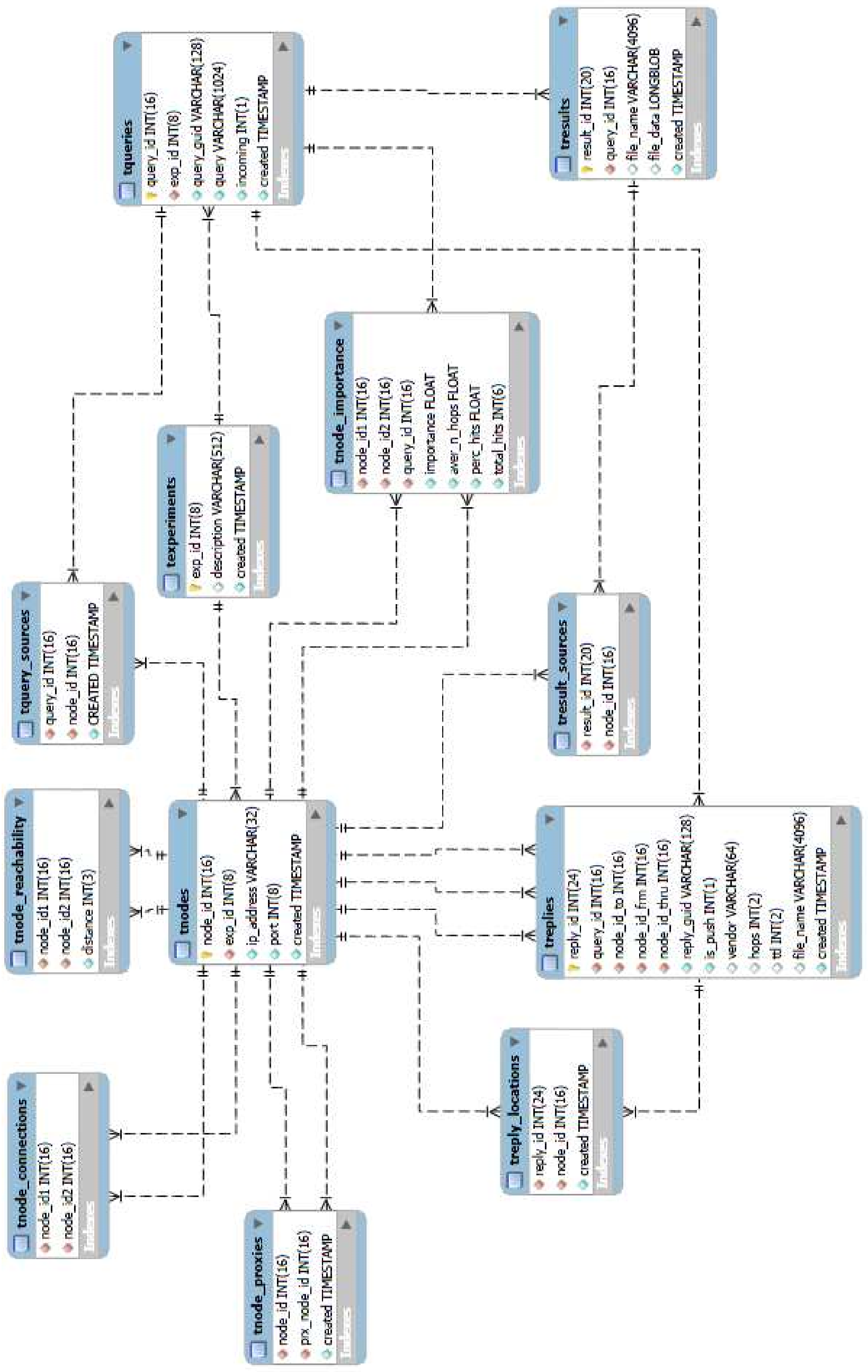
The problem of improving query performance is constrained by the information provided by peers through the existing Gnutella protocol and by the number of nodes in the network that can be instantiated by the operator. While it is possible to manipulate the messages initiated by or passed through a controlled client, this would not affect a significant portion of the messages passed on the network. The modification of messages carries a risk that other sophisticated clients could recognize the manipulation and stop responding to the probe [**14**]. With these restrictions in mind, the fitness formula and algorithms created by Ramanathan, Kalogeraki, and Pruyne [**9**] were adapted for use by a limited number of probe nodes in the network. These probes are listening nodes that can intercept data exchanged through them while adjusting their position in the network in order to identify nodes that are sharing illicit material.

3.1. Tools

The project was developed on the Java 1.5 platform in a Windows XP and Win7 environment. The Eclipse IDE for Java Developers was used to implement the Java code. Database structures were written in a simple text editor. The database objects were deployed to a MySQL 5.4 database using MySQL's command line client. Queries to extract results were also written in a simple text editor and run through MySQL Workbench 5.2. Results were analyzed and graphically depicted with a spreadsheet program, OpenOffice.org Calc. An algorithm provided by another student working on the project, Nick Miles, was written in Python and thus a Python interpreter was used to extract the necessary data.

Java was chosen due to the freely available documentation and support found on the web. Version 1.5 of Java was selected as it included Generics syntax, which allowed for more type safe code. This avoided type mismatch errors that often arose in Java programs that included Java Collections in prior versions. The project was commented using the javadoc tool to provide an easy reference for others to review the code. In addition, the java code profiler included with the Java Development Toolkit (JDK) was used to identify problem sections of the code. The profiler indicated that 80% of the time was spent writing to the database. Database tables and indexes were designed for data integrity and optimized query performance which could reduce insert performance. Modifying the design for improved insertion, would hinder analysis of the results which was deemed more important for this thesis. The application code could be modified to cache results in memory for writing to the database in order to improve execution time. However, this was not done for the simulation as the time to simulations run was acceptable and efforts to improve performance would be better spent on an implementation that operated on a real Gnutella network.

The MySQL database structures were developed to interface with a simple program built on top of the Limewire Gnutella client code. The database relational model is shown below. This program sent a query out on the network and then attempted to download all results returned from the network for that query. Information regarding the node that responded along with any complete files that were able to be retrieved were saved to the database. The program also stored any queries it received from the network to a table. These activities were not a part of the simulation, and thus the schema has numerous data elements that are not needed to collect simulation results. In addition the simulation required elements for network analysis that would be useless when working in a real network.


tqueries
query_id INT(16)
exp_id INT(8)
query_guid VARCHAR(128)
query VARCHAR(1024)
incoming INT(1)
created TIMESTAMP
tresults
result_id INT(20)
query_id INT(16)
file_name VARCHAR(4096)
file_data LONGBLOB
created TIMESTAMP
tnode_importance
node_id1 INT(16)
node_id2 INT(16)
query_id INT(16)
importance FLOAT
aver_n_hops FLOAT
perc_hits FLOAT
total_hits INT(6)
texperiments
exp_id INT(8)
description VARCHAR(512)
created TIMESTAMP
tquery_sources
query_id INT(16)
node_id INT(16)
CREATED TIMESTAMP
tresult_sources
result_id INT(20)
node_id INT(16)
tnode_reachability
node_id1 INT(16)
node_id2 INT(16)
distance INT(3)
tnodes
node_id INT(16)
exp_id INT(8)
ip_address VARCHAR(32)
port INT(8)
created TIMESTAMP
treplies
reply_id INT(24)
query_id INT(16)
node_id_to INT(16)
node_id_frm INT(16)
node_id_thru INT(16)
reply_guid VARCHAR(128)
is_push INT(1)
vendor VARCHAR(64)
hops INT(2)
ttl INT(2)
file_name VARCHAR(4096)
created TIMESTAMP
treply_locations
reply_id INT(24)
node_id INT(16)
created TIMESTAMP
tnode_connections
node_id1 INT(16)
node_id2 INT(16)
tnode_proxies
node_id INT(16)
prx_node_id INT(16)
created TIMESTAMP
Indexes

Default selections in the Installation Wizard for MySQL were used for the initial setup. The automated query tool required a modification to the database configuration after installation to enable larger files to be saved in the database. In particular, the *max_allowed_packet* was increased to 1GB from its default value of 1 MB to save files that were multiple megabytes in size in a single transaction. The "thesis" and "thesisv4" users were created to hold copies of the tables used in the simulation. See Appendix 1. Each user was setup to only allow connections from the local host. A change in configuration would be required to run the database on a separate machine from the simulation code.

Finally, to connect to the database from a Java application required the downloading of the "mysql-connector" for java from the MySQL website. This jar file was added to the CLASSPATH for running the Java application. Once the driver was loaded using normal Java methods, the standard Java database API could be used to communicate with the MySQL database.

3.2. Generating the peer-to-peer network topology

To begin, it was necessary to create a synthetic network that could be used to analyze the results of each algorithm. The algorithm written by team member Nick Miles [**22**] in Python that in turn used a modified Barabasi-Albert [**20**] algorithm was used to generate the networks. Due to the nature of the Gnutella v0.4 protocol, the Barabasi-Albert distribution applied within an algorithm that built the network from scratch would have generated some nodes with too high of a degree to accurately model a v0.4 network. Thus team member Miles' algorithm limited the number of edges for any single node at a level appropriate for Gnutella v0.4. The modified Barabasi-Albert algorithm has 3 inputs:

(1) the size of the network to be generated
(2) the minimum number of edges per node
(3) the maximum number of edges per node.

The function was modified to output the edges in a usable file format. A single 50,000 node network with 2 to 5 edges per node was created to test the position optimization algorithms. See Appendix 2.

In order to approximate a modern Gnutella v0.6 network, a variation of the bootstrapping mechanism proposed in [**16**] was developed to designate some nodes as ultra nodes. This algorithm insured that all nodes are connected to at least one super node, as well as maintain a ratio of super nodes to normal nodes that exists in a real Gnutella network of one super node for every three normal nodes [**16**]. This algorithm took 6 input values to control the nature of the network.

(1) the size of the network to generate
(2) the minimum number of edges for any node
(3) the maximum number of edges for a leaf node
(4) the maximum number of edges for a super node
(5) the percentage of all nodes that should be super nodes
(6) the percent chance that a super node will connect to another super node.

While there was data on the percentage of nodes in a Gnutella network that are super nodes, there was no data on the frequency of connections between super nodes. However, to avoid the network being fractured, it was generated with a 30% chance that each connection for a super node would also be a super node. See Appendix 3.

Based on the algorithms described, there were two variations of the network that the position optimization algorithms were tested on. Change in the network over time was neglected in these experiments. The only change in the network was caused by modifications initiated by the probe nodes. However the code was designed to allow behavioral classes to be implemented that would use an interface to control the behavior of any number of nodes in the synthetic network. This interface is how the probe behavior was implemented.

3.3. File Distribution

Another data-set was needed to simulate the distribution of files in the network. More difficult than knowing the true network topology is knowing the ever changing nature of files distributed on the network. New content on the Internet is generated very frequently and this data can quickly become available on a peer-to-peer network. However as discussed previously there have been surveys [**5, 6, 15**] on the availability of illicit material such as child pornography in real networks.

An algorithm (see Appendix 4) was developed to distribute contraband material based on this information. It was written to be configurable with parameters in case these values changed in the future. The algorithm took 11 parameters to generate the distribution.

(1) the list of keywords to generate contraband file names
(2) the list of keywords to generate normal file names
(3) percent distribution for each contraband keyword
(4) percent distribution for each normal keyword
(5) the number of peers in the network
(6) the minimum number of files found at a peer that shares files
(7) the maximum number of files on a peer that shares files
(8) the percent of nodes that share any files
(9) the percent chance that a node with files will have contraband
(10) the percent chance a node with contraband will exclusively share contraband files
(11) the percent of contraband found in a node sharing normal and contraband files

3.4. Simulating the network

To simulate the network, a single class was implemented that mimicked the messages passed in the Gnutella protocol in function calls. This was not an exact simulation as the messages were passed using function calls between individual instances of the GnutellaSimNode class. The functions were named after the message they simulated. These messages include ping, pong, query, queryhit and push. The push message was never implemented as the impact of firewalls on the ability of shared files was neglected. If a firewall does not allow a node on the real network to share data, then while that node may have contraband data, no one but the user would be able to access those files. Other approaches such as query analysis or using a “honey pot” would be needed to determine whether the user at nodes behind restrictive firewalls were collecting illegal content. However in a real network, a node that was forced to push a file to the probe, would require additional investigation in order to track down the clients operating behind the firewall.

In order to simulate more complex behavior, it would be desirable to have the nodes operating independently on separate threads rather than all being controlled by a single thread. The code would need modification to allow for multi-threaded access to internal members as the classes were not designed for such environments.

3.5. Probe Implementation

To implement the behavior of a probe node, there were hooks added to the GnutellaSimNode class to send messages to those interested in events at that node. In this way, the probe was able to react to all the messages that were received at a particular GnutellaSimNode. The interface for this functionality was defined by GnutellaSimNodeController. While these controllers could not alter the basic responses required by the Gnutella protocol, it could record results or initiate new messages on the network in response to these messages. While in a real network, a probe could also alter the behavior of a node to not comply with the Gnutella protocol, this was ignored since a modified client could also be detected and ignored by some nodes on the network [**14**].

The probe nodes were selected from the set of all nodes at random at the beginning of the simulation. Then a list of configurable queries was initiated one at a time from each probe. The probes would then collect all results that were returned. Once all queries had been sent and results collected, each probe would calculate the importance [**9**] values for each of its direct peers using Equation 2.5.3. After recording these values, an algorithm was run to select the direct peers, if any, to be replaced with a new peer.

3.6. Position Improvement Algorithms

Two algorithms were evaluated against a baseline exploration (**Algorithm** 4) of the network. **Algorithm 5** would randomly select new peers from the network to replace uninteresting peers. This could be accomplished in a real network using methods such as the host-cache functionality provided by Limewire [**17**]. While the entire network would not be available for selection, a new peer could be drawn from a significant pool of known hosts. **Algorithm 6** instead used ping messages to get lists of known peers from a peer that had previously returned contraband data. This is based on the assumption that if clients regularly search for these files that they would naturally gravitate toward a neighborhood of users sharing or collecting these files. There is, however, no known data that would support this model of natural neighborhood formation within the Gnutella network. The synthetic file distribution or network topologies did not assume that these neighborhoods existed.

3.6.1. Original Ramanathan Algorithm and Base Algorithm. The original algorithm defined by Ramanathan [**9**], as shown in **Algorithm** 3, was intended to be implemented at each node on the network. Since modifying the entire network is not possible, the algorithm was modified to account for a limited number of probes that could be placed on the network. In addition part of Ramanathan's algorithm was broken into modules that could be called from other algorithms being tested to improve the modularity of the proposed solutions. The base algorithm used to measure results against omitted any selection criteria, such that it would randomly replace any direct peer with a randomly selected node from the network.

3.6.2. Increased Random Exploration. Since the non-probe nodes on the network would not be attempting to move closer to the probes, the algorithm had to be modified in order to increase that exploration of the network (**Algorithm** 5). The first approach to do this was simply to replace low performing nodes with new nodes that are randomly selected from the network. These nodes could have been attached previously, as no history

Algorithm 1 Module Maximum Percent Query Hits

```
MaxPercentQueryHits(Node p, Node i_max)
{
// This algorithm will find the indirect peer of p
// that has the maximum percentQueryHits returning it as i_max
 for each indirect peer, i, that sent a queryhit to p
 {
  Pcurr = percentQueryHits(p, i)
  if Pcurr > Pmax
  {
   Pmax = Pcurr
   i_max = i
  }
 }
}
```

Algorithm 2 Module Get Minimum Importance

```
MinImportance(Node p, time t, Node d_min, Set Z)
{
// This algorithm finds the direct peer of p
// at the current time, t, that has the minimum importance
// returning the value as d_min
// Z also stores all peers that have a zero importance value
// which could include d_min
 Imin = infinity
 Z = empty set
 d_min = random direct peer of p
 for each direct peer, d
 {
  Icurr = Importance(p, d, t)
  if Icurr < Imin then
  {
   Imin = Icurr
   d_min = d
  }
  if Icurr = 0 then
  {
   add d to Z
  }
 }
}
```

Algorithm 3 Original Ramanathan Algorithm

```
Ramanathan_Algorithm(Node p, time t)
{
 MaxPercentQueryHits(p, s)
 MinImportance(p, q, t)
// Greedy replacement of direct peer q by s
 if percQueryHits(p, s) >= percQueryHits(p, q) then
 {
  if addConnection(p, s) successful then
  {
   if NumberConnections(p) > MAX_CONNECTIONS(p) then
   {
    dropConnection(p, q)
   }
  }
 }
}
```

Algorithm 4 Baseline Random Placement Algorithm

```
BaseAlgorithm(Node p)
{
 Select random direct peer, d, from p
 Select random node, r, that is not a direct peer of p
// Add r as a direct peer of p
 if addConnection(p, r) successful then
 {
// Remove d as a direct peer of p if now exceed maximum # of connections
  if NumberConnections(p) > MAX_CONNECTIONS(p) then
  {
   dropConnection(p, d)
  }
 }
}
```

is maintained about previously attached direct peers. However, the large number of peers to select from in any peer-to-peer network makes re-attaching to an old peer unlikely, and the dynamic nature of the network may be desirable as new content or peers could now be accessible through the peer.

3.6.3. Targeted Exploration. The other method (**Algorithm** 6) uses ping messages in an attempt to discover the neighbors of a contraband node. This would prove more useful if

Algorithm 5 Increased Random Exploration

```
IncreasedRandomExploration(Node p, time t)
{
 MaxPercentQueryHits(p, i_max) // Algorithm 1
 MinImportance(p, t, d_min, Z) // Algorithm 2
// This comparison only considers direct results and not results that
// may have been received through d_min. However the importance
// calculation includes all results through d_min and thus d_min is the worst
// of current direct peers. This is a greedy selection of new peers
 do_exploration = false
 if percentQueryHits(p, i_max) >= percentQueryHits(p, d_min) then
 {
  if addConnection(p, i_max) successful then
  {
   if NumberConnections(p) > MAX_CONNECTIONS(p) then
   {
    dropConnection(p, d_min)
   }
  }
  else
   do_exploration = true
 }
 else
  do_exploration = true

 if do_exploration = true then
 {
// If could not find a better node that would accept a connection then randomly
// with 50% probability replace nodes that are currently not returning any results
  for each node, c, in Z
  {
   r = random R[0, 1)
   if r < 0.5 then // With probability 50%
   {
    Select random node in network, n
    if addConnection(p, n) successful then
    {
     if NumberConnections(p) > MAX_CONNECTIONS(p) then
     {
      dropConnection(p, c)
     }
    }
   }
  }
 }
}
```

there are a few known existing nodes at the start. However in the course of investigating the Limewire client, there were numerous sources available that provided lists of known peers in peer-to-peer networks. In addition, the Limewire client also queries known host caches at start up if it cannot find a peer on its own. While there may be situations where a host will not respond to ping messages, one can also use an "X-Try" header message [**17**] in Limewire clients to find other peers from a host that is refusing a connection. Other client implementations may not provide this alternative.

The methods implemented in this chapter promise to provide a good basis for accomplishing the objective of improving law enforcement's capability to track down purveyors of child pornography on peer-to-peer networks. The tools used are widely available and well understood by those in the field. Thus even if new technologies are desired, the tools used here should provide an easy transition to any other technology. With the neighbor selection algorithms above in mind, it is hoped that the experimental results will bear out the hypothesis that these techniques can improve search and detection results for those tracking down illicit material on peer-to-peer networks.

