Peer-to-Peer Networks: Gnutella

Team #6

Frigo, Daniel **daf2@sfu.ca**

Alwaris, Tauseef **talwaris@sfu.ca**

Ma, Mark **mma58@sfu.ca**

Table of contents

Abstract

Peer-to-Peer (P2P) networks are networks that enable computers traditionally used for client-based applications to be used as client or server based applications. This type of network enables users to access/share files and information from a multitude of other users rather than relying on a dedicated server. This type of network makes better use of the capabilities of modern personal computers which have become widespread around the world. In this project we will model, simulate and analyze the performance of the Gnutella P2P network and contrast it with other types of networks.

Introduction

Gnutella is one of the largest P2P networks primarily used for file sharing. Nowadays, the word Gnutella is used to describe an open network protocol used by many clients. Theoretically, if everyone is willing to share their file online, then anything can be found through Gnutella. Basically, the protocol allows users to transfer files between each other instead of having to search through a main server or database. In Gnutella, a node or computer can connect to multiple other nodes simultaneously and these nodes in turn will also be connected to multiple other nodes. The protocol was meant to be set up in a way that information could be transferred from different sources simultaneously. Gnutella has many advantages that make it worthwhile to be studied and discussed; which is the main purpose of this project. For instance, it doesn't require the services of a company or any special equipment such as a central server to run this protocol. It is also a network that can be seen to work all the time. There is no key piece of equipment that should it fail would cause the network to be disrupted. In addition, the transfer of data is available under most of firewall systems.

Objectives

The objective of this project is to model the Gnutella network’s architecture and to study and analyze it from a design and performance point of view. We will build and simulate the network model using OPNET 16 software to study the behavioral and functional units of this network. We will prepare performance coverage studies of both the standard and self-modified versions of Gnutella in order to track any improvements in its behavior. Moreover, we will also reflect on its evolution from its predecessor P2P network “Napster” and will compare the two in light of performance and security. In order to study the performance of the Gnutella network, we will measure certain parameters such as delay, link utilization, link efficiency, throughput, bandwidth usage, etc. In the end, we will conclude the project by listing the advantages and disadvantages of this network and will also discuss the feasibility and applications of this network model in future use. In addition, we will briefly discuss some improvements that can be made to the model such as implementing Gnutella protocol V.0.6 and will discuss some of the improvements that have been made or are still being made in the Gnutella V.0.6 protocol.

Network Architecture

Gnutella is in some ways similar to the old Napster peer to peer network with one big important difference; it does not rely on a central server. The old Napster had users connecting to the network tell a central server the files they had available for sharing. Someone wishing to download a file would then query the server which in turn would look into its database to see which (if any) machines had that file available for downloading. The server would then reply with the machines that had the file and the end user would then connect to one of those machines for downloading the file. The Napster architecture is shown in the Figure 1 below.

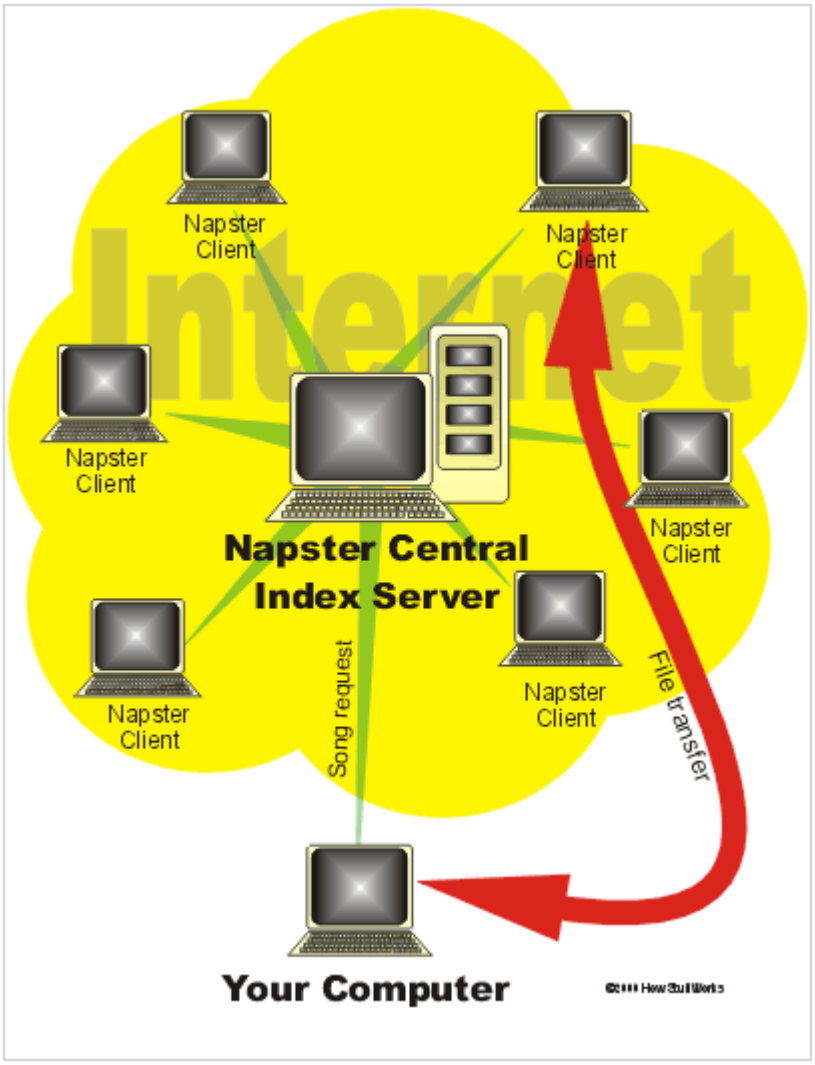

Figure 1: Napster network architecture [1]

Because of Napster's reliance on a central server to function properly, the system was very easy to shut down when legal issues concerning copyrighted material appeared.

Gnutella on the other hand uses a distributed query approach in order to find a file on another machine as shown in Figure 2 below [1].

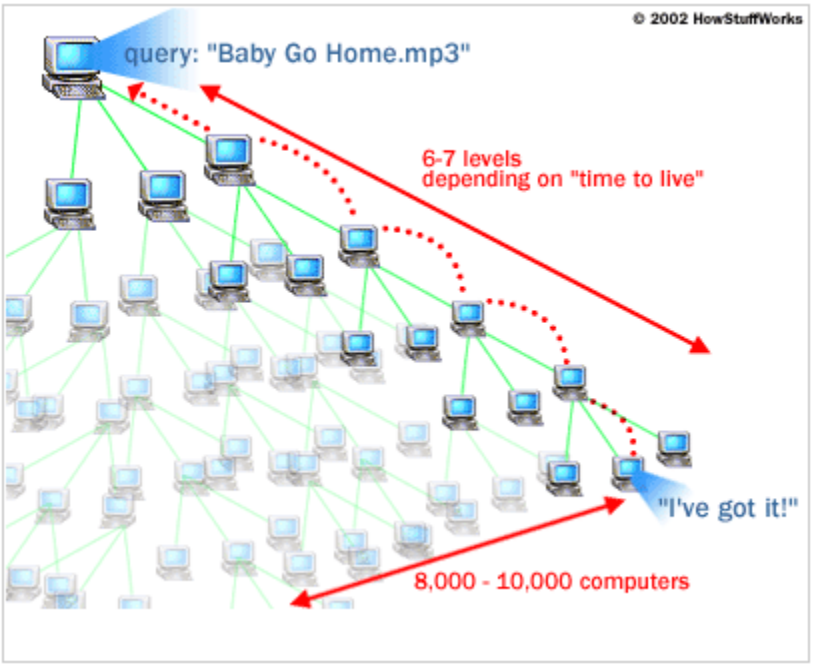

Figure 2: Gnutella distributed query approach [1]

A user wishing to download a file would send a query to all his neighbors who would in turn forward the request to all their neighbors and so on, up to a maximum number of hops. The machines that have the file available would send a reply back to the machine that initiated the query. As we can see, this approach does not rely on a central server to function and can be seen to work all the time. In addition, it is a lot harder to simply shut down this system since it cannot be shut down by simply removing a critical piece of equipment as in the case of Napster.

In the remainder of this section, we will discuss some of the details of the Gnutella protocol V.0.4. It should be noted that the current version of the Gnutella protocol is V.0.6, which contains some new features and improvements to V.0.4. Due to its increased complexity however, we will stick to the simpler V.0.4 and use this as a basis for studying the performance of the Gnutella network.

Gnutella Protocol Functionality

The Gnutella protocol defines the way in which servents communicate over the network. It consists of a set of descriptors used for communicating data between servents and a set of rules governing the inter-servent exchange of descriptors [2].

Servents are defined as nodes that act as both clients and servers. In this model, every client is a server, and vice versa. These so-called Gnutella servents perform tasks normally associated with both clients and servers [2].

Currently, the following descriptors are defined:

Ping	Used to actively discover hosts on the network. A servent receiving a Ping descriptor is expected to respond with one or more Pong descriptors
Pong	The response to a Ping. Includes the address of a connected Gnutella servent and information regarding the amount of data it is making available to the network
Query	The primary mechanism for searching the distributed network. A servent receiving a Query descriptor will respond with a QueryHit if a match is found against its local data set
QueryHits	The response to a Query. This descriptor provides the recipient with enough information to acquire the data matching the corresponding Query
Push	A mechanism that allows a firewalled servent to contribute file-based data to the network

Table 1: descriptors [2]

Ping/Pong

When a servent first connects to the Gnutella network, he must have the address of at least one other node connected to the network in order initiate a “PING” packet. Obtaining this initial address is carried out in an initialization process that is referred to as bootstrapping. There are a number of ways that are used to obtain this address such as including a pre-existing address list of working nodes in the Gnutella software, or using updated web caches of known nodes [3].