Algorithm 6 Increased Targeted Exploration

```
IncreasedTargetedExploration(Node p, Integer QueryTTL, time t)
{
 MaxPercentQueryHits(p, i_max) // Algorithm 1
 MinImportance(p, t, d_min, Z) // Algorithm 2
// This comparison only considers direct results and not results that
// may have been received through d_min. However the importance
// calculation includes all results through d_min and thus d_min is the worst
// of current direct peers. This is a greedy selection of new peers
 do_exploration = false
 if percentQueryHits(p, i_max) >= percentQueryHits(p, d_min) then
 {
  if addConnection(p, i_max) successful then
  {
   if NumberConnections(p) > MAX_CONNECTIONS(p) then
   {
    dropConnection(p, d_min)
   }
  }
  else
   do_exploration = true
 }
 else
  do_exploration = true

 if do_exploration = true then
 {
// If could not find a better node that would accept a connection then randomly
// with 50% probability replace nodes that are currently not returning any results
// by sending a ping to i_max
  for each node, c, in Z
  {
   r = random R[0, 1)
   if r < 0.5 then
   {
    initiatePing(p, i_max, TTL = FLOOR(QueryTTL / 2))
    for each node, o, that respond with a pong to p
    {
     if addConnection(p, o) successful then
     {
      if NumberConnections(p) > MAX_CONNECTIONS(p) then
      {
       dropConnection(p, c)
      }
      exit loop
     }
    }
   }
  }
 }
}
```

CHAPTER 4

Experimental Results

In this chapter the experimental methodology and analysis of the results will be presented. The factors used to measure each algorithm's success in improving search results will be detailed, including how they were measured or calculated during or after the simulation. In addition, this chapter will present a statistical analysis to evaluate whether or not each of the tested algorithms was effective.

4.1. Experimental Methodology

For each experimental run, the network was loaded and the files distributed amongst all nodes in the network. N_p nodes with connections and no contraband were then randomly selected to serve as probes. This approximates bootstrapping N_p probe nodes randomly. The simulation then proceeded to iterate through a list of queries, collecting the query results for the importance calculation. After all queries had been processed, the simulation attempted to connect to new peers and recorded the results for that generation. The program also calculated the distance to all contraband peers from all probe nodes using Dijkstra's shortest path algorithm [**21**]. The number of generations, probes and list of queries could all be controlled using the input parameters to the simulation. **Algorithm 7** for running each experiment is shown below followed by a flowchart (**Figure 1**).

These algorithms were tested against two network topologies. The first topology tested was built using Nick Mile's algorithm (see Appendix 2) [**22**] and was set up to mimic a Gnutella v0.4 network. The second topology was built using the algorithm in Appendix 3 and was generated using techniques that simulated a more modern Gnutella network with version 0.6 of the protocol. Both networks contained 50,000 nodes and were populated

Algorithm 7 Pseudocode for Experimental Run

```
ExperimentRun(Number generations,
                         Algorithm PeerSelectionAlgorithm,
                         Set Queries,
                         Set Probes,
                         Integer QueryTTL)
Outputs: Network G,
                Importance for each probe in Probes,
                Shortest distance from each probe in Probes to all contraban
{
 t=0
 while t<generations
 {
  t += 1
  for each query, q in queries
  {
   for each probe, p in probes
   {
    // p initates query q with TTL of QueryTTL
    initiateQuery(p, q, QueryTTL)
    Collects results in Set Replies
   }
  for each probe, p in probes
  {
   calculateImportance(p, t) // Using equation 2.5.3
   selectNewPeers(p, Replies, PeerSelectionAlgorithm) // Algorithm 4, 5 or 6
  }
// All contraband nodes are known since this is a simulation
// Thus calculating Dijkstra's shortest path is used as
// a baseline for measuring how results improved between generations
// Dijkstra's shortest path (Algorithm 8)
  Generate Dijkstra's shortest path from each probe to all contraband nodes f
  Save modified network state, G, for t
  Save distance from Dijkstra's calcuation for each probe in t
  Save importance for each probe in t
 }
}
```

with the same file distribution. Each network was tested with 50 and 100 of the total nodes in the network being selected as probe nodes. The two algorithms also varied the α value for the Ramanathan importance calculations in Equation 2.5.3 to measure its influence on the results. The α values tested were 1.0, 0.9 and 0.8. The control group ran the same

Algorithm 8 Dijkstra's Shortest Path [**21**]

```
Algorithm ShortestPath(NodeSet V)
Output: Set of minimum distances, D, for each v in V
S <- {v};
D[v] <- 0;
for each v in V - {v} do D[v] <- l(v, v);
while S != V do
{
  choose a vertex w in V - S such that D[w] is a minimum;
  add w to S
  for each v in V - S do
  {
   D[v] <- MIN(D[v], D[w] + l(w,v)) // l(w,v) always 1 in our case
  }
}
```

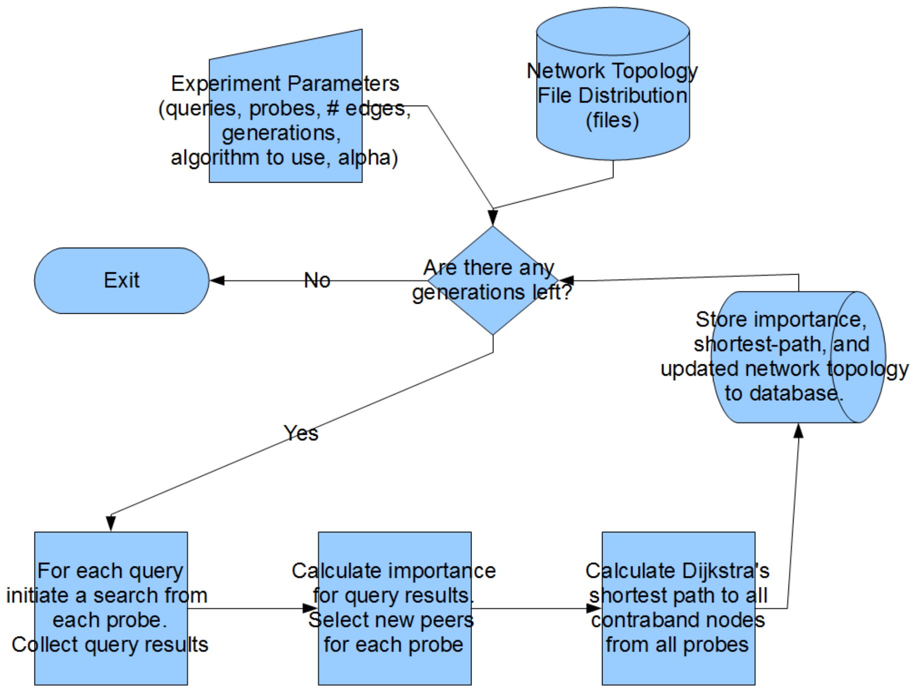


FIGURE 1. Flow chart for experimental run (**Algorithm** 7)

experimental algorithm above, but randomly replaced direct peers with new nodes instead of using the importance calculations to select replacements.

A null hypothesis was formulated to test each algorithm against the control group that used random exploration (**Algorithm** 4). The null hypothesis stated, "the algorithm to improve probe positioning in the network does not improve contraband search results." Each group was tested with the same parameters for 10 runs. Statistical analysis was then performed to test the null hypothesis for each algorithm and its input α value.

4.2. Quality Metrics

Six quality metrics were used to measure the viability of the algorithms implemented. These metrics were as follows:

(1) percent of contraband files discovered
(2) improvement in percent contraband files discovered
(3) percent of contraband nodes discovered
(4) improvement in percent contraband nodes discovered
(5) average distance to all contraband nodes
(6) average distance to reachable contraband nodes.

Each metric was evaluated against the random placement using a t-test to calculate the p-value for each algorithm [**19**]. The t-test with a t-distribution was chosen because it allowed for smaller samples to be compared, where the variance of each sample could be different.

The first metric quantifies the number of total contraband files that could be found using a set of parameters. The file distribution generated for all experiments had 20,381 contraband files allocated among a total of 3,724,405 files in the network. Just as in a real network, only a fraction of all nodes had any files with 15,015 nodes sharing one or more files.

The second metric, the percent improvement in total contraband files, complements the first metric since it quantifies how the probe's positions were improved over the course of the experiment with regard to content discovered. This measure was calculated by dividing the difference of the percent contraband found between the first and n^{th} generation by the percent found in the first generation:

(4.2.1)

$$PercFileImpr = (PercContraFiles(n) - PercContraFiles(0))/PercContraFiles(0)$$

Next the percentage of all contraband nodes that had been discovered by any probe was determined. More important than discovering all contraband files, is being able to find all nodes sharing contraband. If the supply of contraband material can be significantly reduced, then queries for the data will stop returning results. Only 69 of the 50,000 nodes on the network contained any contraband content. As with the second metric, the fourth metric gives insight into how quickly the technique can improve results by measuring the increase in contraband nodes found during the run. The fourth metric was calculated using the following formula:

(4.2.2)

$$PercNodeImpr = (PercContraNodes(n) - PercContraNodes(0))/PercContraNodes(0)$$

The fifth and sixth metrics are very similar as they represent the average distance between the probes and contraband nodes. The fifth metric measures the average distance from each probe to all contraband nodes in the network. $DijkstraDist(p,c)$ is the distance calculated by Dijkstra's algorithm in hops between probe p and contraband node c. $TotalContrabandNodes$ is the total number of contraband nodes in the network. $TotalReachableContraba$ is the total number of contraband nodes that are less than the query TTL value in hops from a probe.

(4.2.3)

$$averDistToContra(q) = \sum_{p} DijkstraDist(p,c)/TotalContrabandNodes$$

(4.2.4)

$$averDistToReachContra(q) = \sum_{p} DijkstraDist(p,c)/TotalReachableContrabandNodes$$

The overall goal is to try to move the probes closer to as many contraband nodes as possible. The sixth metric measures the average distance from each probe to all reachable contraband nodes. "Reachable" simply means that the contraband nodes are close enough to be queried from the probe based on the input TTL values. Shorter distances to known contraband nodes could increase the likelihood that queries for contraband in addition to replies to incoming queries could be seen by surrounding probes.

4.3. Gnutella v0.4 Topology

4.3.1. Contraband Content Discovered. The content distributed in the simulated network had 0.547% contraband. This is actually significantly less than what was reported on real networks [**5, 15**]. However this should not impact our results, as file searches for contraband assumed that all results from contraband queries were in fact contraband. Other files on the network were generally ignored by the queries. It would be possible to generate keywords that could return both types of files in the simulation, but this was neglected as it provided no additional value in assessing the quality of the algorithms. In a real client, the downloaded content would have to be verified as contraband. There are some known methods for doing this verification [**18**].

The file distribution also designated six types of contraband files that could be uniquely identified by a keyword. The search of the network used four of the keywords as queries. The four keywords that were used in the experiments were BAD, AWFUL, HORRID and MALICIOUS. **Table** 1 lists all six keywords and their frequency relative to all contraband files.

Keyword	% Distribution
BAD	40%
TERRIBLE	30%
AWFUL	20%
HORRID	5%
WORST	3%
MALICIOUS	2%

TABLE 1. Contraband keywords with their frequency in the network

Figure 2 and **3** show the results on the Gnutella v0.4 topology for each algorithm against 50 and 100 probes respectively. Random searching (**Algorithm** 4) with 50 probes only returned an average of 46.97% of all contraband content across all runs with a standard deviation of 5.82%. Just from visual inspection, the "Increased Random" approach (**Algorithm** 5), appears to be performing better with averages ranging from 60.98% up to 61.96%. Doubling the number of probes did not yield better results with the "Increased Random" having averages between 65.68% and 66.00%. **Figure** 9 and **10** indicate the overall improvement resulting from each algorithm from the start of the experiment until the end.

Algorithm	Figure Label
4	Random
5	IncRand
6	IncTarg

TABLE 2. Algorithm's label in the figures 2-5, 7-10, and 16-27

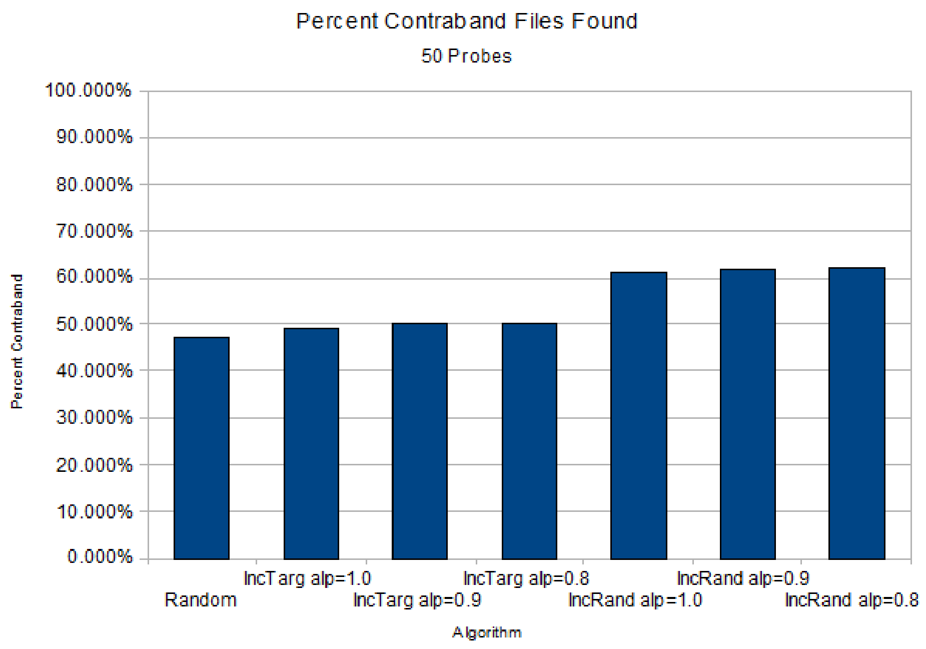


FIGURE 2. Percent Contraband Files discovered using 50 probes in v0.4 network

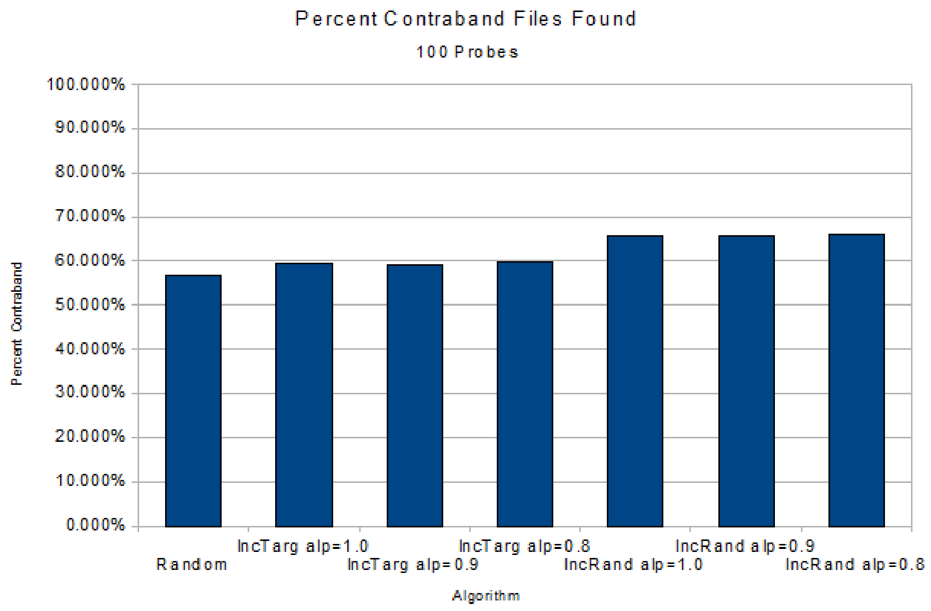


FIGURE 3. Percent Contraband Files discovered using 100 probes in v0.4 network

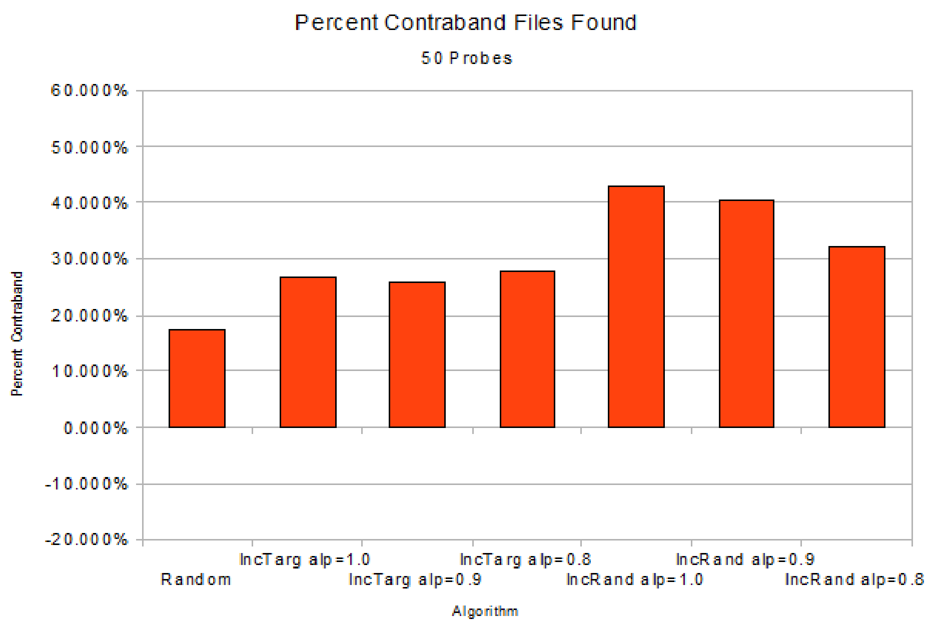


FIGURE 4. Percent Improvement in Contraband Files queriable using 50 probes in v0.4 network

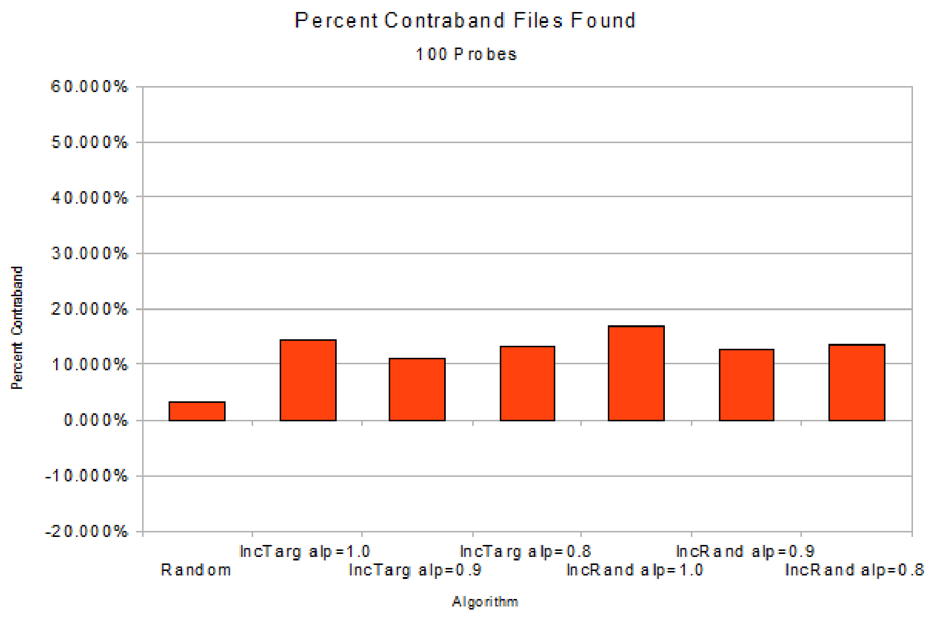


FIGURE 5. Percent Improvement in Contraband Files queriable using 100 probes in v0.4 network

4.3.2. Contraband Nodes Discovered. Metrics 3 and 4 (**Figures** 7 through **10**) will generally be of greater interest, as identifying all nodes sharing contraband is more important than simply finding all possible contraband content. As one can see from **Figures** 7 and

8, **Algorithm 5** or “Increased Random” searching produced the best results in the Gnutella v0.4 topology. Looking at the edge frequency in the Gnutella v0.4 network (**Figure** 6), one can see that most nodes have a nearly equal number of edges, and thus each node is roughly equivalent. Thus **Algorithm** 5, random exploration, will provide a greater chance to find a node outside a locally formed neighborhood than **Algorithm** 6, targeted searching.