Once this initial address or addresses are obtained, the servent can then initiate a “PING” packet to all its neighbors he is connected to in the Gnutella network. These neighbors will in turn respond with a “PONG” packet as well as forward the “PING” packet to all its neighbors. This process will continue until the time to live (TTL) field of the packet reaches 0. The TTL field starts at an initial value and is decremented by 1 each time the packet traverses a node. This process is shown in Figure 3 below for an initial TTL value of 5.

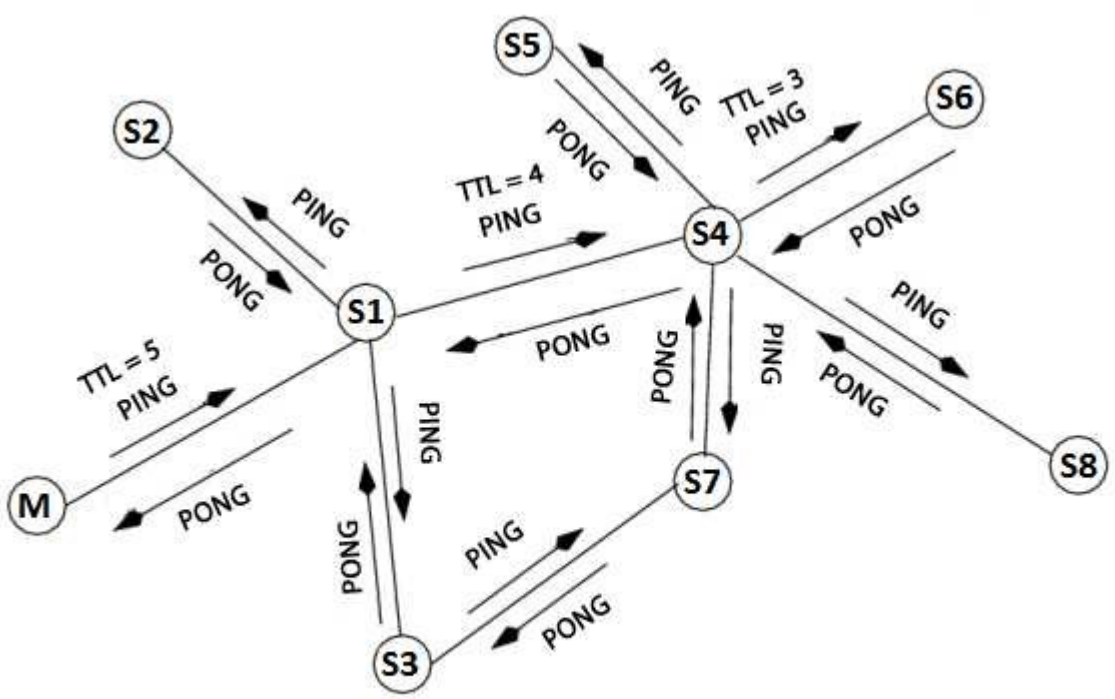

Figure 3: Ping Pong functionality in Gnutella [4]

Node M sends a PING packet to his neighbor, servent S1, who will then forward it to all his neighbors and so on. Each servent receiving a PING will respond with a PONG destined to the node who initiated the PING, in this case, node M. Node M uses these PONG packets to determine nodes he could potentially connect to for file sharing. These PONG packets will traverse the same path as the PING packet.

Query/Query Hit

Queries are done in the same manner as PING packets. A servent wishing to download a file would send a QUERY packet to all its neighbors who would then forward the QUERY to all their neighbors, and so on. Any servent containing the desired file would reply with a QUERY HIT packet. As is the case for Ping Pong, Query hits would travel the same path as Queries. This process is shown in Figure 4 below.

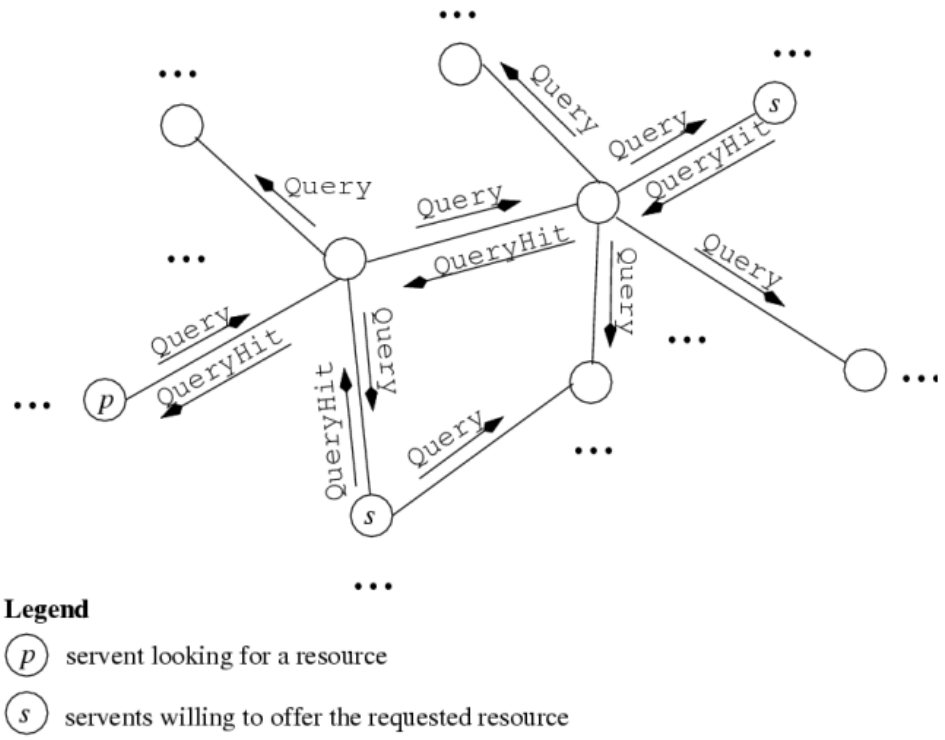

Figure 4: Locating Resources in a Gnutella-like P2P environment [4]

Once the node receives a QUERY HIT he can begin negotiating a TCP connection to the servent containing the file. Once the connection is set up, file transfer can begin using HTTP.

Push

In the case that the servent containing the file is firewalled, the servent who initiated the query can send a PUSH request to indicate to the source servent to initiate the connection instead. This functionality however will not be modeled in this project.

Modeling Gnutella

Figure 5 below shows a simple diagram illustrating the idea of Gnutella simulation.

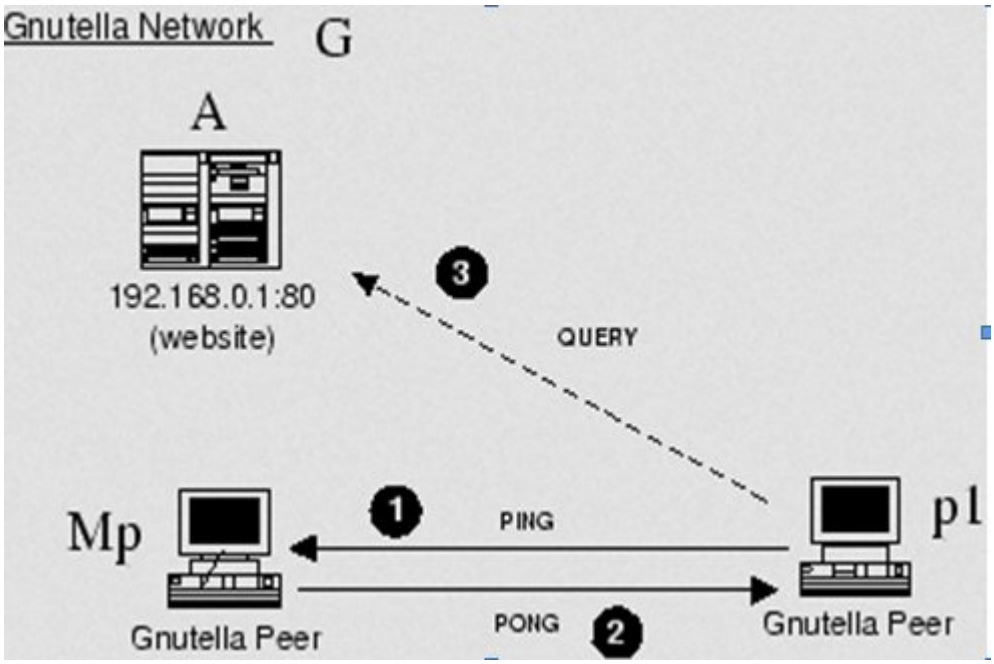

Figure 5: A simple diagram illustrate the idea of Gnutella simulation

Packet Format

In order to properly model the Gnutella network, we will first define a packet format which all servents are going to use. The packet format is shown in Figure 6 below.