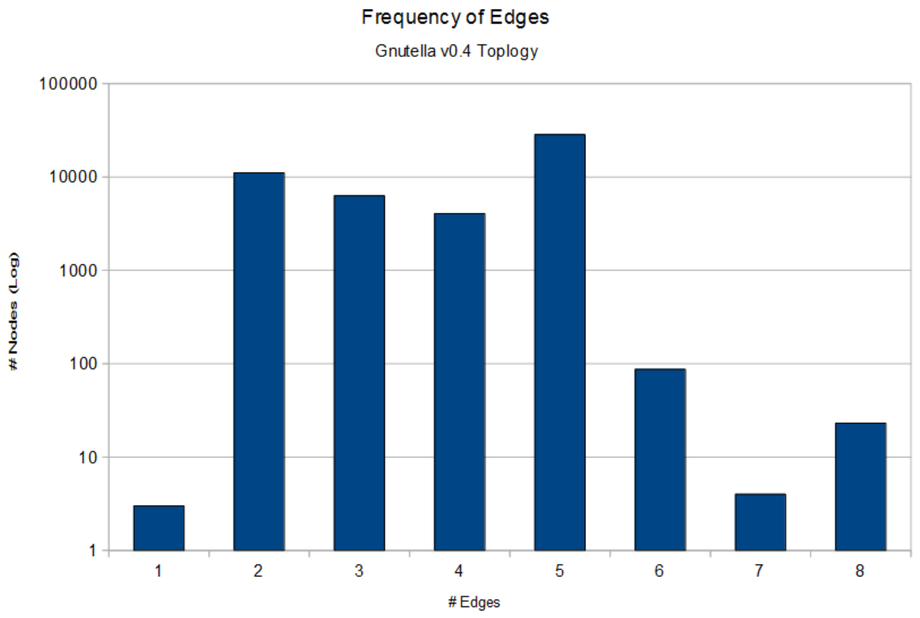


FIGURE 6. Edge number distribution for simulated Gnutella v0.4 network

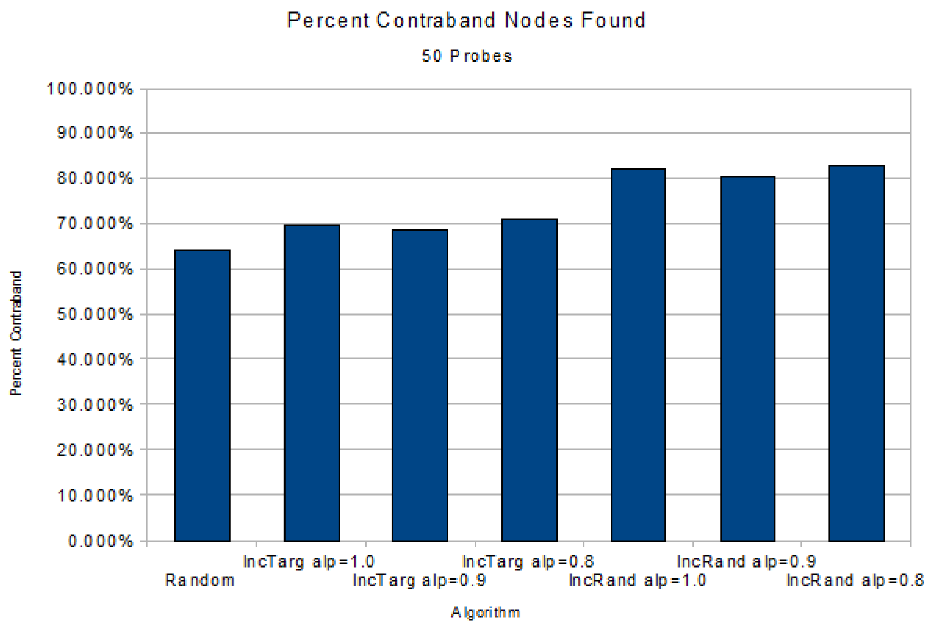


FIGURE 7. Percent Contraband Nodes discovered using 50 probes in v0.4 network

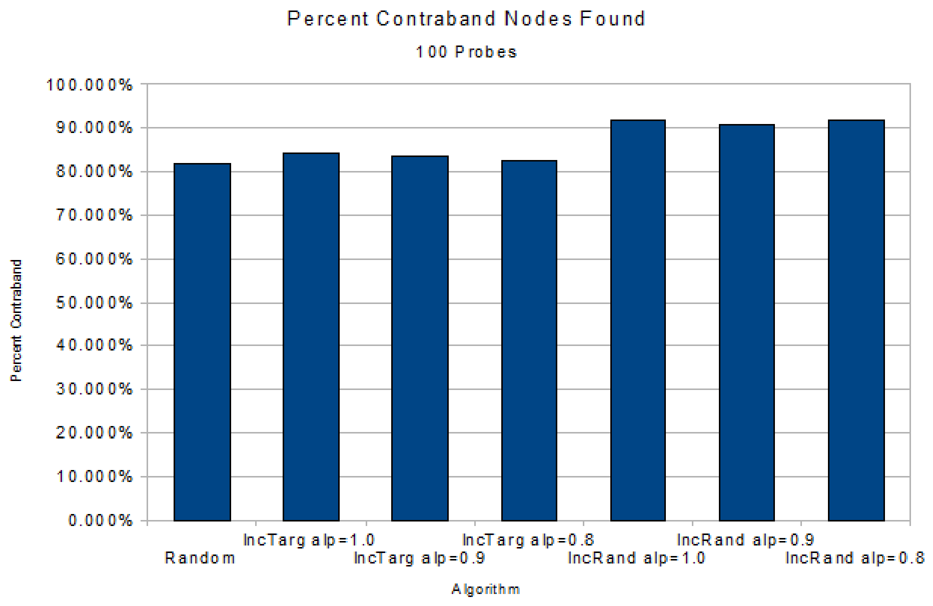


FIGURE 8. Percent Contraband Nodes discovered using 100 probes in v0.4 network

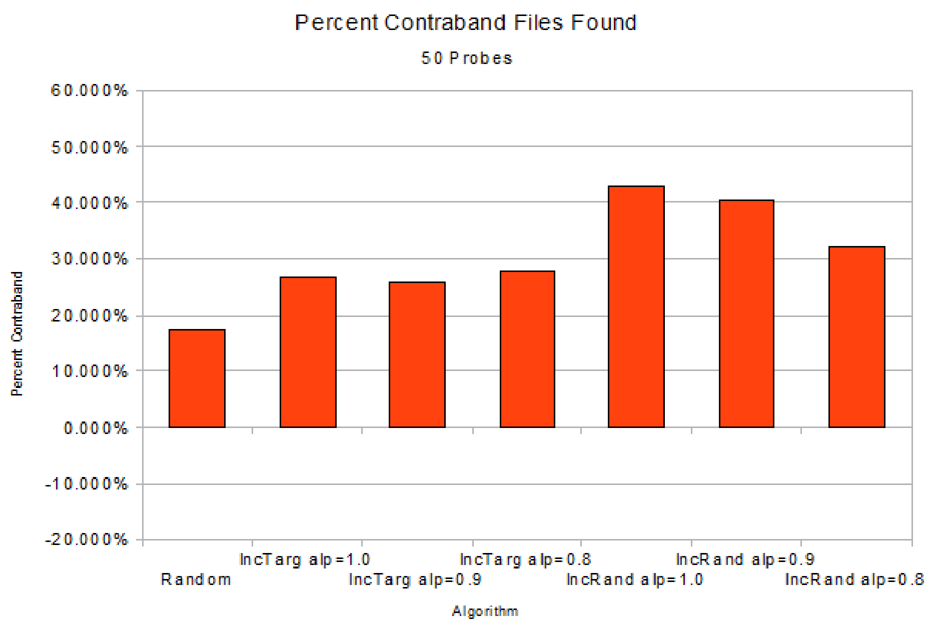


FIGURE 9. Percent Improvement in Contraband Nodes reachable using 50 probes in v0.4 network

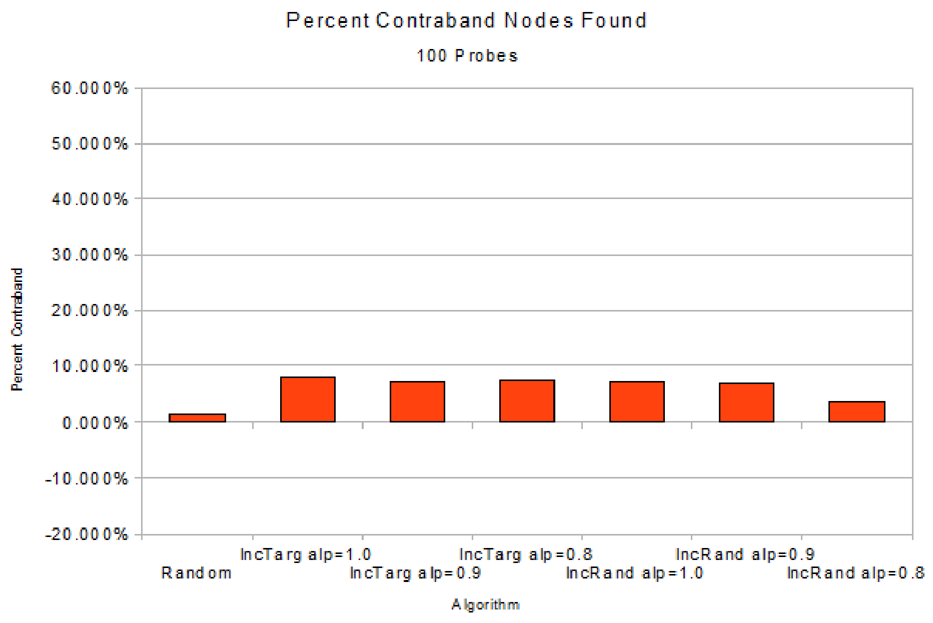


FIGURE 10. Percent Improvement in Contraband Nodes reachable using 100 probes in v0.4 network

4.3.3. Distance to Contraband. Both algorithms for each α value appeared to have similar success in shrinking the average distance from probes to contraband. Assuming that network nodes are not programmed to ignore increased TTL values, this information could be used for greater exploration by being able to increase the TTL used for queries by a small amount, and thereby reach additional contraband nodes.

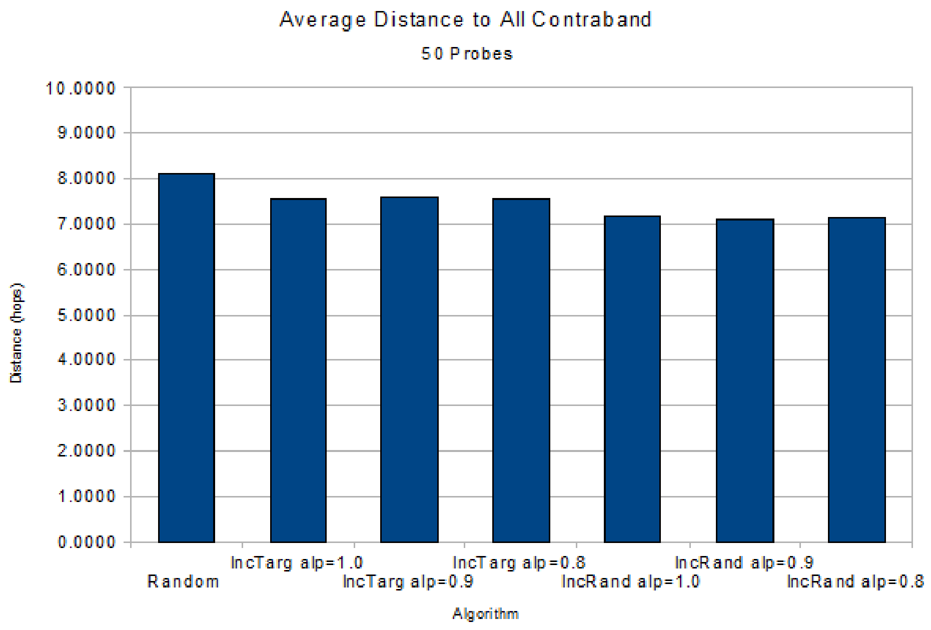


FIGURE 11. Average distance to all contraband nodes using 50 probes in v0.4 network

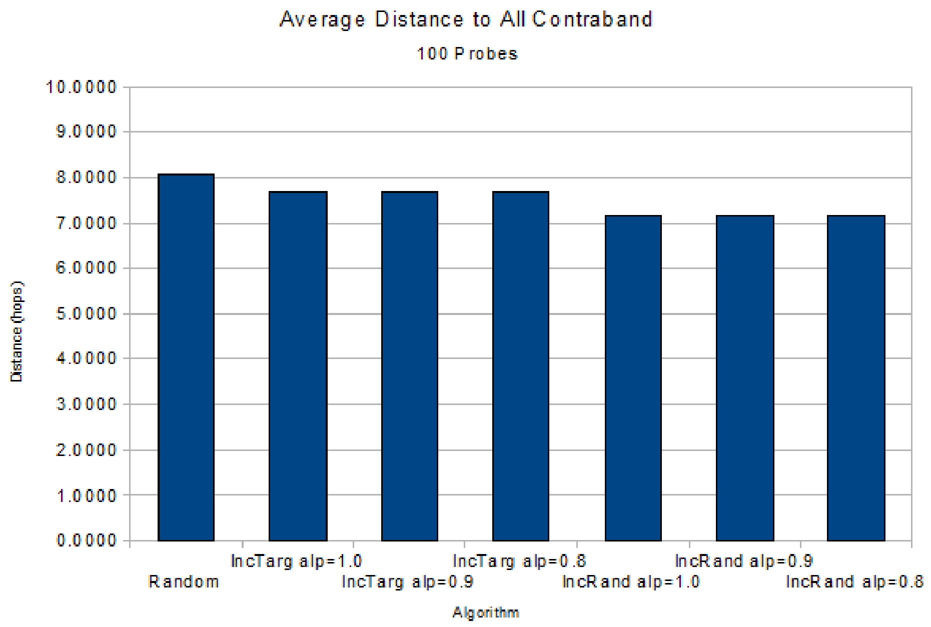


FIGURE 12. Average distance to all contraband nodes using 100 probes in v0.4 network

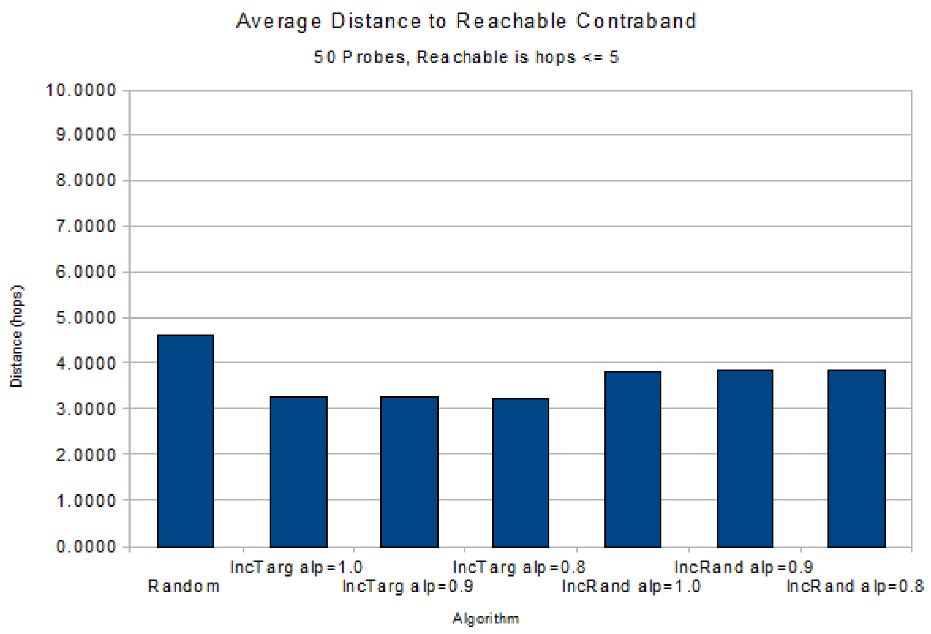


FIGURE 13. Average distance to reachable contraband nodes using 50 probes in v0.4 network

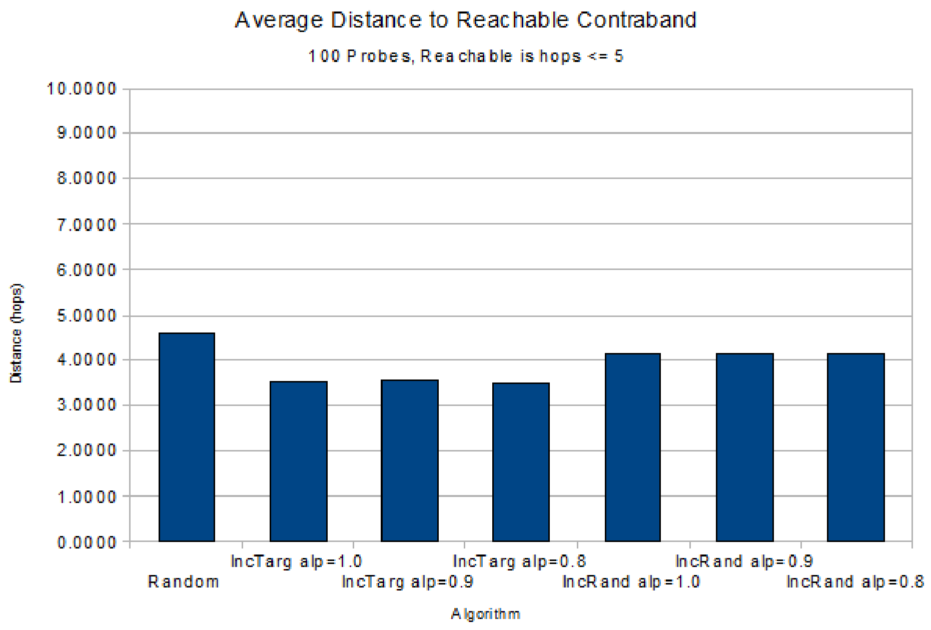
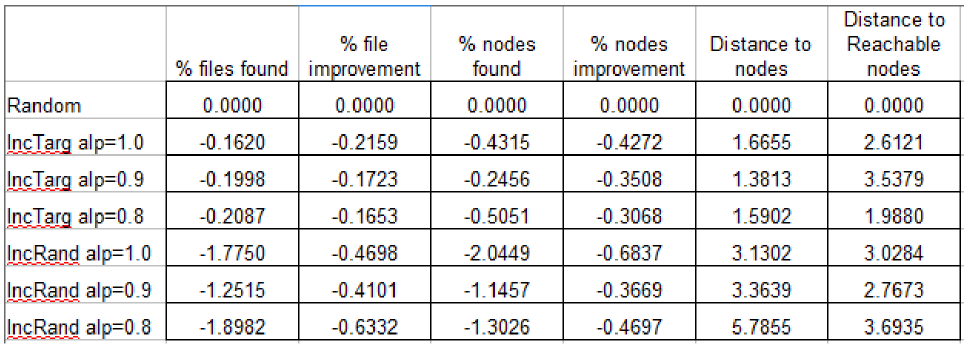


FIGURE 14. Average distance to reachable contraband nodes using 100 probes in v0.4 network

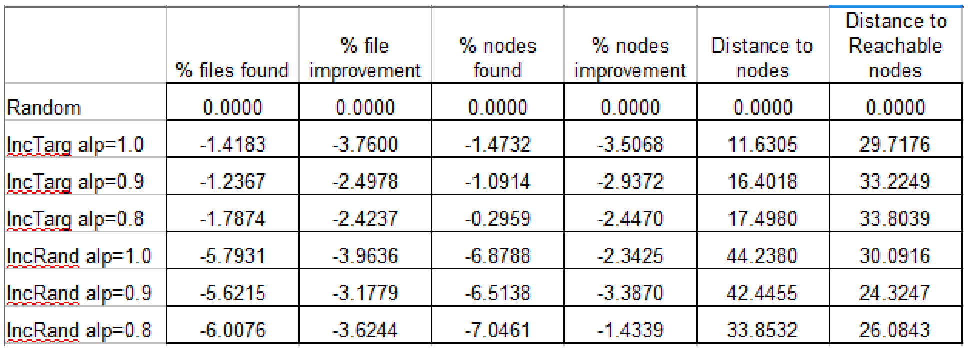
4.3.4. Statistical Analysis. A t-test was employed to test the null hypothesis that the algorithms do not improve node positioning appreciably better than a random search. First that test requires the calculation of the t-statistic in 4.3.1 [**20**] where $\overline{X}_i$ is the i^{th} sample mean, s_i is the i^{th} sample standard deviation, and N_i is the i^{th} sample size. The two samples are the results generated by the base algorithm and tested algorithm.

$$t = (\overline{X}_1 - \overline{X}_2)/\sqrt{s_1^2/N_1 + s_2^2/N_2} \tag{4.3.1}$$



	% files found	% file improvement	% nodes found	% nodes improvement	Distance to nodes	Distance to Reachable nodes
Random	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
IncTarg alp=1.0	-0.1620	-0.2159	-0.4315	-0.4272	1.6655	2.6121
IncTarg alp=0.9	-0.1998	-0.1723	-0.2456	-0.3508	1.3813	3.5379
IncTarg alp=0.8	-0.2087	-0.1653	-0.5051	-0.3068	1.5902	1.9880
IncRand alp=1.0	-1.7750	-0.4698	-2.0449	-0.6837	3.1302	3.0284
IncRand alp=0.9	-1.2515	-0.4101	-1.1457	-0.3669	3.3639	2.7673
IncRand alp=0.8	-1.8982	-0.6332	-1.3026	-0.4697	5.7855	3.6935

TABLE 3. t-statistic for Gnutella v0.4 and 50 probes

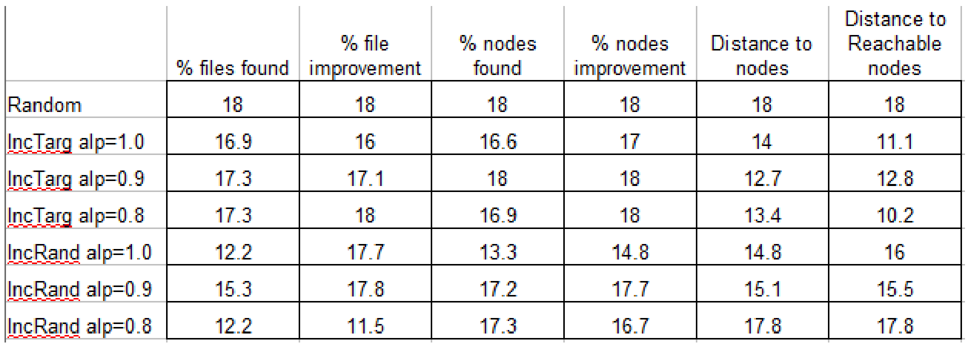


	% files found	% file improvement	% nodes found	% nodes improvement	Distance to nodes	Distance to Reachable nodes
Random	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
IncTarg alp=1.0	-1.4183	-3.7600	-1.4732	-3.5068	11.6305	29.7176
IncTarg alp=0.9	-1.2367	-2.4978	-1.0914	-2.9372	16.4018	33.2249
IncTarg alp=0.8	-1.7874	-2.4237	-0.2959	-2.4470	17.4980	33.8039
IncRand alp=1.0	-5.7931	-3.9636	-6.8788	-2.3425	44.2380	30.0916
IncRand alp=0.9	-5.6215	-3.1779	-6.5138	-3.3870	42.4455	24.3247
IncRand alp=0.8	-6.0076	-3.6244	-7.0461	-1.4339	33.8532	26.0843

TABLE 4. t-statistic for Gnutella v0.4 and 100 probes

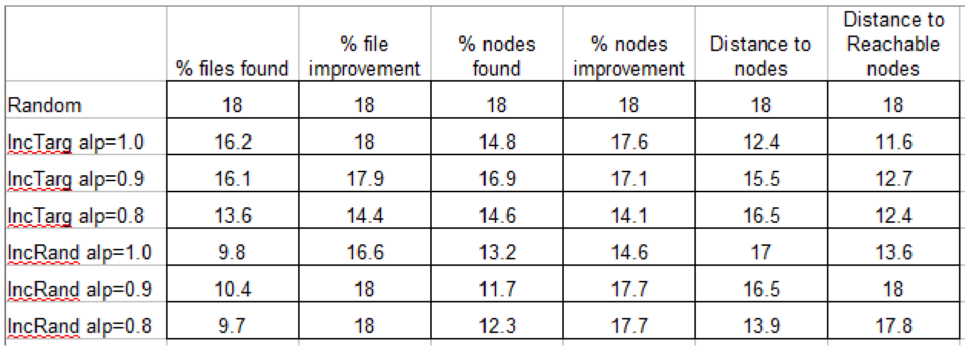
In addition, the test requires that the degrees of freedom for each sample must be calculated using Equation 4.3.2. **Tables** 5 and **6** show the results of this calculation for each set.

$$v = (s_1^2/N_1 + s_2^2/N_2)^2/((s_1^4/(N_1^2 - (N_1 - 1))) + (s_2^4/(N_2^2 - (N_2 - 1))) \tag{4.3.2}$$



	% files found	% file improvement	% nodes found	% nodes improvement	Distance to nodes	Distance to Reachable nodes
Random	18	18	18	18	18	18
IncTarg alp=1.0	16.9	16	16.6	17	14	11.1
IncTarg alp=0.9	17.3	17.1	18	18	12.7	12.8
IncTarg alp=0.8	17.3	18	16.9	18	13.4	10.2
IncRand alp=1.0	12.2	17.7	13.3	14.8	14.8	16
IncRand alp=0.9	15.3	17.8	17.2	17.7	15.1	15.5
IncRand alp=0.8	12.2	11.5	17.3	16.7	17.8	17.8

TABLE 5. Degrees of freedom for Gnutella v0.4 and 50 probes



	% files found	% file improvement	% nodes found	% nodes improvement	Distance to nodes	Distance to Reachable nodes
Random	18	18	18	18	18	18
IncTarg alp=1.0	16.2	18	14.8	17.6	12.4	11.6
IncTarg alp=0.9	16.1	17.9	16.9	17.1	15.5	12.7
IncTarg alp=0.8	13.6	14.4	14.6	14.1	16.5	12.4
IncRand alp=1.0	9.8	16.6	13.2	14.6	17	13.6
IncRand alp=0.9	10.4	18	11.7	17.7	16.5	18
IncRand alp=0.8	9.7	18	12.3	17.7	13.9	17.8

TABLE 6. Degrees of freedom for Gnutella v0.4 and 100 probes

Once the t-statistic and degrees of freedom for each sample have been determined, a t-distribution is used to find the one-tailed p-value. A low p-value indicates that there is a low probability that the null hypothesis is true and thus that the algorithms do in fact improve node position for obtaining query results. The threshold for rejecting the null hypothesis was set at 0.05 which is a typical value when evaluating surveys with a low sample population. In **Tables** 7 and **8**, the green highlight indicates those results that can confidently be said to be significantly better than random search (i.e. p-value < 0.05).

	% files found	% file improvement	% nodes found	% nodes improvement	Distance to nodes	Distance to Reachable nodes
Random	0.500	0.500	0.500	0.500	0.500	0.500
IncTarg alp=1.0	0.168	0.118	0.010	0.009	0.000	0.000
IncTarg alp=0.9	0.112	0.149	0.052	0.012	0.000	0.000
IncTarg alp=0.8	0.102	0.127	0.004	0.021	0.000	0.000
IncRand alp=1.0	0.000	0.003	0.000	0.002	0.000	0.000
IncRand alp=0.9	0.000	0.007	0.000	0.014	0.000	0.000
IncRand alp=0.8	0.000	0.024	0.000	0.006	0.000	0.000

TABLE 7. p-value of the t-test for Gnutella v0.4 and 50 probes. Highlighted values indicated results that are significantly better than random searching for that technique and metric.

	% files found	% file improvement	% nodes found	% nodes improvement	Distance to nodes	Distance to Reachable nodes
Random	0.500	0.500	0.500	0.500	0.500	0.500
IncTarg alp=1.0	0.088	0.001	0.081	0.001	0.000	0.000
IncTarg alp=0.9	0.117	0.012	0.146	0.005	0.000	0.000
IncTarg alp=0.8	0.049	0.015	0.386	0.014	0.000	0.000
IncRand alp=1.0	0.000	0.001	0.000	0.017	0.000	0.000
IncRand alp=0.9	0.000	0.003	0.000	0.002	0.000	0.000
IncRand alp=0.8	0.000	0.001	0.000	0.085	0.000	0.000

TABLE 8. p-value of the t-test for Gnutella v0.4 and 100 probes. Highlighted values indicated results that are significantly better than random searching for that technique and metric.

4.4. Gnutella v0.6 Topology

The objective with these results was to check that the results on the Gnutella v0.4 network were still valid when applied to a more structured and modern network, where some nodes are more highly connected than others. Thus the difference in the network can be seen in the edge degree frequency for nodes in the network.

4.4.1. Contraband Content Discovered. The same file distribution and queries used in the Gnutella v0.4 network were used again on the new topology. The overall contraband nodes and content found on the network were reduced for each approach to querying. Random searching with 50 probes dropped from an average of 47.0% contraband files found in the v0.4 network to 35.7% contraband content found in the v0.6 network. The structured

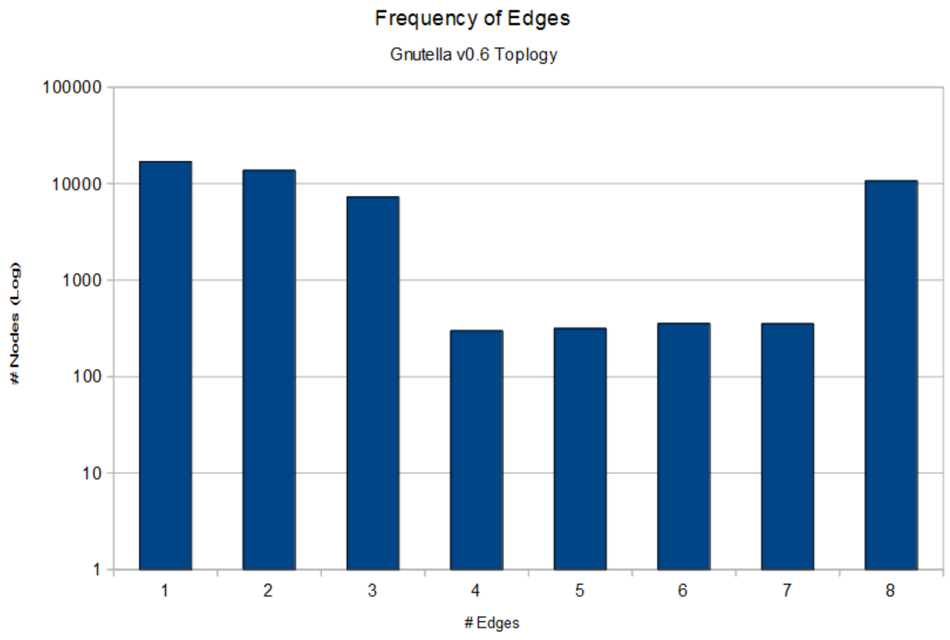


FIGURE 15. Edge number distribution for simulated Gnutella v0.6 network

networks in v0.6 were supposed to make the network more scalable and increase the availability of popular content. It could be inferred from these results that content falling below a certain threshold may have become harder to find in the structured network. However there were no surveys found that support that conclusion regarding the move to v0.6 of the protocol. The reduced amount of contraband found when doing random search gives greater opportunity for the algorithms to improve the results of queries.

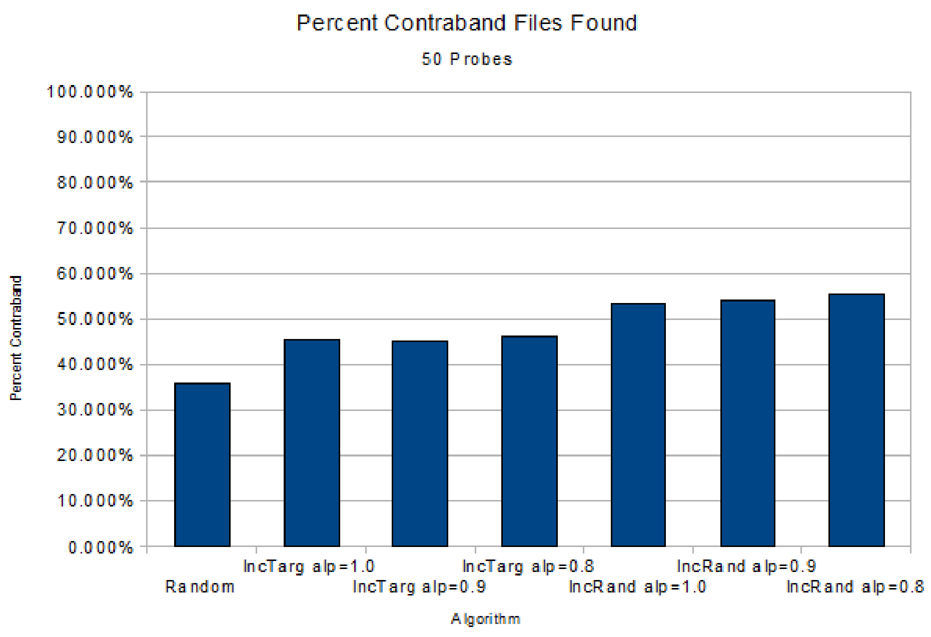


FIGURE 16. Percent Contraband Files discovered using 50 probes in a v0.6 network

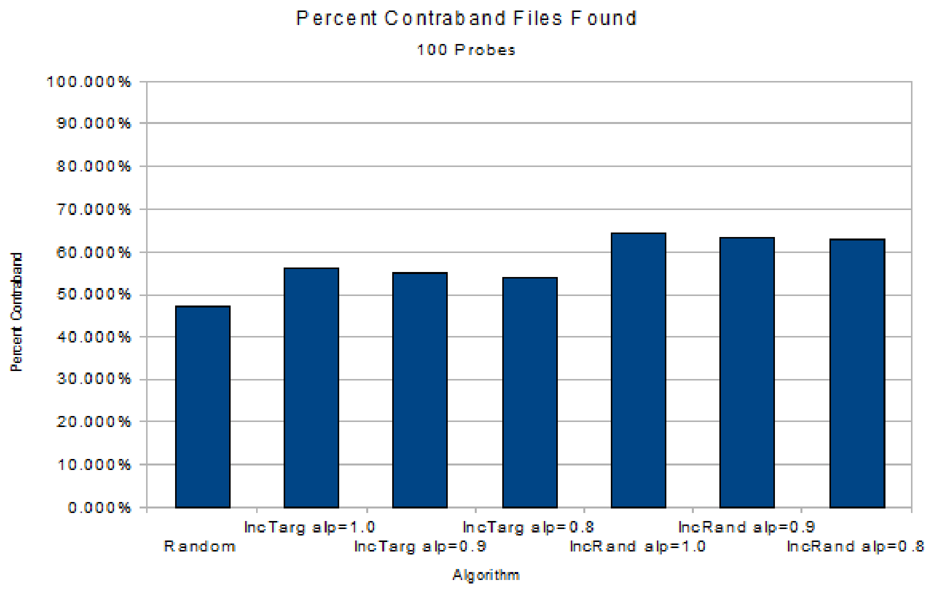


FIGURE 17. Percent Contraband Files discovered using 100 probes in a v0.6 network

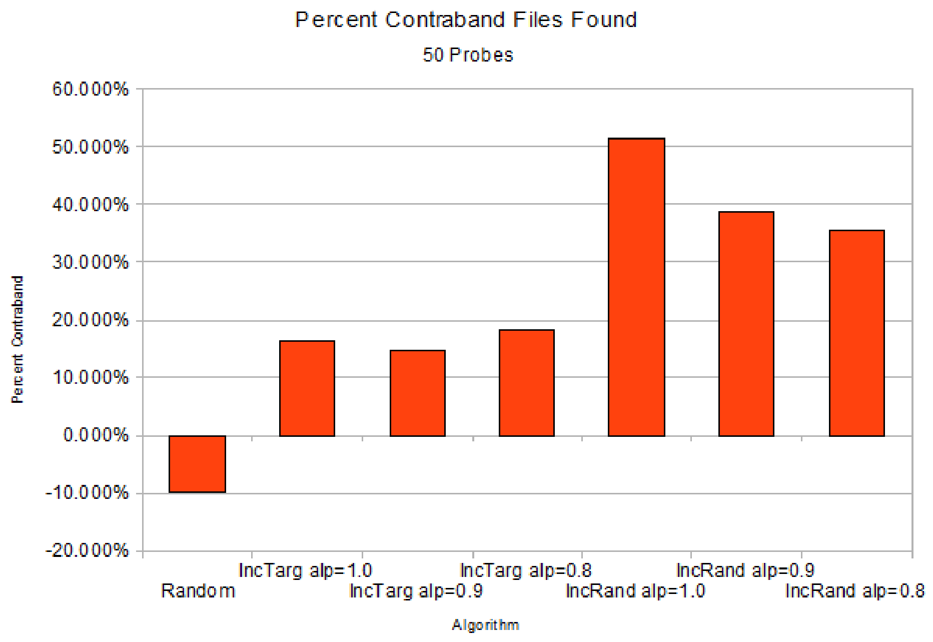


FIGURE 18. Percent Improvement in Contraband Files queriable using 50 probes in a v0.6 network