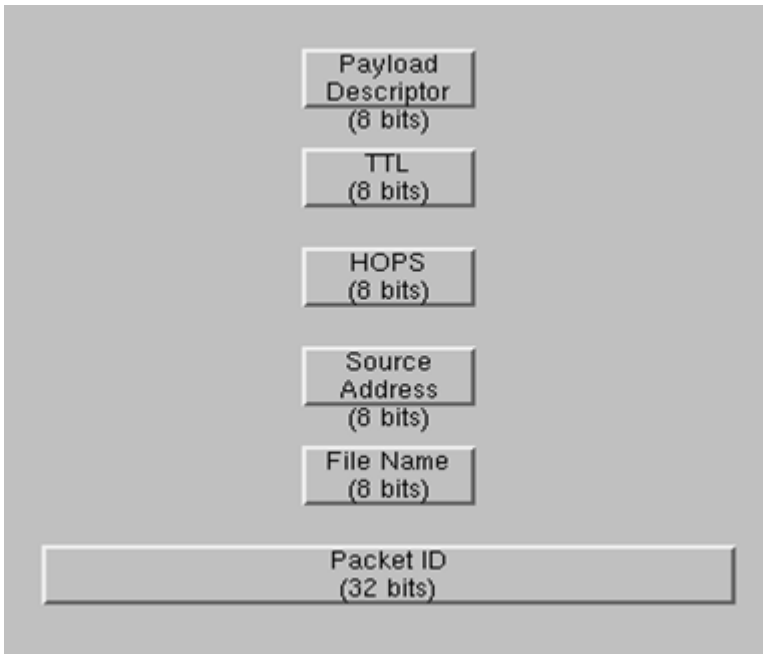

Figure 6: The packet format of the simulation

We can see it consists of 6 fields. The length of each field was chosen somewhat arbitrarily but we figured a byte for each field was realistic and more then capable of handling all the different values the fields can take. The choice for the last field being 4 bytes will be discussed later in the

Packet ID section. The function of these fields is discussed below

1) Payload Descriptor

This field determines what type of packet this is. Based on protocol specifications we got from reference [2] we assign this field 1 of 4 distinct hexadecimal values, based on what type of packet it is, as shown in Table 2 below.

HEX value	Packet
0x00	PING
0x01	PONG
0x80	QUERY
0x81	QUERY HIT

Table 2: Payload Descriptor

2) TTL (Time to Live)

This filed determines the number of servents a packet is to traverse. Each time a packet arrives at a servent; its TTL value is decremented by 1. If its TTL value is 0 after decrementing, the packet is destroyed, otherwise, the packet is forwarded to the next servent or servents in the network.

3) HOPS

This field keeps track of the number of servents a packet has traversed. Each time a packet arrives at a servent, its HOPS value is incremented by 1.

4) Source Address

This field contains the address of the servent that initiated the packet. This field is essential for routing back response packets, such as PONGS or QUERY HITS, to the appropriate servent who initiated the packet.

5) File Name

This field is used to model file searching when sending QUERY packets to the network. Each time a servent wants to search for a file, it assigns a numeric value to its “File Name” field, which for our purposes will be equivalent to typing the name of a file. Note that this field will only need to be set when sending QUERY packets.

5) Packet ID

This field is a distinct numeric value that OPNET assigns to each packet it generates. We use this field to detect duplicate packets. Since each packet has a distinct Packet ID, if a servent receives a packet with the same Packet ID as a previous packet, it knows it’s a duplicate and can go ahead and discard the packet. Duplicate packet detection will be discussed in more detail in the process model section. The reason we set Packet ID to 4 bytes rather than 1 is to ensure that we have a sufficient number of bits to assign a unique Packet ID to every packet our simulation generates. 4 bytes can handle a total of 2^{32} different Packet ID’s and is more than sufficient for our simulation. In a realistic scenario, the number of packets could exceed this number, but only

after a significant period of time. After this period of time, the first Packet ID's can safely be reused without any fear of the packets being duplicate packets.

Node model

Figure 7 below demonstrates the module connections within each servent in the simulation. Each servent has 4 transmitters and 4 receivers. This configuration allows us to connect each servent to a maximum of 4 other servents. The servent model also contains 2 separate traffic generators ("src_PING", "src_QUERY"). This allows us to generate PING and QUERY packets independently. These traffic generators use the "simple source" model. Finally we have a processor module ("proc_pkt") where everything is connected to. This module is responsible for processing all the packets that arrive and performing appropriate actions. The process model for this module is discussed in more detail in the next section.

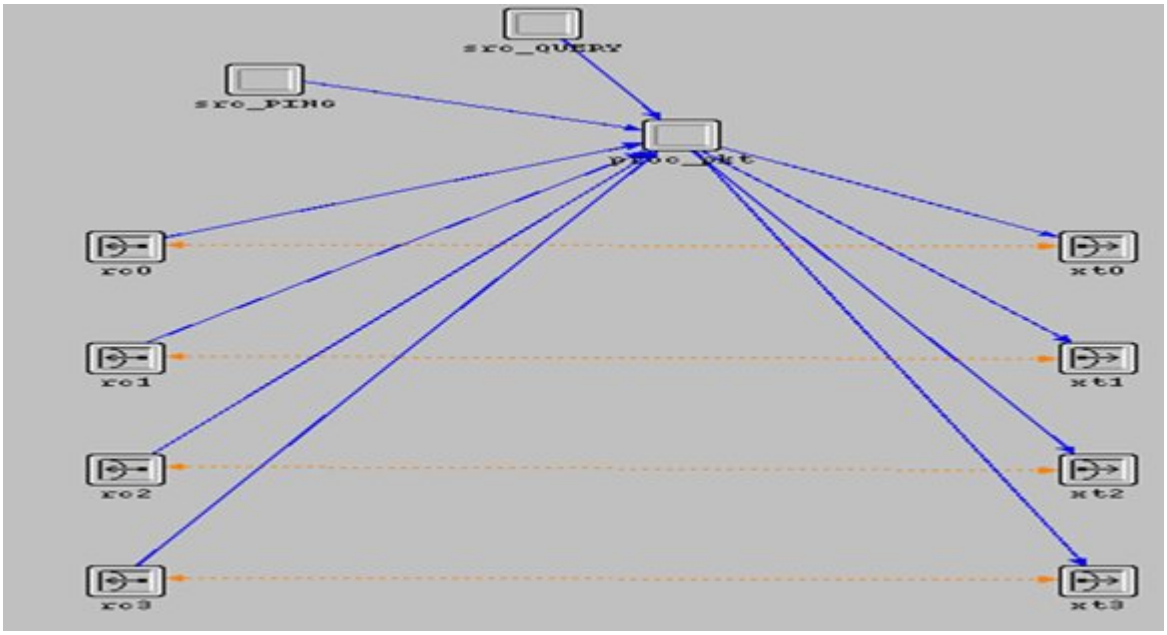

Figure 7: The interconnection within each node

Process Model

The process model shown in Figure 8 below describes the logic of the processor module shown in Figure 7. This model has 3 states ("init", "idle", "proc_pkt").

init state:

In the "init" state, we do a number of initialization functions. We initialize TTL value, load random number distributions, assign files randomly to each servent to model file searching, set up the routing tables and initialize other important variables. For simplicity we set up static routing tables that work specifically for the scenario at hand taking into account the connections we made between all the servents.

idle state:

In the idle state, we simply wait for a packet to arrive. A packet can come from 2 places. It can come from one of the traffic generator shown in Figure 7, or it can come from another servent. Since where a packet came from determines what actions need to be taken, we implement these 2 cases in different areas of our process model.

If a packet arrives from one of the traffic generators, the variable “SRC_ARRVL” will be TRUE and a transition from “idle” to itself will occur. During this transition, the function “transmit_pkt()” will execute. This function will determine whether the arriving packet is a PING or a QUERY packet by looking at which traffic generator the packet came from. It will then set the “Payload Descriptor” field appropriately as described in Table 2. It will also set the other packet fields appropriately (“TTL”, “HOPS”, “Source Address”, “File Name”, “Packet ID”). TTL is set to some initial value which we define at the beginning. HOPS value is initialized to 0. Source Address is set according to a “user id” attribute we set at the network model level. In order for this to work properly, we ensure each servent has a unique “user id”. File Name field is only set for QUERY packets. This value is set according to the outcome of a uniform integer random number distribution we load in the “init” state. If the packet is not a QUERY packet, this value is set to 0. In our simulation, we treat files with a value of 0 as invalid files. Once the packet fields are set, the servent sends the packet to all the neighbors he is connected to. A final task of the “transmit_pkt()” function is to increment a PING counter every time a PING packet is generated and increment a QUERY counter, each time a QUERY packet is generated. This will keep track of the number of PING and QUERY packets generated and is useful in order to see our results.

If on the other hand a packet is received from another servent, RCV_ARRVL will be TRUE and a transition from the “idle” state to the “proc_pkt” state will occur. This state processes the received packet and takes appropriate actions. This will be discussed in more detail in the next section.

SRC_ARRVL and RCV_ARRVL are defined in the Header Block of our code and can be found in the appendix.

proc_pkt state:

A transition to this state will occur only when a packet is received from another servent. In this state, we first read all the packet fields and determine whether the packet is a PING, PONG, QUERY, or QUERY HIT packet. Thus 4 different things can happen in this state depending on what type of packet it is. The 4 scenarios are described below.

1) PING packet received

If a PING packet is received, the first thing that will need to occur is to check if this packet is a duplicate. This is done by checking the Packet ID against the Packet ID's of previously received PING packets, which are stored in a local cache. If this is indeed a duplicate packet, the packet is discarded and no further action is taken. If it is not a duplicate packet, then we first cache the Packet ID. We then proceed to reply with a PONG packet with appropriate field values. Finally, we decrement the PING packet TTL value by 1 and if it's not 0, we forward the PING packet to all connected neighbors. If the TTL has reached 0, we discard the packet.