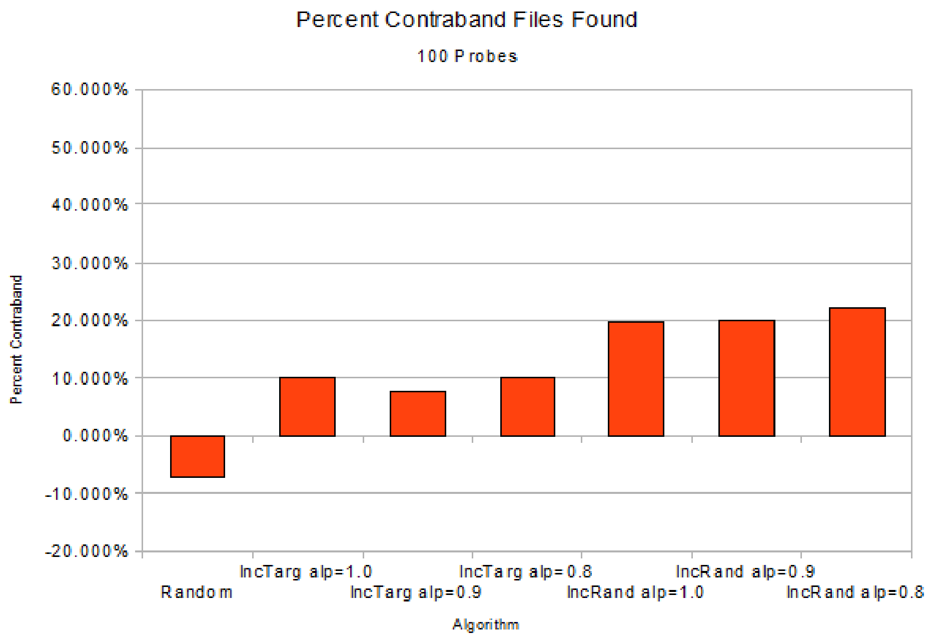


FIGURE 19. Percent Improvement in Contraband Files queriable using 100 probes in v0.6 network

4.4.2. Contraband Nodes Discovered. For contraband nodes, there was a corresponding drop in percentage of nodes found in the v0.6 network versus the v0.4 network. When searching randomly with 50 probes, the percent of contraband nodes discovered dropped to 48.0%, down from 64.2%. The number of probes in the v0.6 network therefore had a much greater impact on the number of contraband nodes found compared to the v0.4 network. There was approximately a 38% increase in the number of nodes found randomly searching using 100 probes versus 50 in the v0.6 network compared to only 26% increase in the v0.4 network.

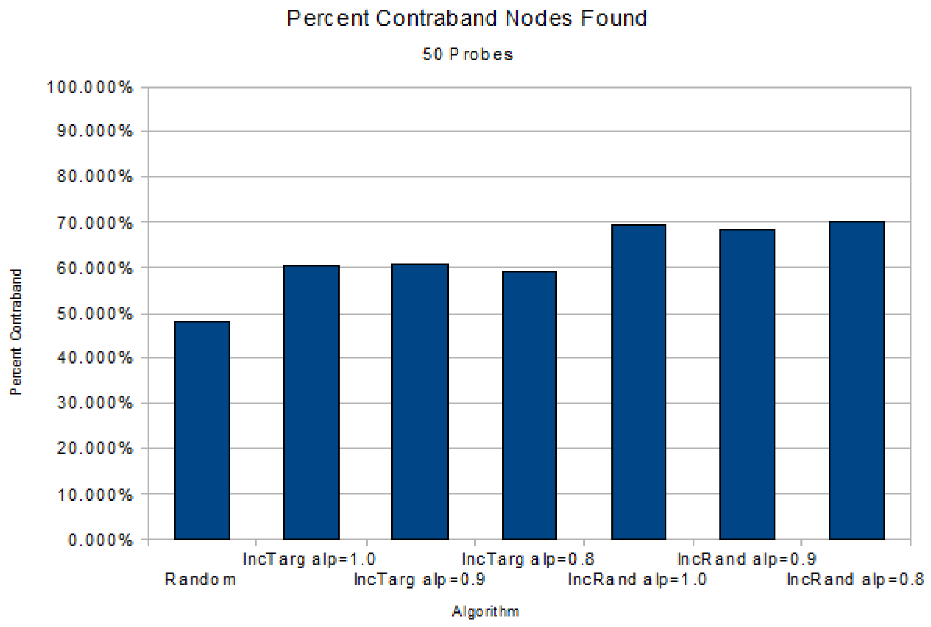


FIGURE 20. Percent Contraband Nodes discovered using 50 probes in v0.6 network

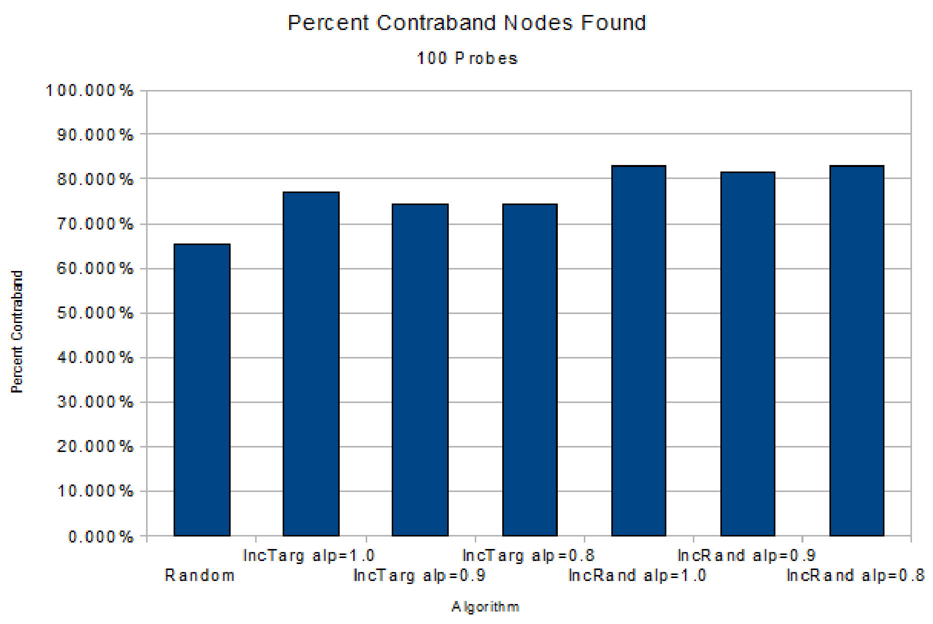


FIGURE 21. Percent Contraband Nodes discovered using 100 probes in v0.6 network

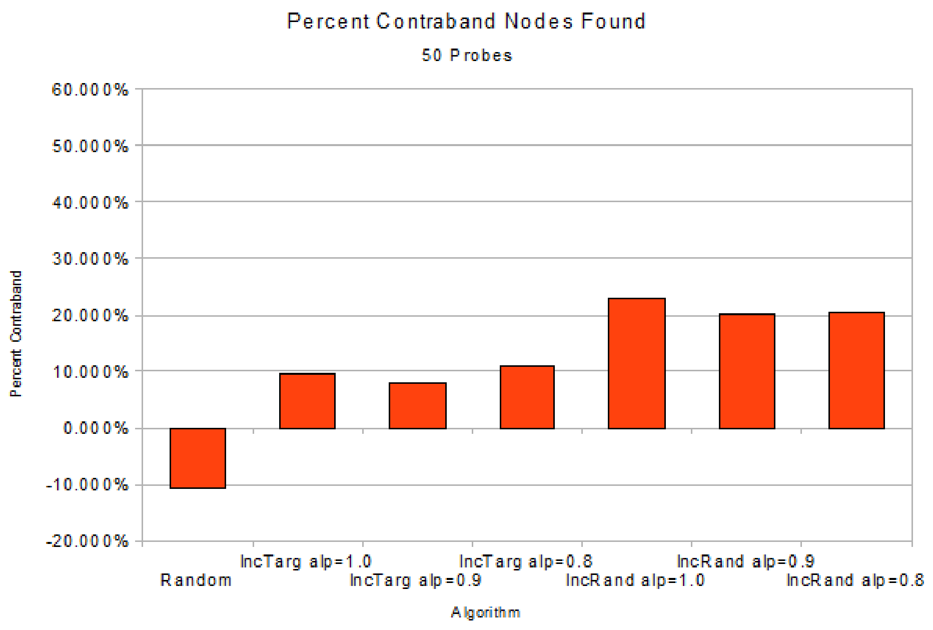


FIGURE 22. Percent Improvement in Contraband Nodes queriable using 50 probes in v0.6 network

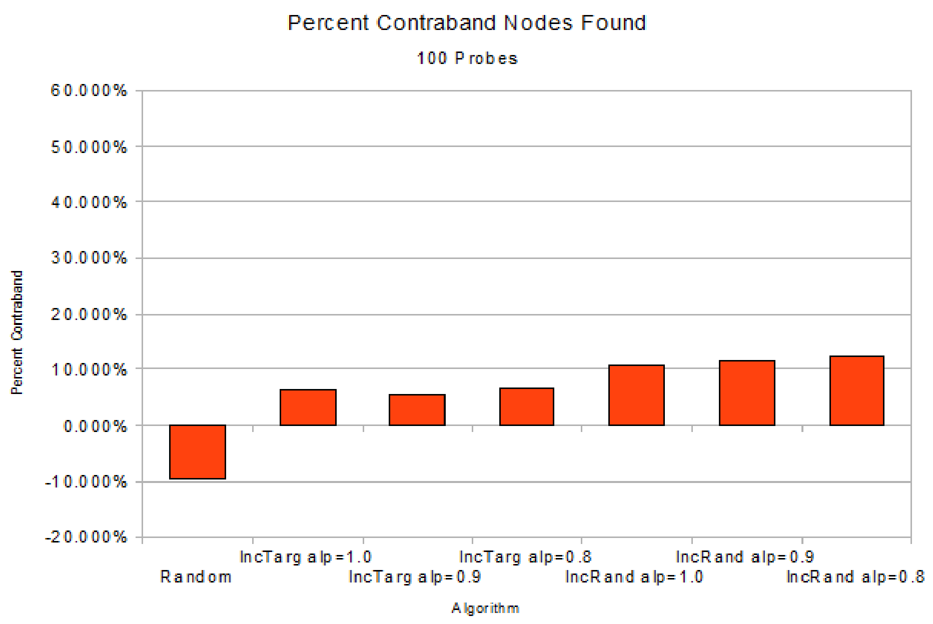


FIGURE 23. Percent Improvement in Contraband Nodes queriable using 100 probes in v0.6 network

4.4.3. Distance to Contraband. As shown in **Figures** 24 and **25** distance to contraband in the v0.6 network also increased, which can be attributed to most nodes having to work through a smaller number of super nodes in order to get access to the entire network. Even after improving overall position, the average distance to contraband nodes did not drop significantly compared to random searching.

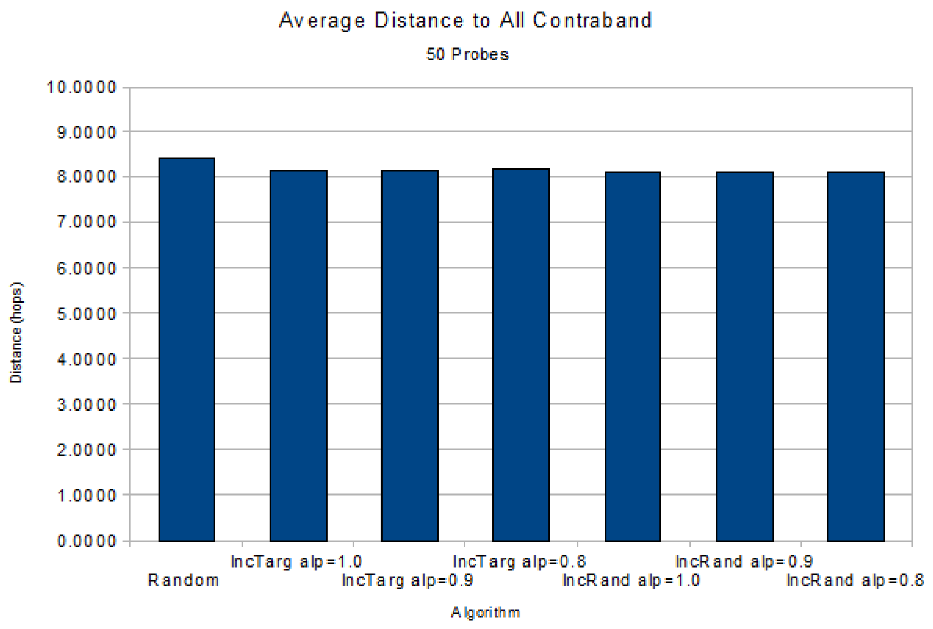


FIGURE 24. Average distance to all contraband nodes using 50 probes in v0.6 network

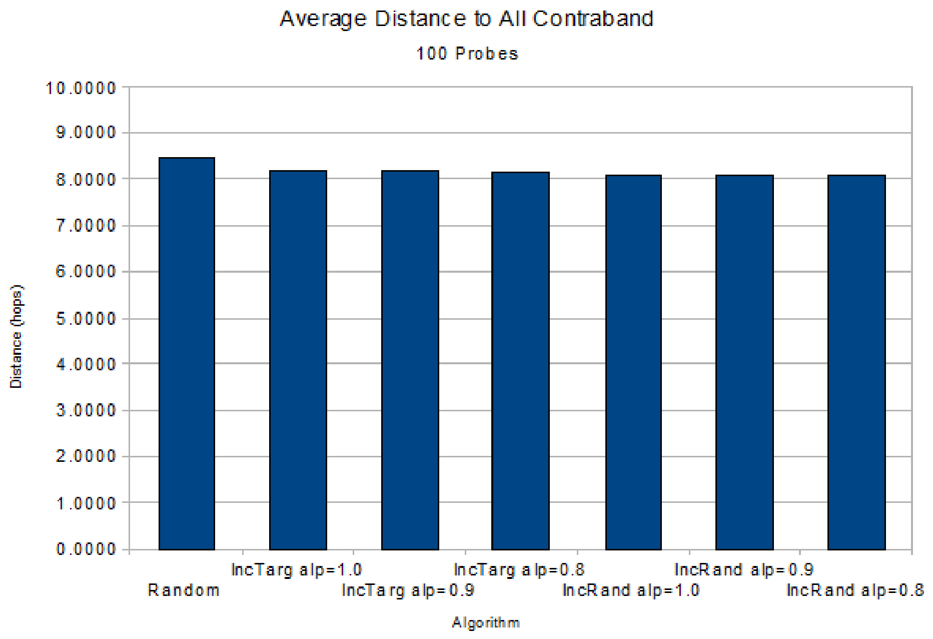


FIGURE 25. Average distance to all contraband nodes using 100 probes in v0.6 network

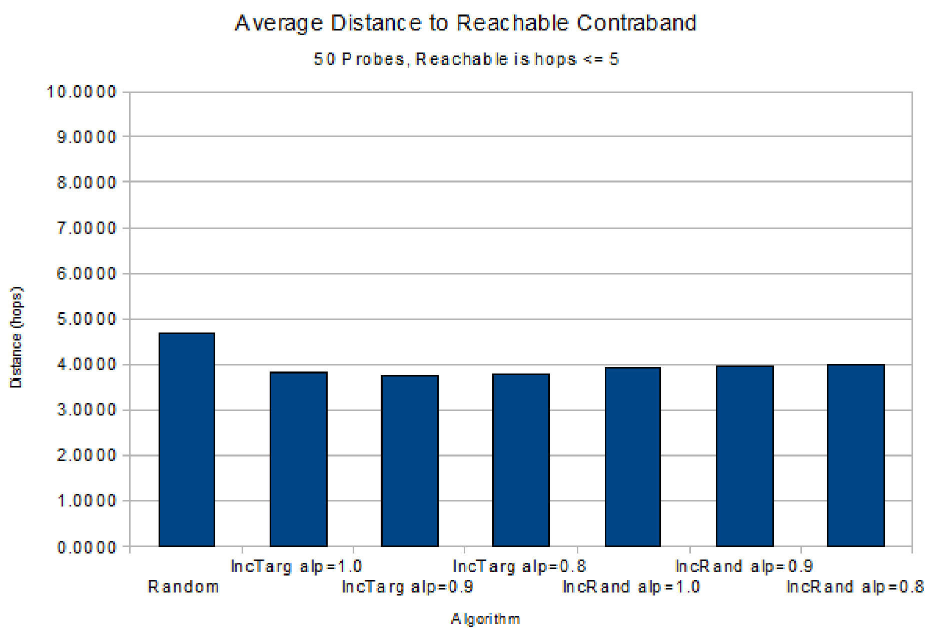


FIGURE 26. Average distance to reachable contraband nodes using 50 probes in v0.6 network

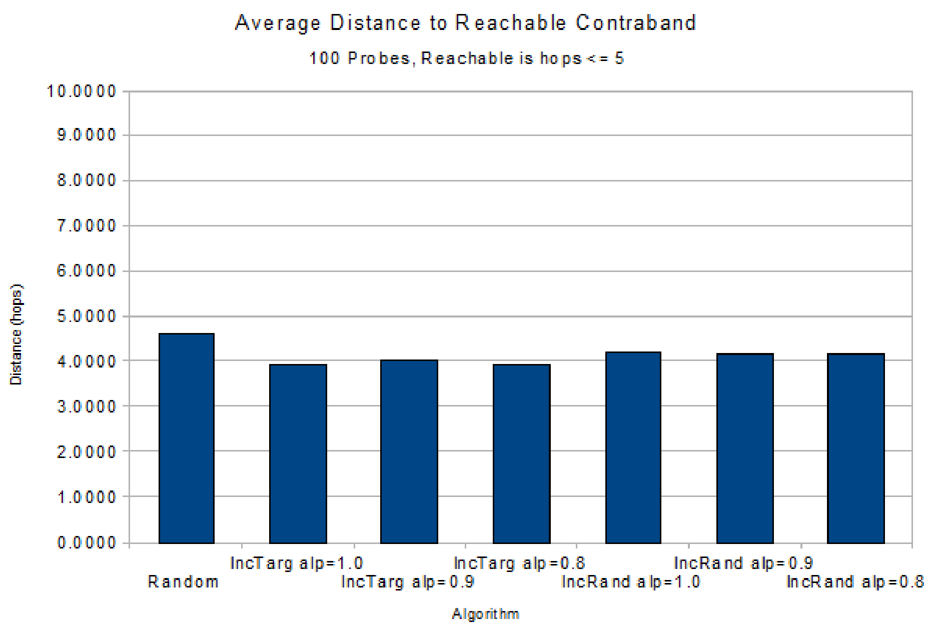
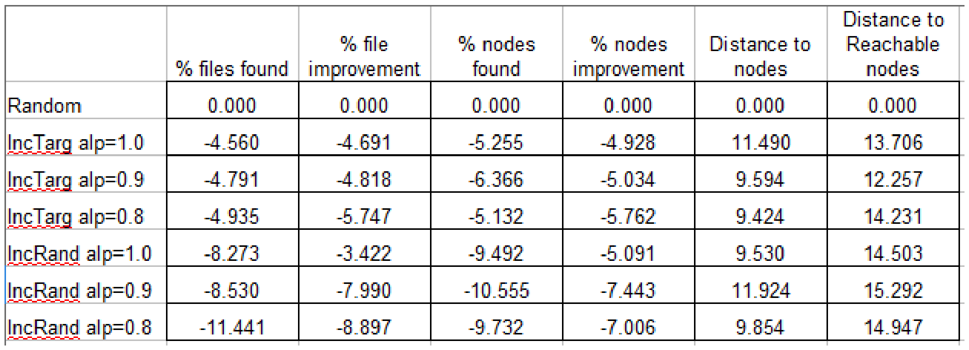


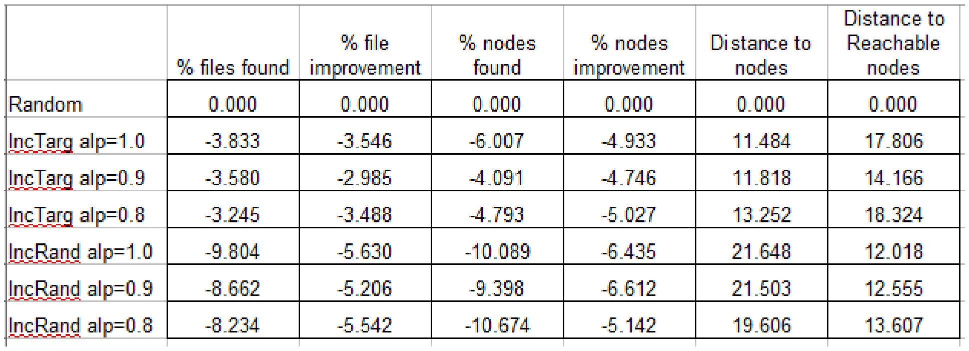
FIGURE 27. Average distance to reachable contraband nodes using 100 probes in v0.6 network

4.4.4. Statistical Analysis. The results of the t-test for the v0.6 network as was done for v0.4 network is shown in **Tables** 4.4.4-14. Again the green highlighted p-values are those that validate the significance of the improvements and disprove the null hypothesis (p-value < 0.05).