2) PONG packet received

If a PONG packet is received, we check the Source Address to see if the packet is destined to this node. If it is, we increment a counter keeping track of the number of PONG packets received and discard the packet. If the packet is not destined to this node, we decrement its TTL value by 1 and if it's not 0, we route the packet along the appropriate route based on the Source Address and routing table we set up in the "init" state. Since we don't expect to receive duplicate PONG packets in this model, there is no need to check for duplicate PONG packets.

3) QUERY packet received

If a QUERY packet is received, the actions taken are almost identical to the actions taken when a PING packet is received (exempting the replacement of PING with QUERY and PONG with QUERY HIT). The only difference is that the servent will now read the File Name field and check it against its local file database. For this simulation, we set up each servent to have a local data base of a maximum of 10 files. The servent will only respond with a QUERY HIT, if a match is found. If there is no match, the servent will not respond with anything but will still forward the QUERY packet to all his connected neighbors as long as the TTL value is greater than 0.

4) QUERY HIT packet received

The actions taken when a QUERY HIT packet is received are identical to those taken when a PONG packet is received (exempting replacing PONG with QUERY HIT).

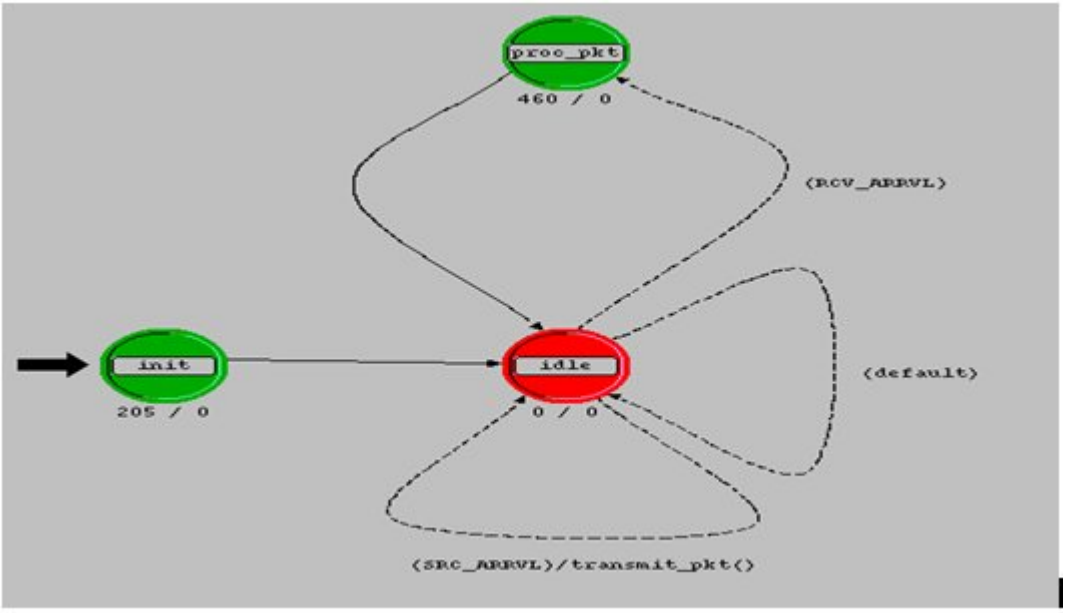

Figure 8: The process model that describes the behavior for each servent

Simulation Results

Simple Case

We came up with a simple simulation which was intended to confirm that our simulation matches our prediction. In our scenario, certain servents would query other servents for certain files, and if the servent does not have the requested file, they will forward the query onto his connected neighbors. Basically, each servent connects to each other through the servent closest to them.

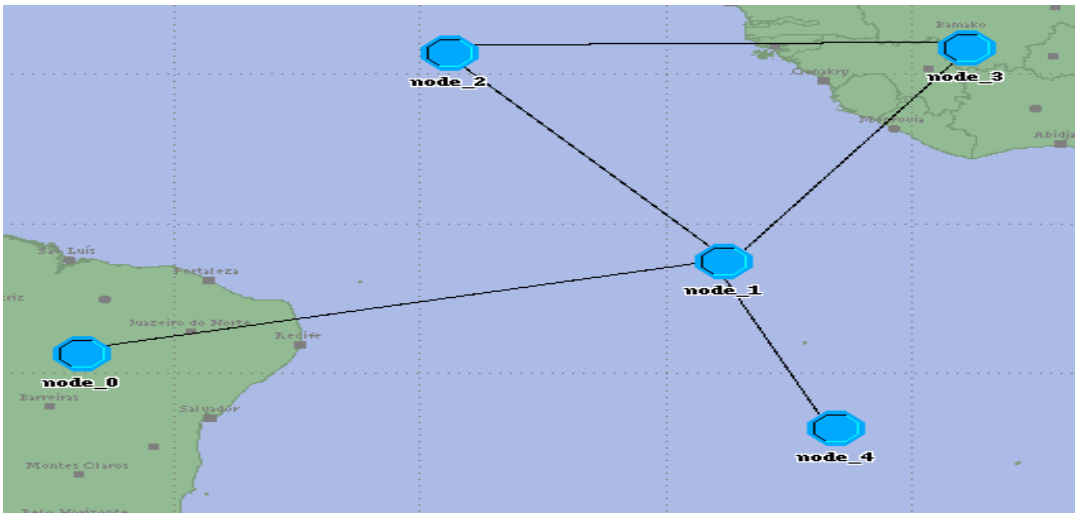

Figure 9: The network topology for the simple simulation

The parameters for this simple scenario are described in Table 3 below

Parameter	Value
Number of nodes	5
Initial TTL	3
Nodes generating traffic	0, 4
PING generation rate	1 PING every 60 seconds
QUERY generation rate	1 QUERY every 30 seconds
File database size	10
Files stored distribution	[1-100]
File search distribution	[1-4]

Table 3: Simple simulation parameters

The file distributions are random uniform integer distributions bounded by the limits shown in the table.

For this scenario each servent has a maximum of 10 files which it can store in its database. The 10 files which the servent will store are determined by the random integer distribution bounded by the limits [1-100]. In other words, this servent will choose to store 10 files randomly from a total possible 100 files. Similarly, the file that a node will search for when sending a QUERY is determined randomly by the uniform integer distribution bounded by the limits [1-4]. In essence this means, each servent generating a QUERY will search for 4 particular files randomly. Although this searching model can be seen to be unrealistic, it is suitable for testing the performance of the Gnutella network.

If we wish for a node to not generate any traffic, we simply set the starting time at infinity.

Ping/Pong Results

The diagram below shows the pings created (in blue) and pongs received (in red) for node 0. We obtained a final value for pings created of 130 and pongs received of 520. Notice that the pongs received is 4X the pings created in the end. The result matches our prediction since there are indeed 4 other nodes connected to the network and node 0 finds them.

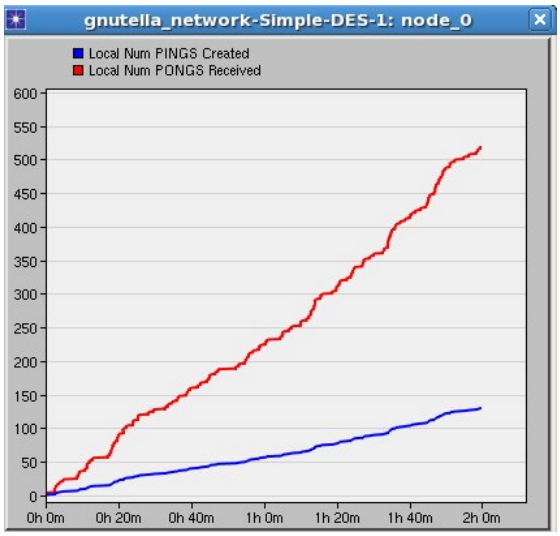

Figure 10: Ping/Pong data comparison

Queries/Query Hit Results

The diagram below shows the queries created and query hits received. In the end, 240 queries were requested by node 0 and 62 query hits were received back. Thus roughly about 25% of the queries were successful. This is a very good success ratio considering we are searching for 4 files from a possible 100 files.

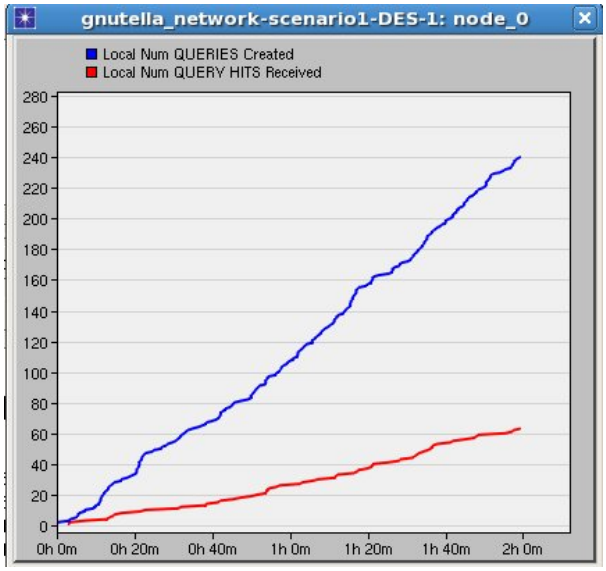

Figure 11: Queries/Query HITS comparison

Complex Case

In the next simulation, we have a much more complicated topology ready to be examined. This topology is shown in Figure 12 below. The parameters for this case are listed in Table 4 below

Parameter	Value
Number of nodes	14
Initial TTL	3, 7
Nodes generating traffic	0, 4, 10, 11
PING generation rate	1 PING every 60 seconds
QUERY generation rate	1 QUERY every 30 seconds
File database size	10
Files stored distribution	[0-500]
File search distribution	[1-10]

Table 4: Complex case simulation parameters

We will simulate this complex scenario using a TTL value of 3 and a TTL value of 7 to compare. One thing to note is that the File store distribution now contains a 0 indicating an invalid file. It should also be noted that in order to compare all 3 scenarios thus far, we also change the file distribution parameters for the simple case.