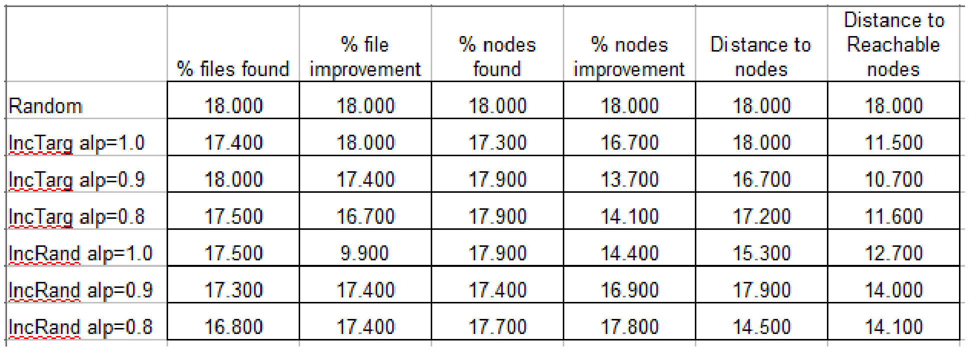
	% files found	% file improvement	% nodes found	% nodes improvement	Distance to nodes	Distance to Reachable nodes
Random	0.000	0.000	0.000	0.000	0.000	0.000
IncTarg alp=1.0	-4.560	-4.691	-5.255	-4.928	11.490	13.706
IncTarg alp=0.9	-4.791	-4.818	-6.366	-5.034	9.594	12.257
IncTarg alp=0.8	-4.935	-5.747	-5.132	-5.762	9.424	14.231
IncRand alp=1.0	-8.273	-3.422	-9.492	-5.091	9.530	14.503
IncRand alp=0.9	-8.530	-7.990	-10.555	-7.443	11.924	15.292
IncRand alp=0.8	-11.441	-8.897	-9.732	-7.006	9.854	14.947

TABLE 9. t-statistic for Gnutella v0.6 and 50 probes



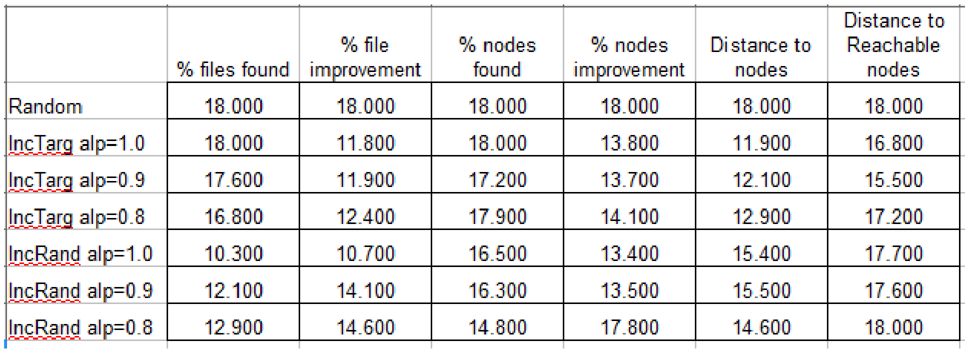
	% files found	% file improvement	% nodes found	% nodes improvement	Distance to nodes	Distance to Reachable nodes
Random	0.000	0.000	0.000	0.000	0.000	0.000
IncTarg alp=1.0	-3.833	-3.546	-6.007	-4.933	11.484	17.806
IncTarg alp=0.9	-3.580	-2.985	-4.091	-4.746	11.818	14.166
IncTarg alp=0.8	-3.245	-3.488	-4.793	-5.027	13.252	18.324
IncRand alp=1.0	-9.804	-5.630	-10.089	-6.435	21.648	12.018
IncRand alp=0.9	-8.662	-5.206	-9.398	-6.612	21.503	12.555
IncRand alp=0.8	-8.234	-5.542	-10.674	-5.142	19.606	13.607

TABLE 10. t-statistic for Gnutella v0.6 and 100 probes



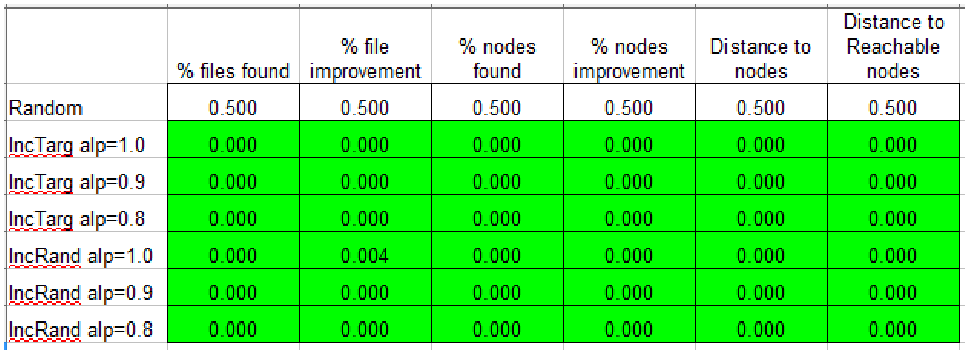
	% files found	% file improvement	% nodes found	% nodes improvement	Distance to nodes	Distance to Reachable nodes
Random	18.000	18.000	18.000	18.000	18.000	18.000
IncTarg alp=1.0	17.400	18.000	17.300	16.700	18.000	11.500
IncTarg alp=0.9	18.000	17.400	17.900	13.700	16.700	10.700
IncTarg alp=0.8	17.500	16.700	17.900	14.100	17.200	11.600
IncRand alp=1.0	17.500	9.900	17.900	14.400	15.300	12.700
IncRand alp=0.9	17.300	17.400	17.400	16.900	17.900	14.000
IncRand alp=0.8	16.800	17.400	17.700	17.800	14.500	14.100

TABLE 11. Degrees of freedom for Gnutella v0.6 and 50 probes



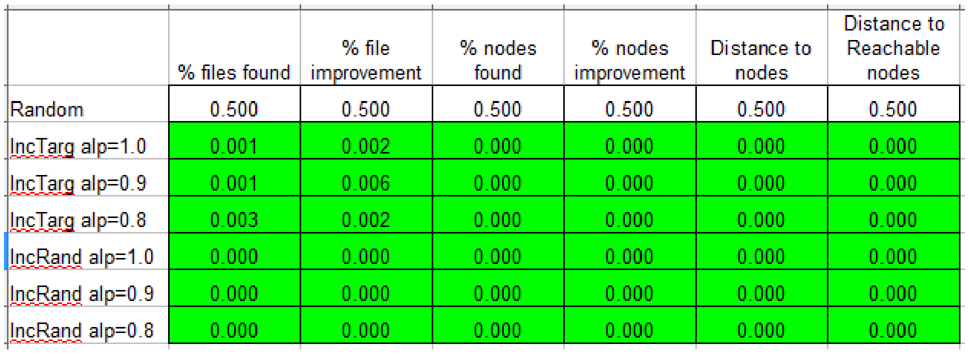
	% files found	% file improvement	% nodes found	% nodes improvement	Distance to nodes	Distance to Reachable nodes
Random	18.000	18.000	18.000	18.000	18.000	18.000
IncTarg alp=1.0	18.000	11.800	18.000	13.800	11.900	16.800
IncTarg alp=0.9	17.600	11.900	17.200	13.700	12.100	15.500
IncTarg alp=0.8	16.800	12.400	17.900	14.100	12.900	17.200
IncRand alp=1.0	10.300	10.700	16.500	13.400	15.400	17.700
IncRand alp=0.9	12.100	14.100	16.300	13.500	15.500	17.600
IncRand alp=0.8	12.900	14.600	14.800	17.800	14.600	18.000

TABLE 12. Degrees of freedom for Gnutella v0.6 and 100 probes



	% files found	% file improvement	% nodes found	% nodes improvement	Distance to nodes	Distance to Reachable nodes
Random	0.500	0.500	0.500	0.500	0.500	0.500
IncTarg alp=1.0	0.000	0.000	0.000	0.000	0.000	0.000
IncTarg alp=0.9	0.000	0.000	0.000	0.000	0.000	0.000
IncTarg alp=0.8	0.000	0.000	0.000	0.000	0.000	0.000
IncRand alp=1.0	0.000	0.004	0.000	0.000	0.000	0.000
IncRand alp=0.9	0.000	0.000	0.000	0.000	0.000	0.000
IncRand alp=0.8	0.000	0.000	0.000	0.000	0.000	0.000

TABLE 13. p-value for t-test for Gnutella v0.6 and 50 probes. Highlighted values indicated results that are significantly better than random searching for that technique and metric.



	% files found	% file improvement	% nodes found	% nodes improvement	Distance to nodes	Distance to Reachable nodes
Random	0.500	0.500	0.500	0.500	0.500	0.500
IncTarg alp=1.0	0.001	0.002	0.000	0.000	0.000	0.000
IncTarg alp=0.9	0.001	0.006	0.000	0.000	0.000	0.000
IncTarg alp=0.8	0.003	0.002	0.000	0.000	0.000	0.000
IncRand alp=1.0	0.000	0.000	0.000	0.000	0.000	0.000
IncRand alp=0.9	0.000	0.000	0.000	0.000	0.000	0.000
IncRand alp=0.8	0.000	0.000	0.000	0.000	0.000	0.000

TABLE 14. p-value for t-test for Gnutella v0.6 and 100 probes. Highlighted values indicated results that are significantly better than random searching for that technique and metric.

CHAPTER 5

Conclusions

In this Chapter, the results will be distilled into a succinct conclusion based on the analysis of the experiment detailed in Chapter 4. Also some of the limits of the analysis and techniques evaluated will be reviewed with some recommendations of how one might want to proceed if these initial results proved promising.

5.1. Summary

The results showed that the v0.6 network is ripe for techniques that would improve query results by modifying connections based on user queries. Both **Alogrithm 5** and **6** gave across-the-board improvements in query results for each quality metric compared to random searching. These techniques deployed to a sufficient percentage of overall network nodes could be used by law enforcement to significantly increase their ability to identify users sharing illicit content. The α values tested appeared to have little influence on the results given the universal improvement compared to random searching.

The v0.4 network however appears to require more exploratory approaches to improving query results as the targeted exploration approach did not appear to significantly improve query results returned or nodes found compared to random searching. However, the increased random exploration improved results for all input parameters evaluated. There seems to be some advantage for the targeted search technique using an α value of 1.0 and 0.8 as these found additional results more reliably when 0.1% of the network consisted of probes. However in the v0.4 network when 0.2% of the network was set up as probes that search for contraband, there was no advantage for the targeted technique at any tested value of α.

5.2. Limitations

One of the greatest limitations of the analysis presented in this thesis happens to be one of the strongest advantages of peer-to-peer networks. In particular, the highly dynamic nature of the network makes it difficult to know if the simulated network was representative of a real world scenario. From the cited surveys [**7, 8**] one can see just how varied the Gnutella network has been over a few days to a few years. Another limiting factor is how accurate the estimate used for the prevalence of contraband files on the network. If either of these assumptions proved significantly inaccurate then much of the results could be discounted.

5.3. Recommendations

There are numerous areas where additional techniques could be applied to improve results. Some heuristics applied to the Ramanathan update equations, or used to control network exploration could improve results or decrease the resources needed to explore the network. In addition scaling up the network simulation or implementing a real client would be logical next steps in evaluating these techniques.

When implementing the Ramanathan formulas, it became obvious that a simulated annealing technique applied to the α and β values could be used to let a probe explore more broadly at first and then settle into a more stable neighborhood after a certain time. A similar method could also be applied to the constant 50% chance to increase exploration and replace nodes that have currently returned zero relevant results. This again would allow for increased exploration at appropriate stages of the search and allow for a stable set of connections after certain criteria were met.

Also, there is some data [**7, 8**] on the Gnutella network that would help to validate whether the networks tested would indeed be representative of real networks. There was no data found that quantified the number of nodes in the network that left connections open for new nodes in the network. It could be that in the v0.6 network only super nodes keep open connections and that leaf nodes regularly fill all connections at start-up. This would tend to diminish the ability to use a random exploration technique as only super nodes would ever accept new connections, potentially leaving some parts of the network unexplorable. In fact, while testing a small client in the Limewire network, only other super nodes tended to ever accept new connection requests. However this experience would need to be validated by a survey of the network.

APPENDIX 1

```
DROP VIEW IF EXISTS vreplies;
DROP VIEW IF EXISTS vquery_sources;
DROP VIEW IF EXISTS vnode_reachability;
DROP VIEW IF EXISTS vnode_importance;
DROP VIEW IF EXISTS vnode_connections;
DROP VIEW IF EXISTS vqueries_sent;
DROP VIEW IF EXISTS vqueries_received;
DROP TABLE IF EXISTS tresult_sources;
DROP TABLE IF EXISTS tresults;
DROP TABLE IF EXISTS treply_locations;
DROP TABLE IF EXISTS treplies;
DROP TABLE IF EXISTS tquery_sources;
DROP TABLE IF EXISTS tnode_reachability;
DROP TABLE IF EXISTS tnode_importance;
DROP TABLE IF EXISTS tnode_connections;
DROP TABLE IF EXISTS tnode_proxies;
DROP TABLE IF EXISTS tnodes;
DROP TABLE IF EXISTS tqueries;
DROP TABLE IF EXISTS texperiments;
CREATE TABLE texperiments
(
    exp_id       INTEGER(8) NOT NULL AUTO_INCREMENT
   ,description  VARCHAR(512)
   ,created      TIMESTAMP DEFAULT CURRENT_TIMESTAMP
   ,CONSTRAINT pk_exp_id PRIMARY KEY ( exp_id )
);
CREATE TABLE tqueries
(
    query_id      INTEGER(16) AUTO_INCREMENT
   ,exp_id        INTEGER(8) NOT NULL
   ,query_guid    VARCHAR(128) NOT NULL
   ,query         VARCHAR(1024) NOT NULL
   ,incoming      INTEGER(1) NOT NULL DEFAULT 0
   ,created       TIMESTAMP DEFAULT CURRENT_TIMESTAMP
   ,CONSTRAINT pk_query_id PRIMARY KEY ( query_id )
   ,CONSTRAINT queries_fk_exp FOREIGN KEY ( exp_id ) REFERENCES texperiments ( exp_id )
   ,CONSTRAINT incoming_check CHECK ( incoming = 0 OR incoming = 1 )
);
CREATE TABLE tnodes
(
    node_id       INTEGER(16) AUTO_INCREMENT
   ,exp_id        INTEGER(8) NOT NULL
   ,ip_address    VARCHAR(32) NOT NULL
   ,port          INTEGER(8) NOT NULL
   ,created       TIMESTAMP DEFAULT CURRENT_TIMESTAMP
   ,CONSTRAINT pk_node_id PRIMARY KEY ( node_id )
   ,CONSTRAINT nodes_fk_exp FOREIGN KEY ( exp_id ) REFERENCES texperiments ( exp_id )
   ,CONSTRAINT uniq_exp_ip_port UNIQUE ( exp_id , ip_address , port )
);
CREATE TABLE tnode_proxies
(
    node_id       INTEGER(16) NOT NULL
   ,prx_node_id   INTEGER(16) NOT NULL
   ,created       TIMESTAMP DEFAULT CURRENT_TIMESTAMP
   ,CONSTRAINT node_proxies_fk_nodes1 FOREIGN KEY ( node_id ) REFERENCES tnodes ( node_id )
   ,CONSTRAINT node_proxies_fk_nodes2 FOREIGN KEY ( prx_node_id ) REFERENCES tnodes ( node_id )
   ,CONSTRAINT uniq_id_ip_port UNIQUE ( node_id , prx_node_id )
);
CREATE TABLE tnode_connections
(
    node_id1      INTEGER(16) NOT NULL
   ,node_id2      INTEGER(16) NOT NULL
   ,CONSTRAINT node_conns_fk_nodes1 FOREIGN KEY ( node_id1 ) REFERENCES tnodes ( node_id )
   ,CONSTRAINT node_conns_fk_nodes2 FOREIGN KEY ( node_id2 ) REFERENCES tnodes ( node_id )
   ,CONSTRAINT uniq_conn_id1_id2 UNIQUE ( node_id1 , node_id2 )
);
CREATE TABLE tnode_importance
(
    node_id1     INTEGER(16) NOT NULL
   ,node_id2     INTEGER(16) NOT NULL
   ,query_id     INTEGER(16) NOT NULL
   ,importance   FLOAT(16,14) NOT NULL
   ,aver_n_hops  FLOAT(6,4) NOT NULL
   ,perc_hits    FLOAT(8,6) NOT NULL
   ,total_hits   INTEGER(6) NOT NULL
   ,CONSTRAINT node_imp_fk_nodes1 FOREIGN KEY ( node_id1 ) REFERENCES tnodes ( node_id )
   ,CONSTRAINT node_imp_fk_nodes2 FOREIGN KEY ( node_id2 ) REFERENCES tnodes ( node_id )
   ,CONSTRAINT node_imp_fk_queries FOREIGN KEY ( query_id ) REFERENCES tqueries ( query_id )
   ,CONSTRAINT uniq_imp_node_id1_2_query_id UNIQUE ( node_id1 , node_id2 , query_id )
);
CREATE TABLE tnode_reachability
(
    node_id1      INTEGER(16) NOT NULL
   ,node_id2      INTEGER(16) NOT NULL
   ,distance      INTEGER(3) NOT NULL
   ,CONSTRAINT nodes_of_interest_fk_nodes1 FOREIGN KEY ( node_id1 ) REFERENCES tnodes ( node_id
   ,CONSTRAINT nodes_of_interest_fk_nodes2 FOREIGN KEY ( node_id2 ) REFERENCES tnodes ( node_id
);
CREATE TABLE tquery_sources
(
    query_id     INTEGER(16) NOT NULL
   ,node_id      INTEGER(16) NOT NULL
   ,CREATED      TIMESTAMP DEFAULT CURRENT_TIMESTAMP
   ,CONSTRAINT query_sources_fk_queries FOREIGN KEY ( query_id ) REFERENCES tqueries ( query_id
   ,CONSTRAINT query_sources_fk_nodes FOREIGN KEY ( node_id ) REFERENCES tnodes ( node_id )
);
```

```
CREATE TABLE treplies (
    reply_id      INTEGER(24) AUTO_INCREMENT
   ,query_id      INTEGER(16) NOT NULL
   ,node_id_to    INTEGER(16) NOT NULL
   ,node_id_frm   INTEGER(16) NOT NULL
   ,node_id_thru  INTEGER(16) NOT NULL
   ,reply_guid    VARCHAR(128) NOT NULL
   ,is_push       INTEGER(1) NOT NULL
   ,vendor        VARCHAR(64)
   ,hops          INTEGER(2)
   ,ttl           INTEGER(2)
   ,file_name     VARCHAR(4096)
   ,created       TIMESTAMP DEFAULT CURRENT_TIMESTAMP
   ,CONSTRAINT pk_reply_id PRIMARY KEY ( reply_id )
   ,CONSTRAINT replies_fk_queries FOREIGN KEY ( query_id ) REFERENCES tqueries ( query_id )
   ,CONSTRAINT replies_to_fk_nodes FOREIGN KEY ( node_id_to ) REFERENCES tnodes ( node_id )
   ,CONSTRAINT replies_frm_fk_nodes FOREIGN KEY ( node_id_frm ) REFERENCES tnodes ( node_id )
   ,CONSTRAINT replies_thru_fk_nodes FOREIGN KEY ( node_id_thru) REFERENCES tnodes ( node_id )
   ,CONSTRAINT is_push_check CHECK ( is_push IN (-1, 0, 1) )
   ,CONSTRAINT hops_check CHECK ( hops > 0 )
   ,CONSTRAINT ttl_check CHECK ( ttl > 0 )
);
CREATE TABLE treply_locations
(
    reply_id     INTEGER(24) NOT NULL
   ,node_id      INTEGER(16) NOT NULL
   ,created      TIMESTAMP DEFAULT CURRENT_TIMESTAMP
   ,CONSTRAINT reply_locations_fk_replies FOREIGN KEY ( reply_id ) REFERENCES treplies ( reply_
   ,CONSTRAINT reply_locations_fk_nodes FOREIGN KEY ( node_id ) REFERENCES tnodes ( node_id )
   ,CONSTRAINT uniq_id_ip_port UNIQUE ( reply_id , node_id )
);
CREATE TABLE tresults
(
    result_id    INTEGER(20) AUTO_INCREMENT
   ,query_id      INTEGER(16) NOT NULL
   ,file_name     VARCHAR(4096)
   ,file_data     LONGBLOB
   ,created       TIMESTAMP DEFAULT CURRENT_TIMESTAMP
   ,CONSTRAINT pk_result_id PRIMARY KEY ( result_id )
   ,CONSTRAINT result_fk_queries FOREIGN KEY ( query_id ) REFERENCES tqueries ( query_id )
) ENGINE=InnoDB;
CREATE TABLE tresult_sources
(
    result_id     INTEGER(20) NOT NULL
   ,node_id       INTEGER(16) NOT NULL
   ,CONSTRAINT result_sources_fk_results FOREIGN KEY ( result_id ) REFERENCES tresults ( result
   ,CONSTRAINT result_sources_fk_nodes FOREIGN KEY ( node_id ) REFERENCES tnodes ( node_id )
);
CREATE VIEW vqueries_sent AS
SELECT query_id
      ,exp_id
      ,query_guid
      ,query
      ,created
  FROM tqueries
 WHERE incoming = 0;
CREATE VIEW vqueries_received AS
SELECT query_id
      ,exp_id
      ,query_guid
      ,query
      ,created
  FROM tqueries
 WHERE incoming = 1;
CREATE VIEW vnode_connections AS
SELECT n1.ip_address ip_address
      ,n1.port port
      ,n2.ip_address peer_ip_address
```

```
       ,n2.port peer_port
       ,e.description
       ,n1.exp_id
       ,c.node_id1 node_id
       ,c.node_id2 peer_node_id
  FROM tnode_connections c
 INNER JOIN tnodes n1
    ON n1.node_id = c.node_id1
 INNER JOIN tnodes n2
    ON n2.node_id = c.node_id2
 INNER JOIN texperiments e
    ON e.exp_id = n1.exp_id
 WHERE n1.exp_id = n2.exp_id;
CREATE VIEW vnode_importance AS
SELECT n1.ip_address ip_address
       ,n1.port port
       ,n2.ip_address peer_ip_address
       ,n2.port peer_port
       ,i.importance
       ,i.aver_n_hops
       ,i.perc_hits
       ,i.total_hits
       ,e.description
       ,n1.exp_id
       ,i.query_id
       ,i.node_id1 node_id
       ,i.node_id2 peer_node_id
  FROM tnode_importance i
 INNER JOIN tnodes n1
    ON n1.node_id = i.node_id1
 INNER JOIN tnodes n2
    ON n2.node_id = i.node_id2
 INNER JOIN texperiments e
    ON e.exp_id = n1.exp_id
 WHERE n1.exp_id = n2.exp_id;
CREATE VIEW vnode_reachability AS
SELECT n1.ip_address ip_address
      ,n1.port port
      ,n2.ip_address peer_ip_address
      ,n2.port peer_port
      ,r.distance
      ,e.description
      ,n1.exp_id
      ,r.node_id1 node_id
      ,r.node_id2 peer_node_id
  FROM tnode_reachability r
 INNER JOIN tnodes n1
    ON n1.node_id = r.node_id1
 INNER JOIN tnodes n2
    ON n2.node_id = r.node_id2
 INNER JOIN texperiments e
    ON e.exp_id = n1.exp_id
 WHERE n1.exp_id = n2.exp_id;
CREATE VIEW vquery_sources AS
SELECT s.query_id
      ,s.node_id
      ,s.created
      ,n.exp_id
      ,n.ip_address
      ,n.port
      ,q.query_guid
      ,q.query
  FROM tquery_sources s
 INNER JOIN tnodes n
     ON n.node_id = s.node_id
 INNER JOIN tqueries q
    ON q.query_id = s.query_id;
CREATE VIEW vreplies AS
```

```
SELECT r.reply_id
       ,r.query_id
      ,r.node_id_to
       ,r.node_id_frm
       ,r.node_id_thru
       ,r.reply_guid
       ,r.is_push
       ,r.vendor
       ,r.hops
       ,r.ttl
       ,r.file_name
       ,r.created
       ,q.exp_id
       ,q.query_guid
       ,q.query
       ,q.incoming
       ,n1.ip_address to_ip_address
       ,n1.port to_port
       ,n2.ip_address frm_ip_address
       ,n2.port frm_port
       ,n3.ip_address thru_ip_address
       ,n3.port thru_port
  FROM treplies r
 INNER JOIN tqueries q
    ON q.query_id = r.query_id
 INNER JOIN tnodes n1
    ON n1.node_id = r.node_id_to
 INNER JOIN tnodes n2
    ON n2.node_id = r.node_id_frm
 INNER JOIN tnodes n3
    ON n3.node_id = r.node_id_thru;
```

APPENDIX 2

```
import random
def barabasi_random_graph(num_nodes
, min_edges_per_node
, max_edges_per_node): '''Generates Barabasi random graph '''
 repeated_nodes = []
  G = {}
   targets = []
   for i in range(0, num_nodes):
     G[str(i)] = {}

   for i in range(0, min_edges_per_node):
      targets.append(str(i))

   source = min_edges_per_node
   while source < num_nodes:
      for x in targets:
        G[x][str(source)] = 1
        G[str(source)][x] = 1
         print (x, "␣␣␣", str(source))

      repeated_nodes.extend(targets)
      repeated_nodes.extend([str(source)]*min_edges_per_node)

      targets = set()
      while len(targets) < min_edges_per_node:
        x = random.choice(repeated_nodes)
         if(len(G[x]) < max_edges_per_node):
           targets.add(x)
```

```
    source += 1
return G
```

APPENDIX 3

```
import random
def barabasi_random_graph(num_nodes \
                         ,min_edges_per_node \
                         ,max_edges_per_leaf \
                         ,max_edges_per_ultra \
                         ,percent_ultra \
                         ,percent_ultra_to_ultra):
'''Generates Gnutella graph with super nodes and \
leaves using Barabasi distribution for each '''
    repeated_nodes = []
    repeated_ultra = []
    targets = set()
    G = {}

    # Parameter validation. Disallow bad values
    if (max_edges_per_leaf > max_edges_per_ultra):
        max_edges_per_ultra = max_edges_per_leaf
    if (percent_ultra >= 1.0):
        percent_ultra = 0.0
    if (percent_ultra_to_ultra >= 1.0):
        percent_ultra_to_ultra = 0.5

    min_edges_per_ultra = max_edges_per_leaf + 1
    # Select ultra nodes based on percent of network
    # that is supposed to be ultra peers (percent_ultra)
    ultra_nodes = set()
    while ( ( (len(ultra_nodes)+1) / num_nodes ) < percent_ultra ):
        new_ultra = random.randint(0, num_nodes-1)
        if ( new_ultra not in ultra_nodes ):
            ultra_nodes.add(int(new_ultra))
            repeated_ultra.extend([int(new_ultra)]);

    # Initialize G and put min_edges_per_node
    # edges in for each node in the network
    # Do not include ultra nodes in repeated_nodes
    # as they are populated previously in repeated_ultra
    for i in range(0, num_nodes):
        G[int(i)] = {}
        if (  i not in ultra_nodes ):
            repeated_nodes.extend([int(i)])

# Start processing at node 0
source = 0
while source < num_nodes:
    targets = set()
    # get current number of edges for source node
    curr_len_src = len(G[int(source)])
    # Setup up edge limits based on whether
    # source is ultra peer or leaf
    if ( source in ultra_nodes ):
        is_src_super_node = 1
        has_super_node = 1
        curr_min_edges = max_edges_per_leaf+1
        curr_max_edges = max_edges_per_ultra
    else:
        is_src_super_node = 0
        has_super_node = 0
        curr_min_edges = min_edges_per_node
        curr_max_edges = max_edges_per_leaf
        # Test if source already has an ultra peer
        for n in G[int(source)]:
            if (n in ultra_nodes):
                has_super_node = 1
                break

    # Stop loop if curr_len_src gets to max
    # Continue loop while less than min edges with random chance of
    # adding more up to max that decreases with additional edges
    while ( curr_len_src < curr_max_edges and \
            ( curr_len_src < curr_min_edges or \
              random.random() < \
            (1.0 - (float(curr_len_src)) / float(curr_max_edges)) ) ) :
        new_edge = int(source)
        # Loop until a valid edge candidate is found
        while( int(new_edge) == int(source) or \
                source in G[int(new_edge)] or \
                new_edge in G[int(source)] or \
                new_edge in targets ):
            # percent_ultra_to_ultra is the chance that ultra
            # nodes will connect to other ultra nodes
            # Need to avoid fracturing
            # Give leaf nodes with super nodes already
            # a chance to connect to multiple super nodes based
            # on the current number of connections
            if is_src_super_node == 1 and \
                random.random() < percent_ultra_to_ultra:
                new_edge = random.choice(repeated_ultra)
            elif has_super_node == 0 or \
                  random.random() < (1.0 - \
                  (float(curr_len_src)) / float(max_edges_per_ultra)):
                new_edge = random.choice(repeated_ultra)
            else:
                new_edge = random.choice(repeated_nodes)

        if new_edge in ultra_nodes:
            new_edge_min_edges = max_edges_per_leaf+1
            new_edge_max_edges = max_edges_per_ultra
        else:
            new_edge_min_edges = min_edges_per_node
            new_edge_max_edges = max_edges_per_leaf

        if(len(G[int(new_edge)]) < new_edge_max_edges):
            targets.add(int(new_edge))

            if( has_super_node == 0 and \
                 (new_edge in ultra_nodes or source in ultra_nodes) ):
                has_super_node = 1
            # Maintain list of edges for source edge added
            if(curr_len_src > 1):
                if ( is_src_super_node == 1 ):
                    repeated_ultra.extend([int(source)])
                else:
                    repeated_nodes.extend([int(source)])
            # Maintain list of edges for new_edge added
            if((len(G[int(new_edge)])+1) > 1):
                if ( new_edge in ultra_nodes ):
                    repeated_ultra.extend([int(new_edge)])
                else:
                    repeated_nodes.extend([int(new_edge)])

            # Increment length of source
            curr_len_src += 1
        else:
            continue

        if(curr_len_src >= curr_max_edges):
            break

    for x in targets:
        G[int(source)][int(x)] = 1
        G[int(x)][int(source)] = 1


    source += 1

# Print the output
sorted_G = sorted(G)
for first in sorted_G:
    sorted_First = sorted(G[int(first)])
    for second in sorted_First:
        if ( G[int(first)][int(second)] == 1 ):
            print ("%6d␣␣%6d" % (int(first), int(second)))
            G[int(second)][int(first)] = 0
            G[int(first)][int(second)] = 0

return G
```