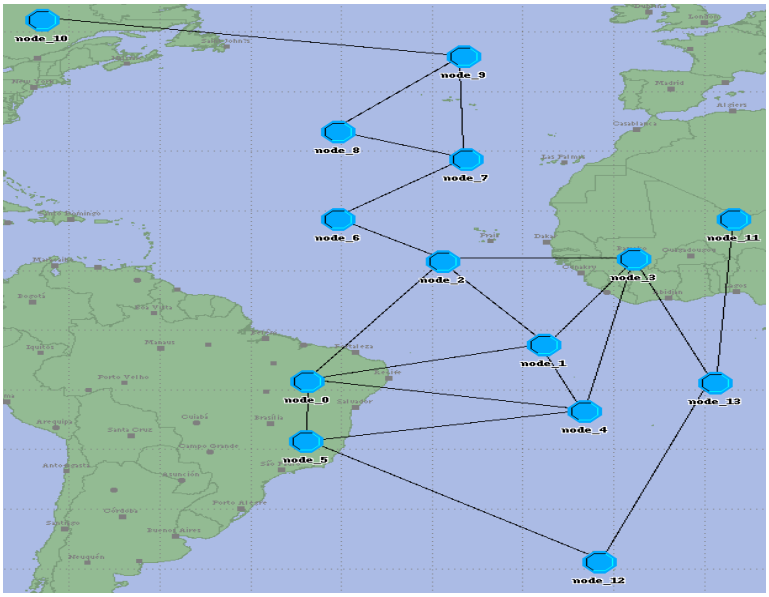

Figure 12: A complicated network topology for our Gnutella simulation

Ping/Pong Results

The results for the pings created and pongs received for node 0 for the 3 scenarios are shown in Figure 13 below and are summarized in Table 5.

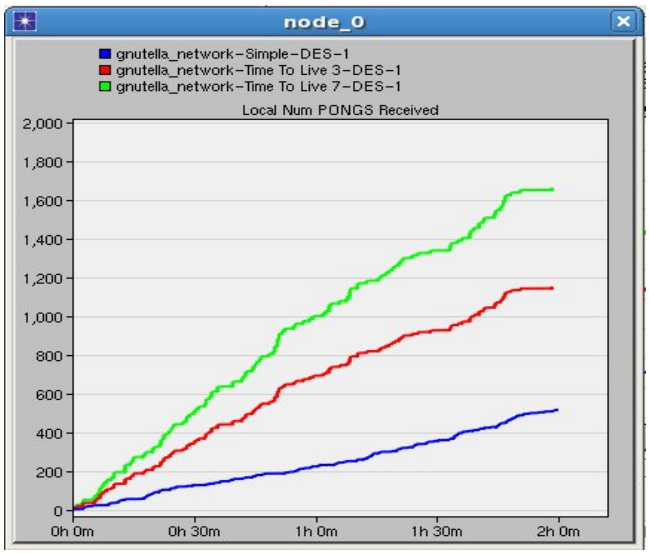

Figure 13: Pongs received by node 0

Scenario	**PONGS received (130 PINGS created in all cases)**
Simple	520 (4 nodes found)
Complex TTL = 3	1170 (9 nodes found)
Complex TTL = 7	1690 (13 nodes found)

Table 5: PONGS received by node 0

We can see a clear increase in the number of nodes discovered as the network gets more complex and as the TTL value increases.

Query and Queryhit Results

The results for the total number of queries created and query hits received for the 3 scenarios are shown in Figure 14 below and are summarized in Table 6

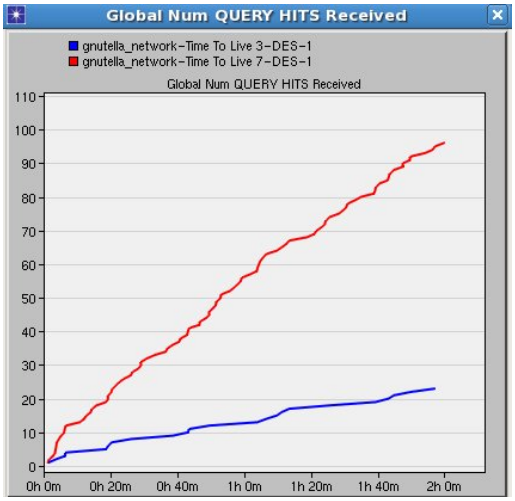

Figure 14: Query hits received

Scenario	QUERY HITS received (996 QUERY globally created in all cases)
Simple	0 (0%)
Complex TTL = 3	23 (2.3%)
Complex TTL = 7	96 (9.6%)

Table 6: QUERY HITS globally received

We can see a clear increase in the HITS received as the TTL value increases, as is to be expected. Although the query success ratio appears to be low, when we take a look at how relatively small our network is compared to a real-life network and the large file distribution we used for such a small network (500 files in total) we start to see that such low numbers are to be expected.

It is very difficult if not impossible to accurately model a realistic file searching scenario. Many factors have to be taken into account. For example, a very large network naturally has more probability of obtaining desired files than a small network. A popular file will obviously be easier to find than a rare one since the likelihood of someone having that file is high. Users' willfulness to share a file also plays a role. All these factors play a role in the success ratio of queries. Nevertheless we see in this simple simulation scenario the role of network size and TTL value on the query success ratio.

Link Utilization

Figure 15 below shows the link utilization for the 3 scenarios for the link between nodes 1 and 4

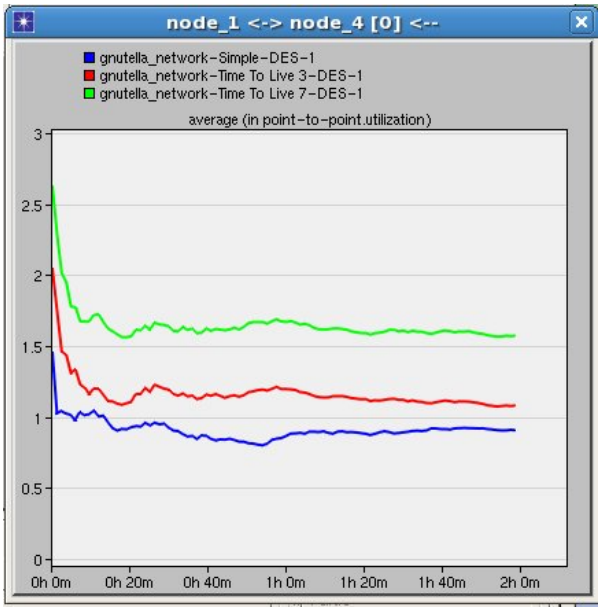

Figure 15: Link utilization

We can see that the simple scenario has the least link utilization and the complex scenario with a TTL value of 7 has highest link utilization. Thus we see that increasing TTL value has a cost on

bandwidth usage and we cannot increase it indefinitely as this would overwhelm the network and use up all the available bandwidth. In a real scenario, network size is much larger and there can be many servents generating traffic.

Figure 16 below shows the same link utilization for the complex case with TTL of 7 using 2 different traffic rates. The increased rate generated 1 PING every 5 seconds and 1 QUERY every second

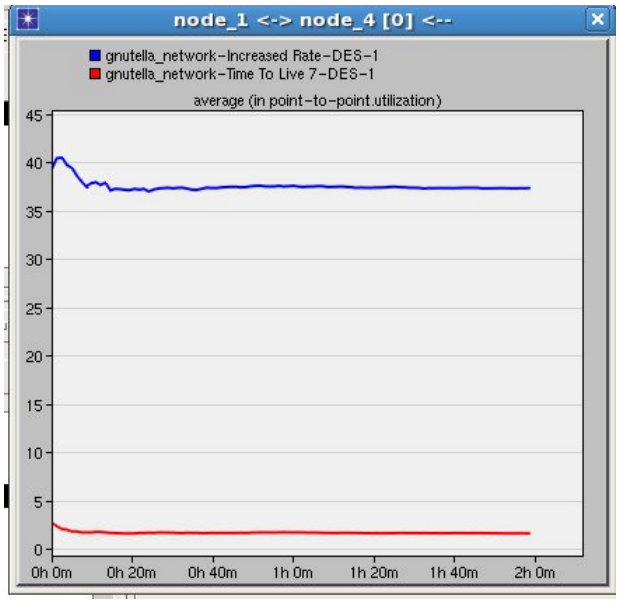

Figure 16: Comparison of link utilization with increased rate

Even with such high traffic rates, we see the link utilization stabilizes to a value below 50% and is acceptable.

Failed Links

Finally, in this section, we will simulate the effect node failures have on the performance of the network in terms of query hits received. The network topology is the same as in the complex case but with nodes 3 and 1 failed as shown in Figure 17 below.