APPENDIX 4

```
package edu.louisville.cecs.darees01;


import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;

import java.io.IOException;

import java.util.Date;
import java.util.Properties;
import java.util.Random;
import java.util.Vector;

/**
 * Class used to generate a file distribution for a network Generates files for
 * a 50,000 node network Assigns 1-1000 files to a node Each node has a 30%
 * chance to have files Each node that has a file has a 1% chance to have
 * contraband (0.3% chance of contraband for all nodes) Contraband file names
 * are generated from a distribution of keywords A node with contraband has a
 * 57% of having exclusively contraband files Nodes with a mix of contraband and
 * regular files have a 0.45% contraband files
 *
 * Input parameters can overwrite all values, but the defaults are as described
 *
 *
 * @author Derek Reese
 *
 */
```

```
public class GenerateFileDistribution
{
    public enum AppProperties
    {
        FILE_DIST("FileDist"),
        CONTRABAND_KEYWORDS(
            "ContrabandKeywords"),
        NONCONTRABAND_KEYWORDS(
            "NoncontrabandKeywords"),
        CONTRABAND_KEY_DIST(
            "ContrabandKeyDist"),
        NONCONTRABAND_KEY_DIST(
            "NoncontrabandKeyDist"),
        NUM_PEERS("NumPeers"),
        MIN_FILES_PER_PEER(
            "MinFilesPerPeer"),
        MAX_FILES_PER_PEER(
            "MaxFilesPerPeer"),
        CHANCE_NODE_HAS_CONTENT(
            "ChanceNodeHasContent"),
        CHANCE_NODE_HAS_CONTRABAND(
            "ChanceNodeHasContraband"),
        CHANCE_NODE_HAS_ALL_CONTRABAND(
            "ChanceNodeHasAllContraband"),
        PERCENT_CONTRA_FOR_MIX_NODE(
            "PercentContraForMixNode");

        private String Ms_name;

        AppProperties(String Ps_name)
        {
            Ms_name = Ps_name;
        }
        public String getName()
        {
            return Ms_name;
        }
    }

    public GenerateFileDistribution()
    {
    }

    /**
     * Static function used to generate distrubtion Distribution is written to
     * C:\\dar\\school\\thesis\\gen_vertex_files.txt
     *
     * @param pAs_inparams
     */
    public static void main(
                            String[] PAs_args)
    {
        // Load properties from file
        Properties o_properties =
            new Properties();
        if ( PAs_args.length > 0 )
        {
            try
            {
                FileInputStream o_prop_in =
                    new FileInputStream(
                                        PAs_args[0]);
                o_properties
                    .load(o_prop_in);
            }
            catch (FileNotFoundException Eo_fnf)
            {
```

```
        System.out
            .println("Properties file does not exist: "
                + PAs_args[0]);
    }
    catch (IOException Eo_io)
    {
        System.out
            .println("Error opening properties file: "
                + Eo_io
                    .getMessage());
    }
}

int i_num_nodes = 50000, i_min_num_files =
    1, i_max_num_files = 1000, i_num_files_for_i =
    0, i_total_contra = 0, i_total_noncontra =
    0, i_nodes_with_files = 0, i_nodes_with_contra =
    0;

double d_perc_node_has_files =
    0.3, d_perc_node_has_contra =
    0.01, d_perc_node_has_all_contra =
    .57, d_perc_contra_files =
    0.0045;

boolean b_i_has_contra = false, b_only_contra_for_i =
    false, b_assigned_a_contra_for_i =
    false;

String s_file_dist =
    new String(
            "C:\\dar\\school\\thesis\\gen_vertex_files.txt"), s_contra_keys =
    new String(
            "BAD|TERRIBLE|AWFUL|HORRID|WORST|MALICIOUS"), s_noncontra_keys =
    new String(
                "GOOD|GREAT|NICE|EXCELLENT|SUPER|SPLENDID"), s_contra_key_dist =
    new String(
                "0.4|0.3|0.2|0.05|0.03|0.02"), s_noncontra_key_dist =
    new String(
                "0.3|0.2|0.2|0.1|0.1|0.1");

// Read properties
s_file_dist =
    o_properties
        .getProperty(
                    AppProperties.FILE_DIST
                        .getName(),
                    s_file_dist);

s_contra_keys =
    o_properties
        .getProperty(
                    AppProperties.CONTRABAND_KEYWORDS
                        .getName(),
                    s_contra_keys);

s_noncontra_keys =
    o_properties
        .getProperty(
                    AppProperties.NONCONTRABAND_KEYWORDS
                        .getName(),
                    s_noncontra_keys);

s_contra_key_dist =
    o_properties
        .getProperty(
                    AppProperties.CONTRABAND_KEY_DIST
                        .getName(),
                    s_contra_key_dist);


s_noncontra_key_dist =
    o_properties
        .getProperty(
                    AppProperties.NONCONTRABAND_KEY_DIST
                        .getName(),
                    s_noncontra_key_dist);


try
{
    i_num_nodes =
        Integer
            .valueOf(o_properties
                .getProperty(AppProperties.NUM_PEERS
                    .getName()));
}
catch (NumberFormatException Eo_nf)
{
    System.err
        .println("Bad value for num nodes: "
            + o_properties
                .getProperty(AppProperties.NUM_PEERS
                    .getName()));

    i_num_nodes = 50000;
}


try
{
    i_min_num_files =
        Integer
            .valueOf(o_properties
                .getProperty(AppProperties.MIN_FILES_PER_PEER
                    .getName()));
}
catch (NumberFormatException Eo_nf)
{
    System.err
        .println("Bad value for min files per peer: "
            + o_properties
                .getProperty(AppProperties.MIN_FILES_PER_PEER
                    .getName()));
    i_min_num_files = 1;
}

try
{
    i_max_num_files =
        Integer
            .valueOf(o_properties
                .getProperty(AppProperties.MAX_FILES_PER_PEER
                    .getName()));
}
catch (NumberFormatException Eo_nf)
{
    System.err
        .println("Bad value for max files per peer: "
            + o_properties
                .getProperty(AppProperties.MAX_FILES_PER_PEER
                    .getName()));
    i_max_num_files = 1000;
}

try
{
    d_perc_node_has_files =
        Double
```

```
                .valueOf(o_properties
                    .getProperty(AppProperties.CHANCE_NODE_HAS_CONTENT
                        .getName()));
}
catch (NumberFormatException Eo_nf)
{
    System.err
        .println("Bad value for % chance node has content: "
            + o_properties
                .getProperty(AppProperties.CHANCE_NODE_HAS_CONTENT
                    .getName()));
    d_perc_node_has_files = 0.3;
}
catch (NullPointerException Eo_np)
{
    System.err
        .println("Null value for % chance node has content");
    d_perc_node_has_files = 0.3;
}

try
{
    d_perc_node_has_contra =
        Double
            .valueOf(o_properties
                .getProperty(AppProperties.CHANCE_NODE_HAS_CONTRABAND
                    .getName()));
}
catch (NumberFormatException Eo_nf)
{
    System.err
        .println("Bad value for % chance node has contraband: "
            + o_properties
                .getProperty(AppProperties.CHANCE_NODE_HAS_CONTRABAND
                    .getName()));
    d_perc_node_has_contra =
        0.01;
}
catch (NullPointerException Eo_np)
{
    System.err
        .println("Null value for % chance node has contraband");
    d_perc_node_has_contra =
        0.01;
}

try
{
    d_perc_node_has_all_contra =
        Double
            .valueOf(o_properties
                .getProperty(AppProperties.CHANCE_NODE_HAS_ALL_CONTRABAND
                    .getName()));
}
catch (NumberFormatException Eo_nf)
{
    System.err
        .println("Bad value for % chance node has all contraband: "
            + o_properties
                .getProperty(AppProperties.CHANCE_NODE_HAS_ALL_CONTRABAND
                    .getName()));
    d_perc_node_has_all_contra =
        0.57;
}
catch (NullPointerException Eo_np)
{
    System.err
        .println("Null value for % chance node has all contraband");
    d_perc_node_has_all_contra =
        0.57;
}

try
{
    d_perc_contra_files =
        Double
            .valueOf(o_properties
                .getProperty(AppProperties.PERCENT_CONTRA_FOR_MIX_NODE
                    .getName()));
}
catch (NumberFormatException Eo_nf)
{
    System.err
        .println("Bad value for % contraband for mix node: "
            + o_properties
                .getProperty(AppProperties.PERCENT_CONTRA_FOR_MIX_NODE
                    .getName()));
    d_perc_contra_files =
        0.0045;
}
catch (NullPointerException Eo_np)
{
    System.err
        .println("Null value for % contraband for mix node");
    d_perc_contra_files =
        0.0045;
}

Vector<String> o_contra_keywords =
    new Vector<String>(), o_noncontra_keywords =
    new Vector<String>();
Vector<Double> o_contra_keydist =
    new Vector<Double>(), o_noncontra_keydist =
    new Vector<Double>();

parseKeywords(s_contra_keys,
              o_contra_keywords);
parseKeywords(s_noncontra_keys,
              o_noncontra_keywords);
parseKeyDist(s_contra_key_dist,
             o_contra_keydist);
parseKeyDist(
             s_noncontra_key_dist,
             o_noncontra_keydist);

String[] As_contraband_terms =
    new String[o_contra_keywords
        .size()], As_good_terms =
    new String[o_noncontra_keywords
        .size()];
double[] Ad_contraband_perc =
    new double[o_contra_keywords
        .size()], Ad_good_perc =
    new double[o_noncontra_keywords
        .size()];

o_contra_keywords
    .toArray(As_contraband_terms);
o_noncontra_keywords
    .toArray(As_good_terms);

// Initialize distributions to 0.0
int i_i = 0;
for (i_i = 0; i_i < Ad_contraband_perc.length; i_i++)
{
    Ad_contraband_perc[i_i] =
        0.0;
}

for (i_i = 0; i_i < Ad_good_perc.length; i_i++)
{
    Ad_good_perc[i_i] = 0.0;
}

// Read input parameters into distributions
i_i = 0;
for (Double o_i : o_contra_keydist)
{
    Ad_contraband_perc[i_i] =
        o_i.doubleValue();

    i_i++;
    if ( i_i >= Ad_contraband_perc.length )
    {
        break;
    }
}

i_i = 0;
for (Double o_i : o_noncontra_keydist)
{
    Ad_good_perc[i_i] =
        o_i.doubleValue();

    i_i++;
    if ( i_i >= Ad_good_perc.length )
    {
        break;
    }
}
```

```
// Add up total of distribution (should be = 1.0)
double d_contra_total = 0.0, d_noncontra_total =
    0.0;
for (i_i = 0; i_i < Ad_contraband_perc.length; i_i++)
{
    d_contra_total +=
        Ad_contraband_perc[i_i];
}

if ( d_contra_total > 1.0 )
{
    System.out
        .println("Total percentage of contraband terms greater than 100%: "
            + d_contra_total);
    return;
}

for (i_i = 0; i_i < Ad_good_perc.length; i_i++)
{
    d_noncontra_total +=
        Ad_good_perc[i_i];
}

if ( d_noncontra_total > 1.0 )
{
    System.out
        .println("Total percentage of good terms greater than 100%: "
            + d_noncontra_total);
    return;
}

// Test for match of keywords to distributions and correct if necessary
if ( o_contra_keywords.size() < o_contra_keydist
    .size() )
{
    System.out
        .println("Contraband distribution has "
            + (o_contra_keydist
                .size() - o_contra_keywords
                .size())
            + " values that will be ignored");
}
else if ( o_contra_keywords
    .size() > o_contra_keydist
    .size() )
{
    System.out
        .println("The last "
            + (o_contra_keywords
                .size() - o_contra_keydist
                .size())
            + " contraband keywords will evenly distributed");

    // Distribute to last of keywords
    for (i_i =
        o_contra_keywords
            .size(); i_i < Ad_contraband_perc.length; i_i++)
    {
        Ad_contraband_perc[i_i] =
            (1.0 - d_contra_total)
                / (o_contra_keywords
                    .size() - o_contra_keydist
                    .size());
    }
}

if ( o_noncontra_keywords
    .size() < o_noncontra_keydist
    .size() )
{
    System.out
        .println("Noncontraband distribution has "
            + (o_noncontra_keydist
                .size() - o_noncontra_keywords
                .size())
            + " values that will be ignored");
}
else if ( o_noncontra_keywords
    .size() > o_noncontra_keydist
    .size() )
{
    System.out
        .println("The last "
            + (o_noncontra_keywords
                .size() - o_noncontra_keydist
                .size())
            + " noncontraband keywords will evenly distributed");

    // Distribute to last of keywords
    for (i_i =
        o_noncontra_keywords
            .size(); i_i < Ad_good_perc.length; i_i++)
    {
        Ad_good_perc[i_i] =
            (1.0 - d_noncontra_total)
                / (double) (o_noncontra_keywords
                    .size() - o_noncontra_keydist
                    .size());
    }
}
```

```
System.out
    .println("Contraband:");
for (i_i = 0; i_i < As_contraband_terms.length; i_i++)
{
    System.out
        .println("\t"
            + As_contraband_terms[i_i]
            + " with dist "
            + Ad_contraband_perc[i_i]);
}

System.out.println("Good:");
for (i_i = 0; i_i < As_good_terms.length; i_i++)
{
    System.out.println("\t"
        + As_good_terms[i_i]
        + " with dist "
        + Ad_good_perc[i_i]);
}

// Start Generation
Random o_randgen =
    new Random(new Date()
        .getTime());

String s_searchterm = null, s_line =
    null;

if ( As_contraband_terms.length != Ad_contraband_perc.length )
{
    System.out
        .println("Each contraband term must have a percent distribution");
    return;
}
```

```
if ( As_good_terms.length != Ad_good_perc.length )
{
    System.out
        .println("Each good term must have a percent distribution");
    return;
}

FileWriter o_filewriter = null;
try
{
    o_filewriter =
        new FileWriter(
                        s_file_dist);
}
catch (IOException e_ioexc)
{
    e_ioexc.printStackTrace();
    return;
}

for (i_i = 0; i_i < i_num_nodes; i_i++)
{
    // Determine if node has files 30% chance
    if ( o_randgen.nextDouble() < d_perc_node_has_files )
    {
        // If node has files place a random # of files between 1 and
        // 1000 files
        i_num_files_for_i =
            (o_randgen
                .nextInt() % (i_max_num_files - i_min_num_files))
                + i_min_num_files;

        i_nodes_with_files++;
```

```
// Determine if node contains contraband 1% * 30% = 0.3% chance
if ( o_randgen
    .nextDouble() < d_perc_node_has_contra )
{
    b_i_has_contra =
        true;

    // If node has contraband then give 57% chance of containing
    // 100% contraband
    if ( o_randgen
        .nextDouble() < d_perc_node_has_all_contra )
    {
        b_only_contra_for_i =
            true;
    }
    else
    {
        b_only_contra_for_i =
            false;
    }
}
else
{
    b_i_has_contra =
        false;
    b_only_contra_for_i =
        false;
}

b_assigned_a_contra_for_i =
    false;
for (int i_j = 0; i_j < i_num_files_for_i; i_j++)
{
```

```
double d_rand_for_file =
    o_randgen
        .nextDouble();
int i_file_picked;
if ( b_i_has_contra
    && (b_only_contra_for_i || o_randgen
        .nextDouble() < d_perc_contra_files) )
{
    i_file_picked =
        Ad_contraband_perc.length;
    while ( i_file_picked >= Ad_contraband_perc.length )
    {
        double d_total =
            0.0;
        for (i_file_picked =
            0; i_file_picked < Ad_contraband_perc.length; i_file_picked++)
        {
            d_total +=
                Ad_contraband_perc[i_file_picked];
            if ( d_rand_for_file < d_total )
            {
                break;
            }
        }

        // Choose a new file since last one could not be
        // selected
        d_rand_for_file =
            o_randgen
                .nextDouble();
    }

    s_searchterm =
        new String(
                    As_contraband_terms[i_file_picked]
                        + i_j);
          i_total_contra++;

          b_assigned_a_contra_for_i =
               true;
     }
     else
     {
          i_file_picked =
               Ad_good_perc.length;
          while ( i_file_picked >= Ad_good_perc.length )
          {
               double d_total =
                    0.0;
               for (i_file_picked =
                    0; i_file_picked < Ad_good_perc.length; i_file_picked++)
               {
                    d_total +=
                         Ad_good_perc[i_file_picked];
                    if ( d_rand_for_file < d_total )
                    {
                         break;
                    }
               }

               // Choose a new file since last one could not be
               // selected
               d_rand_for_file =
                    o_randgen
                         .nextDouble();
          }

          s_searchterm =
                new String(
                            As_good_terms[i_file_picked]
                               + i_j);
            i_total_noncontra++;
        }

        s_line =
            new String(
                    i_i
                        + " "
                        + s_searchterm
                        + "\r\n");
        try
        {
            o_filewriter
                .write(
                    s_line,
                    0,
                    s_line
                        .length());
        }
        catch (IOException e_ioexc)
        {
            System.out
                .println("Could not write line:"
                    + s_line);
        }
        // System.out.println(i_i + " " + s_searchterm);
    }

    if ( b_assigned_a_contra_for_i )
    {
        i_nodes_with_contra++;
    }
```

```
            }
        }

        System.out
            .println("Total contraband files: "
                + i_total_contra);
        System.out
            .println("Total non-contraband files: "
                + i_total_noncontra);
        System.out
            .println("Total files: "
                + (i_total_contra + i_total_noncontra));
        System.out
            .println("Total nodes with contraband: "
                + i_nodes_with_contra);
        System.out
            .println("Total nodes with files: "
                + i_nodes_with_files);
        System.out
            .println("Average files per node: "
                + (i_total_contra + i_total_noncontra)
                / i_nodes_with_files);
    }

    public static void parseKeywords(
                                    String Ps_keywords,
                                    Vector<String> Po_keyword_list)
    {
        // Parse out keywords
        int i_cur_index = 0, i_nxt_index =
            0;
        while ( (i_nxt_index =
            Ps_keywords
                .indexOf('|',
                         i_cur_index)) >= 0 )
        {
            Po_keyword_list
                .add(Ps_keywords
                    .substring(
                               i_cur_index,
                               i_nxt_index));
            i_cur_index =
                i_nxt_index + 1;
        }

        if ( !Ps_keywords
            .substring(i_cur_index)
            .isEmpty() )
        {
            Po_keyword_list
                .add(Ps_keywords
                    .substring(i_cur_index));
        }
    }

    public static void parseKeyDist(
                                    String Ps_key_dist,
                                    Vector<Double> Po_key_dist)
    {
        // Parse out keywords
        int i_cur_index = 0, i_nxt_index =
            0;
        while ( (i_nxt_index =
            Ps_key_dist
                .indexOf('|',
                         i_cur_index)) >= 0 )
        {
            Po_key_dist
```

```
                    .add(Double
                        .valueOf(Ps_key_dist
                            .substring(
                                        i_cur_index,
                                        i_nxt_index)));
                i_cur_index =
                    i_nxt_index + 1;
            }

            if ( !Ps_key_dist
                .substring(i_cur_index)
                .isEmpty() )
            {
                Po_key_dist
                    .add(Double
                        .valueOf(Ps_key_dist
                            .substring(i_cur_index)));
            }
        }
    }
```

Bibliography

[1] David Barkai. 2001. *Peer-to-Peer Computing: Technologies for Sharing and Collaborating on the Net*. Intel Press

[2] Sun Microsystems Inc. 2005. "Sun Java System Communications Services 6 2005Q1 Deployment Planning Guide.", available from http://docs.sun.com/source/819-0063/index.html; accessed 16 February 2010

[3] Napster. Wikipedia. available from http://en.wikipedia.org/wiki/Napster; accessed 14 April 2010

[4] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma and Steven Lim. 2004. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. IEEE Communications Survey and Tutorial March 2004.

[5] United States General Accountability Office: Testimony before the Committee on the Judiciary. U.S. Senate. 2003. FILE SHARING PROGRAMS: Users of Peer-to-Peer Networks Can Readily Access Child Pornography. Statements of Linda D. Koontz. GAO-03-1115T.

[6] Stacie Rumemap. 2009. "Peer-to-Peer File Sharing: Pandora's Box of Child Porn?," Internet Source, 16 November 2009, available from http://stopchildpredators.org/pdf/ \ SCP_on_P2P_File_Sharing_Pandoras%C2% \ AD_Box_of_Child_Pornography_1109.pdf; accessed 15 April 2010.

[7] Stefan Saroiu, P. Krishna Gummadi, Steven D. Gribble. 2001. "A Measurement Study of Peer-to-Peer File Sharing Systems," available from http://www.cs.washington.edu/ \ homes/gribble/papers/mmcn.pdf; accessed 11 April 2010.

[8] Mihajlo Jovanović. 2001. MODELING PEER-TO-PEER NETWORK TOPOLOGIES THROUGH "SMALL-WORLD" MODELS AND POWER LAWS. IX TELECOMMUNICATIONS FORUM TELFOR 2001, Belgrade, 20-22.11.2001.

[9] Murali Krishna Ramanathan, Vana Kalogeraki, Jim Pruyne. 2002. Finding Good Peers in Peer-to-Peer Networks. Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02), 1530-2075/02.

[10] Liu Hongbo, Ajith Abraham, Youakim Badr. 2009. “Neighbor Selection in Peer-to-Peer Overlay Networks: A Swarm Intelligence Approach,” available from http://www.softcomputing.net/pervasive.pdf ; accessed 11 April 2010

[11] Stafan Schmid, Roger Wattenhofer. 2007. “Structuring Unstructured Peer-to-Peer Networks.” In 14th Annual IEEE International Conference on High Performance Computing (HiPC). Goa, India: IEEE Press.

[12] Li Zhen-wu, Yang Jian, Shi Xu-dong, Bai Ying-cai. 2003. “A ’cluster’ based search scheme in peer-to-peer network,” Journal of Zheijang University SCIENCE V.4, No. 5, P.549-554

[13] Limewire Inc. 2008. “The Gnutella Protocol Specification v0.4.”, available from http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf; accessed 16 May 2010

[14] Henning Schulzrinne, Enrico Marocco, Emil Ivov. ”Security Issues and Solutions in Peer-to-Peer Systems for Realtime Communications” Internet Source, February 2010, available from http://tools.ietf.org/html/rfc5765; accessed 3 June 2010.

[15] Hughes Daniel, Walkerdine James, Coulson Geoff, and Gibson Stephen. 2006. "Is Deviant Behavior the Norm on P2P File-Sharing Networks?" IEEE Distributed Systems Online, vol. 7, no. 2, 2006, art. no. 0602-o1001.

[16] Dhurandher Sanjay, Misra Sudip, Obaidat Mohammad, Agarwel Raghua, Bhambhani Bhevnesh. May 2009. “Simulating Peer-to-Peer Networks.” Computer Systems and Applications, 2009. P.336-341

[17] Limewire Inc. 2008. “Gnutella Protocol Specification.”, available from http://wiki.limewire.org/index.php?title=GDF; accessed 2/7/2010

[18] United States National Institute of Standards and Technology. 2008. “The CFReDS Project.”, available from http://www.cfreds.nist.gov/; accessed 7/7/2010

[19] Wikipedia. 2010. “Welch’s t-test”, available from http://en.wikipedia.org /wiki/ Welch%27s_t_test; accessed 4/28/2010

[20] Wikipedia. 2010. “Barabasi-Albert Model”, available from http://en.wikipedia.org/wiki/Barab%C3%A1si%E2%80%93Al
accessed 9/12/2010

[21] Alfred Aho, John Hopcroft, and Jeffrey Ullman. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company.

[22] Nicholas Miles. 2010. ARCHITECUTRE ANALYSIS OF PEER-TO-PEER NETWORK STRUCTURE AND DATA EXCHANGES FOR DISTRIBUTION OF CONTRABAND MATERIAL. Masters Thesis. University of Louisville.

VITA

I graduated from the University of Louisville with a Bachelors of Science in Computer Engineering and Computer Science. I have worked for United Parcel Service since February 2001 and I am currently a Senior Programmer Analyst. Having left graduate studies in 2003 to participate in community and political organizations, I returned in 2008 to complete the work. As my employer has a strong interest in planning transportation networks, I was drawn to Dr. Nasraoui's interest in heuristic algorithms applied to large scale networks. The work under her guidance has enhanced my ability to analyze the tools for transportation network planning at UPS.