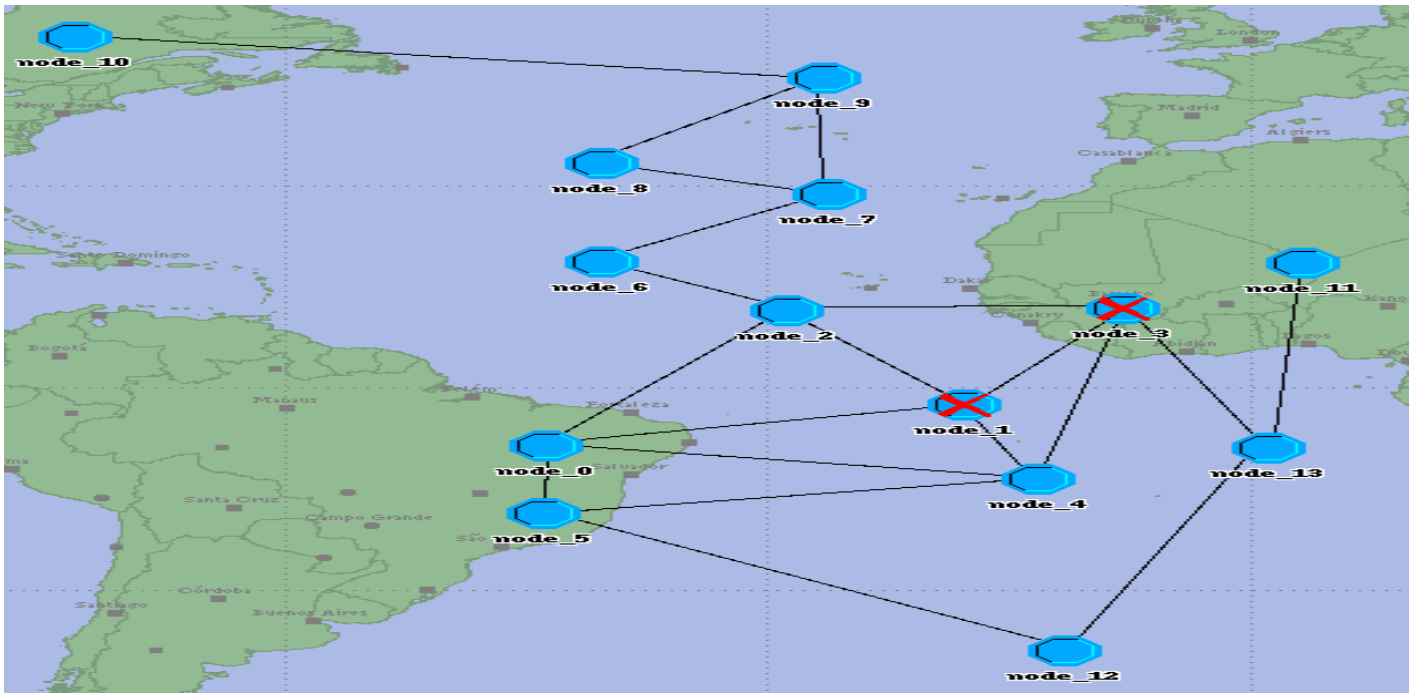

Figure 17: Network topology for complex case with failed nodes

The parameters for this simulation are shown in Table 7 below

Parameter	**Value**
Number of nodes	14
Initial TTL	7
Nodes generating traffic	0, 4, 10, 11
PING generation rate	1 PING every 60 seconds
QUERY generation rate	1 QUERY every 30 seconds
File database size	10
Files stored distribution	[0-250]
File search distribution	[1-10]

Table 7: Complex case failed node parameters

Figure 18 shows the number of query hits received with and without node failures

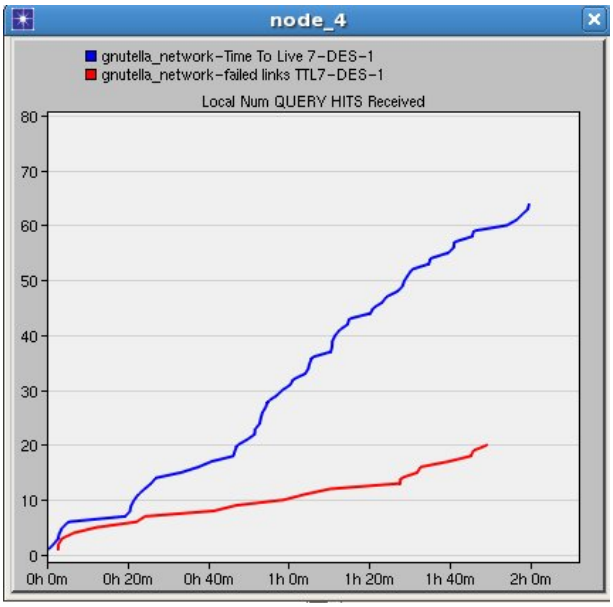

Figure 18: query hits received with and without node failure

From the figure, we can see a significant reduction in the query hits received. However, we are still at least receiving some query hits. In the traditional client server based model, should a critical node fail, the user would be completely unable to retrieve a desired file. The Gnutella network does not have this problem as can be seen from Figure 18.

Future work

Due to our lack of experience with OPNET and because of time constraints, the Gnutella model and simulations described in this project are fairly simple. Future work could involve modifying our model to make it more realistic. Some things that could be added are implementing the "PUSH" descriptor and simulating downloading files. In addition, our model uses a static unchanging network topology. It would be far more realistic to create a dynamic network model where servents enter and leave the network at certain times and see what effect this has on the network. In addition, as mentioned before, the Gnutella model we created was based on the Gnutella protocol V.0.4. It would be better to implement the more current Gnutella protocol V.0.6 which has more functionalities and features. Some of these new features include parallel downloading or swarming, where a user can download a file from multiple other servents to increase download speed. In addition, user ranking could be added to model the ranking system used by V.0.6 where a user is ranked based on how many files he shares and a higher ranked user is assigned a higher download speed.

Conclusion

To summarize, our team was able to successfully analyze the performance and architecture of the Gnutella network by simulating its characteristics in OPNET. The simulation graphs represented the data according to our expectations both in the simple and the complex cases. We saw the important role network size and TTL values play on the performance of the network and on bandwidth usage. It was seen that the larger the network and the larger the TTL values, the better the network performs but at the cost of more bandwidth usage. This bandwidth usage places a limit on how large we can increase TTL values and how many nodes we can connect to. We also saw the effect of nodes failing and demonstrated that node failures don't necessarily bring the whole network down which is a major advantage of Gnutella and peer-to-peer networks in general.

Gnutella being a decentralized network has been very useful in mass file sharing amongst users on the global scale. Its use has been growing drastically with millions of base users becoming part of this file sharing network each year. Its independence from a central server makes it a network with many advantages. For instance, it provides distributed processing with low traffic. It works almost all the time and is also very less vulnerable to major crash or failure since there is no individual component on which its functionality relies. The transfer data is available under most of the firewall systems and is very flexible in query processing. With least maintenance costs compared to other centralized networks, it simply provides easy access to both hosts and clients. Although, Gnutella has many advantages, there are some issues that could not be neglected and need considerable attention. Since the network system is not monitored under a specific company, it does not provide quality service due to overloading and downloading failure. Most often, some files are not available for the users due to the unwillingness of the peers to share the files. It also lacks new features and services due to lack of update and maintenance support. Overall, the network's advantages outnumber its disadvantages which are minor and could be very easily resolved. For instance the issue of users' reluctance to share files can easily be resolved by introducing rating schemes amongst the registered users. Overall, Gnutella is a very powerful file sharing tool and has the capability to change the structure of the internet from web centric to purely distributed data model.

References

[1] “How Gnutella Works” Internet: http://computer.howstuffworks.com/file-sharing1.htm [Mar. 23, 2013]

[2] “The Annotated Gnutella Protocol Specification v0.4 (1)” Internet: http://rfc-gnutella.sourceforge.net/developer/stable/ [Mar. 23, 2013]

[3] “Gnutella” Internet: http://en.wikipedia.org/wiki/Gnutella [Mar. 20, 2013]

[4] Fabrizio Cornelli, Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, Pierangela Samarati. “Choosing Reputable Servents in a P2P Network” Internet: http://www2002.org/CDROM/refereed/68/ [Mar. 22, 2013]

[5] Gene Kan. “Gnutella” in *Peer-To-Peer: Harnessing the Benefits of Disruptive Technology*, 1st ed., Andy Oram, Ed. Sebastopol, CA: O’Reilly & Associates, Inc., 2001, pp. 94-122

[6] H.Su and K.Wu. “Robustness of Gnutella network” Internet: http://www.ensc.sfu.ca/~ljilja/ENSC427/Spring09/Projects/team12/Gnutella_Network_Robustness_Final_Version.pdf [Mar. 23, 2013]

[7] “Implementation of the Gnutella Protocol” Internet: http://www2.ensc.sfu.ca/~ljilja/ENSC427/Spring10/Projects/team7/final_report_Gnutella.pdf [Mar. 30, 2013]

APPENDIX:

HEADER FILE

```
//*****************MAKE SURE LINKS ARE CONNECTED APPROPIATLEY IN NODE
MODEL*****************************************//
//Packets received from packet generator have input stream 4
//Packets received from other nodes have input streams 0-3
//Packet sending to other nodes have output streams 0-3

//Input Streams
#define SRC_IN_STRM_PING 4
#define SRC_IN_STRM_QUERY 5
#define RCV_IN_STRM1 0
#define RCV_IN_STRM2 1
#define RCV_IN_STRM3 2
#define RCV_IN_STRM4 3

//Output Streams
#define XMT_OUT_STRM1 0
#define XMT_OUT_STRM2 1
#define XMT_OUT_STRM3 2
#define XMT_OUT_STRM4 3

//Packet arriving from generator
#define SRC_ARRVL (op_intrpt_type () == OPC_INTRPT_STRM && \
((op_intrpt_strm () == SRC_IN_STRM_PING) ||        (op_intrpt_strm () ==
SRC_IN_STRM_QUERY)))

//Packet arriving from another node
#define RCV_ARRVL (op_intrpt_type () == OPC_INTRPT_STRM && \
((op_intrpt_strm () == RCV_IN_STRM1) || (op_intrpt_strm () == RCV_IN_STRM2) ||
(op_intrpt_strm () == RCV_IN_STRM3) || (op_intrpt_strm () == RCV_IN_STRM4)))

//Define descriptors PING/PONG/QUERY/QUERY_HIT
#define PING 0
#define PONG 1
#define QUERY 128
#define QUERY_HIT 129

//We'll need  this to store our statistics at the end of the simulation
// I'm not using this right now. Might be uneccessary
//#define END_SIM (op_intrpt_type () == OPC_INTRPT_ENDSIM)


//accumulators
int num_PINGS_G_C;
int num_PINGS_G_R;
int num_PONGS_G_C;
int num_PONGS_G_R;
int num_QUERIES_G_C;
int num_QUERIES_G_R;
```

```
int num_QUERY_HITS_G_C;
int num_QUERY_HITS_G_R;
```

init Enter Executive

```
//node_ID will be a unique ID assigned to each node in our network
//it is defined in state variables
Objid node_object_ID;
node_object_ID = op_topo_parent(op_id_self());
op_ima_obj_attr_get_int32 (node_object_ID, "user id",&node_ID);

file_search_dist = op_dist_load ("uniform_int", 1, 10);

files_stored_dist = op_dist_load("uniform_int", 0, 250); //The 0 indicates that
there is no file


//Set which files a node has at start time
file1 = (int)op_dist_outcome(files_stored_dist);
file2 = (int)op_dist_outcome(files_stored_dist);
file3 = (int)op_dist_outcome(files_stored_dist);
file4 = (int)op_dist_outcome(files_stored_dist);
file5 = (int)op_dist_outcome(files_stored_dist);
file6 = (int)op_dist_outcome(files_stored_dist);
file7 = (int)op_dist_outcome(files_stored_dist);
file8 = (int)op_dist_outcome(files_stored_dist);
file9 = (int)op_dist_outcome(files_stored_dist);
file10 = (int)op_dist_outcome(files_stored_dist);


//Initialize accumulators
num_PINGS_G_C = 0;
num_PINGS_G_R = 0;
num_PONGS_G_C = 0;
num_PONGS_G_R = 0;
num_QUERIES_G_C = 0;
num_QUERIES_G_R = 0;
num_QUERY_HITS_G_C = 0;
num_QUERY_HITS_G_R = 0;
num_PINGS_L_C = 0;
num_PINGS_L_R = 0;
num_PONGS_L_C = 0;
num_PONGS_L_R = 0;
num_QUERIES_L_C = 0;
num_QUERIES_L_R = 0;
num_QUERY_HITS_L_C = 0;
num_QUERY_HITS_L_R = 0;
PING_cache1 = -1;
PING_cache2 = -1;
PING_cache3 = -1;
PING_cache4 = -1;
PING_cache5 = -1;
```

```
PING_cache6 = -1;
PING_cache7 = -1;
PING_cache8 = -1;
PING_cache9 = -1;
PING_cache10 = -1;
QUERY_cache1 = -1;
QUERY_cache2 = -1;
QUERY_cache3 = -1;
QUERY_cache4 = -1;
QUERY_cache5 = -1;
QUERY_cache6 = -1;
QUERY_cache7 = -1;
QUERY_cache8 = -1;
QUERY_cache9 = -1;
QUERY_cache10 = -1;
//Register statistics we are going to use
num_PINGS_G_C_h = op_stat_reg ("Global Num PINGS Created",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
num_PINGS_G_R_h = op_stat_reg ("Global Num PINGS Received",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
num_PONGS_G_C_h = op_stat_reg ("Global Num PONGS Created",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
num_PONGS_G_R_h = op_stat_reg ("Global Num PONGS Received",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
num_QUERIES_G_C_h = op_stat_reg ("Global Num QUERIES Created",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
num_QUERIES_G_R_h = op_stat_reg ("Global Num QUERIES Received",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
num_QUERY_HITS_G_C_h = op_stat_reg ("Global Num QUERY HITS Created",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
num_QUERY_HITS_G_R_h = op_stat_reg ("Global Num QUERY HITS Received",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
num_PINGS_L_C_h = op_stat_reg ("Local Num PINGS Created",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
num_PINGS_L_R_h = op_stat_reg ("Local Num PINGS Received",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
num_PONGS_L_C_h = op_stat_reg ("Local Num PONGS Created",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
num_PONGS_L_R_h = op_stat_reg ("Local Num PONGS Received",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
num_QUERIES_L_C_h = op_stat_reg ("Local Num QUERIES Created",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
num_QUERIES_L_R_h = op_stat_reg ("Local Num QUERIES Received",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
num_QUERY_HITS_L_C_h = op_stat_reg ("Local Num QUERY HITS Created",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
num_QUERY_HITS_L_R_h = op_stat_reg ("Local Num QUERY HITS Received",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);



//Routing Tables
//The packets are actually supposed to be sent back through the same route they
```

```
came
//but this way is simpler and is actually a better routing algorithm. Either
way, the purpose of this project
//is not to compare routing algorithms.
if (node_ID == 0)
        {
                node_4 = XMT_OUT_STRM2;
                node_10 = XMT_OUT_STRM3;
                node_11 = XMT_OUT_STRM4;
        }
else if (node_ID == 1)
        {
                node_0 = XMT_OUT_STRM1;
                node_4 = XMT_OUT_STRM4;
                node_10 = XMT_OUT_STRM2;
                node_11 = XMT_OUT_STRM3;
        }
else if (node_ID == 2)
        {
                node_0 = XMT_OUT_STRM3;
                node_4 = XMT_OUT_STRM1;
                node_10 = XMT_OUT_STRM4;
                node_11 = XMT_OUT_STRM2;
        }
else if (node_ID == 3)
        {
                node_0 = XMT_OUT_STRM1;
                node_4 = XMT_OUT_STRM3;
                node_10 = XMT_OUT_STRM2;
                node_11 = XMT_OUT_STRM4;
        }
else if (node_ID == 4)
        {
                node_0 = XMT_OUT_STRM2;
                node_10 = XMT_OUT_STRM1;
                node_11 = XMT_OUT_STRM3;
        }
else if (node_ID == 5)
        {
                node_0 = XMT_OUT_STRM1;
                node_4 = XMT_OUT_STRM2;
                node_10 = XMT_OUT_STRM1;
                node_11 = XMT_OUT_STRM3;
        }
else if (node_ID == 6)
        {
                node_0 = XMT_OUT_STRM1;
                node_4 = XMT_OUT_STRM1;
                node_10 = XMT_OUT_STRM2;
                node_11 = XMT_OUT_STRM1;
        }
else if (node_ID == 7)
        {
                node_0 = XMT_OUT_STRM1;
```

```
                node_4 = XMT_OUT_STRM1;
                node_10 = XMT_OUT_STRM3;
                node_11 = XMT_OUT_STRM1;
        }
else if (node_ID == 8)
        {
                node_0 = XMT_OUT_STRM1;
                node_4 = XMT_OUT_STRM1;
                node_10 = XMT_OUT_STRM2;
                node_11 = XMT_OUT_STRM1;
        }
else if (node_ID == 9)
        {
                node_0 = XMT_OUT_STRM1;
                node_4 = XMT_OUT_STRM3;
                node_10 = XMT_OUT_STRM2;
                node_11 = XMT_OUT_STRM3;
        }
else if (node_ID == 10)
        {
                node_0 = XMT_OUT_STRM1;
                node_4 = XMT_OUT_STRM1;
                node_11 = XMT_OUT_STRM1;
        }
else if (node_ID == 11)
        {
                node_0 = XMT_OUT_STRM1;
                node_4 = XMT_OUT_STRM1;
                node_10 = XMT_OUT_STRM1;
        }
else if (node_ID == 12)
        {
                node_0 = XMT_OUT_STRM2;
                node_4 = XMT_OUT_STRM2;
                node_10 = XMT_OUT_STRM1;
                node_11 = XMT_OUT_STRM1;
        }
else if (node_ID == 13)
        {
                node_0 = XMT_OUT_STRM3;
                node_4 = XMT_OUT_STRM3;
                node_10 = XMT_OUT_STRM1;
                node_11 = XMT_OUT_STRM2;
        }
```

transmit_pkt() function

```
static void transmit_pkt(void)
{
//Defines packet pointer variables of type "Packet"
Packet * pkptr1;
Packet * pkptr2;
Packet * pkptr3;
Packet * pkptr4;

int src_strm; //Determines if it's a PING or QUERY packet

//Packet will be sent out to 4 other nodes
//We are assuming each node is connected to 4 other nodes
//We can modify the process model to change this for a specific node if we want
maybe. See how simulation performs

OpT_Packet_Id pk_ID;

//You can play around with this value if you want
int TTL = 3;

int file_to_search_for;

FIN(transmit_pkt());

//Determine if this is a PING or QUERY packet
src_strm = op_intrpt_strm ();

//Get the packet pointer
pkptr1 = op_pk_get (src_strm);
//Get the packet I.D.
pk_ID = op_pk_id(pkptr1);


//PING//PING//PING//PING//PING//PING//PING//PING//PING//PING//PING//PING//PING/
/PING//PING//PING//PING//PING//PING//PING//PING//PING//PING//PING//PING//PING//
PING
if (src_strm == SRC_IN_STRM_PING)
{        //This is a PING packet
        //SET APPROPIATE FIELD VALUES IN THE SECTION BELOW
        //Define these formally later on

op_pk_nfd_set_int32 (pkptr1, "Payload Descriptor", PING);
op_pk_nfd_set_int32 (pkptr1, "TTL", TTL);
op_pk_nfd_set_int32 (pkptr1, "HOPS",0);                //Hops should always be
initialized to 0
op_pk_nfd_set_int32 (pkptr1, "Source Address",node_ID);
op_pk_nfd_set_int32 (pkptr1, "File Name",0); //We're not looking for any files
op_pk_nfd_set_int32 (pkptr1, "Packet ID",pk_ID);

++num_PINGS_G_C;
++num_PINGS_L_C;
```

```
op_stat_write (num_PINGS_G_C_h, num_PINGS_G_C);
op_stat_write (num_PINGS_L_C_h, num_PINGS_L_C);


//COPY THE PACKET 3 TIMES IN THE SECTION BELOW
pkptr2 = op_pk_create_fmt("gnutella_packet");
//Define these formally later on
op_pk_nfd_set_int32 (pkptr2, "Payload Descriptor", PING);
op_pk_nfd_set_int32 (pkptr2, "TTL", TTL);
op_pk_nfd_set_int32 (pkptr2, "HOPS",0);
op_pk_nfd_set_int32 (pkptr2, "Source Address",node_ID);
op_pk_nfd_set_int32 (pkptr2, "File Name",0);
op_pk_nfd_set_int32 (pkptr2, "Packet ID",pk_ID);


pkptr3 = op_pk_create_fmt("gnutella_packet");
//Define these formally later on
op_pk_nfd_set_int32 (pkptr3, "Payload Descriptor", PING);
op_pk_nfd_set_int32 (pkptr3, "TTL", TTL);
op_pk_nfd_set_int32 (pkptr3, "HOPS",0);
op_pk_nfd_set_int32 (pkptr3, "Source Address",node_ID);
op_pk_nfd_set_int32 (pkptr3, "File Name",0);
op_pk_nfd_set_int32 (pkptr3, "Packet ID",pk_ID);


pkptr4 = op_pk_create_fmt("gnutella_packet");
//Define these formally later on
op_pk_nfd_set_int32 (pkptr4, "Payload Descriptor", PING);
op_pk_nfd_set_int32 (pkptr4, "TTL", TTL);
op_pk_nfd_set_int32 (pkptr4, "HOPS",0);
op_pk_nfd_set_int32 (pkptr4, "Source Address",node_ID);
op_pk_nfd_set_int32 (pkptr4, "File Name",0);
op_pk_nfd_set_int32 (pkptr4, "Packet ID",pk_ID);



//Send packet to the other nodes you are connected to
op_pk_send(pkptr1, XMT_OUT_STRM1);
op_pk_send(pkptr2, XMT_OUT_STRM2);
op_pk_send(pkptr3, XMT_OUT_STRM3);
op_pk_send(pkptr4, XMT_OUT_STRM4);
}
//PING//PING//PING//PING//PING//PING//PING//PING//PING//PING//PING//PING//PING/
/PING//PING//PING//PING//PING//PING//PING//PING//PING//PING//PING//PING//PING//
PING

//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//
QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QU
ERY
else
{
        //This is a QUERY packet
        //SET APPROPIATE FIELD VALUES IN THE SECTION BELOW
        //Define these formally later on
```

```
        file_to_search_for = (int)op_dist_outcome(file_search_dist);

op_pk_nfd_set_int32 (pkptr1, "Payload Descriptor", QUERY);
op_pk_nfd_set_int32 (pkptr1, "TTL", TTL);
op_pk_nfd_set_int32 (pkptr1, "HOPS",0);
op_pk_nfd_set_int32 (pkptr1, "Source Address",node_ID);
op_pk_nfd_set_int32 (pkptr1, "File Name", file_to_search_for); //Set random
file name to search for
op_pk_nfd_set_int32 (pkptr1, "Packet ID",pk_ID);

++num_QUERIES_G_C;
++num_QUERIES_L_C;
op_stat_write (num_QUERIES_G_C_h, num_QUERIES_G_C);
op_stat_write (num_QUERIES_L_C_h, num_QUERIES_L_C);

//COPY THE PACKET 3 TIMES IN THE SECTION BELOW
pkptr2 = op_pk_create_fmt("gnutella_packet");
//Define these formally later on
op_pk_nfd_set_int32 (pkptr2, "Payload Descriptor", QUERY);
op_pk_nfd_set_int32 (pkptr2, "TTL", TTL);
op_pk_nfd_set_int32 (pkptr2, "HOPS",0);
op_pk_nfd_set_int32 (pkptr2, "Source Address",node_ID);
op_pk_nfd_set_int32 (pkptr2, "File Name", file_to_search_for);
op_pk_nfd_set_int32 (pkptr2, "Packet ID",pk_ID);


pkptr3 = op_pk_create_fmt("gnutella_packet");
//Define these formally later on
op_pk_nfd_set_int32 (pkptr3, "Payload Descriptor", QUERY);
op_pk_nfd_set_int32 (pkptr3, "TTL", TTL);
op_pk_nfd_set_int32 (pkptr3, "HOPS",0);
op_pk_nfd_set_int32 (pkptr3, "Source Address",node_ID);
op_pk_nfd_set_int32 (pkptr3, "File Name", file_to_search_for);
op_pk_nfd_set_int32 (pkptr3, "Packet ID",pk_ID);


pkptr4 = op_pk_create_fmt("gnutella_packet");
//Define these formally later on
op_pk_nfd_set_int32 (pkptr4, "Payload Descriptor", QUERY);
op_pk_nfd_set_int32 (pkptr4, "TTL", TTL);
op_pk_nfd_set_int32 (pkptr4, "HOPS",0);
op_pk_nfd_set_int32 (pkptr4, "Source Address",node_ID);
op_pk_nfd_set_int32 (pkptr4, "File Name", file_to_search_for);
op_pk_nfd_set_int32 (pkptr4, "Packet ID",pk_ID);


//Send packet to the other nodes you are connected to
op_pk_send(pkptr1, XMT_OUT_STRM1);
op_pk_send(pkptr2, XMT_OUT_STRM2);
op_pk_send(pkptr3, XMT_OUT_STRM3);
op_pk_send(pkptr4, XMT_OUT_STRM4);
}
//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//
QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QU
```

```
ERY

FOUT;
}

//I might make use of this function in the future

//This function writes the end-of-simulation
//statistics to a vector file

//static void record_stats (void)
//{
//FIN (record_stats());
//Record final statistics
//op_stat_scalar_write ("Number of PINGS Generated", num_PINGS_G);
//FOUT;
//}
```

proc_pkt Enter Executive

```
//This state will read the packet that has arrived and take appropiate actions
//Forward PING
//Forward QUERY
//Respond with PONG
//Check to see if it has requested file
//Respond with QUERY_HIT if it has requested file
//Destroy packet if TTL = 0 after decrementing it

int PD;                                 //Payload Descriptor
int TTL;                          //Time To Live
int HOPS;                          //Hops
int src_address;        //Determines where the packet came from. Used for
routing purposes
int FILE;                          //File we're searching for
int pk_ID;                          //Packet ID used for detecting duplicate
packets


//int file_found = 0;//Used to generate QUERY_HIT


Packet * pkptr1;
Packet * pkptr2;
Packet * pkptr3;
Packet * pkptr4;        //This is used for the PONG/QUERY_HIT packets for now

//Indicates which node the packet came from
int receive_strm;

receive_strm = op_intrpt_strm ();
pkptr1 = op_pk_get(receive_strm);
```

```
//Read what type of packet this is
op_pk_nfd_get_int32 (pkptr1, "Payload Descriptor", &PD);

//Read packet header
op_pk_nfd_get_int32 (pkptr1, "TTL", &TTL);
op_pk_nfd_get_int32 (pkptr1, "HOPS", &HOPS);
op_pk_nfd_get_int32 (pkptr1, "Source Address", &src_address);
op_pk_nfd_get_int32 (pkptr1, "File Name", &FILE);
op_pk_nfd_get_int32 (pkptr1, "Packet ID", &pk_ID);

//Update TTL and HOPS fields
--TTL;
++HOPS;

///PING///PING///PING///PING///PING///PING///PING///PING///PING///PING///PING//
/PING///PING///PING///PING///PING///PING///PING///PING///PING///PING///PING
if (PD == PING)
        {
                //This is a PING packet
                //Check if it's a duplicate
                if ((PING_cache1 == pk_ID) || (PING_cache2 == pk_ID) ||
(PING_cache3 == pk_ID) || (PING_cache4 == pk_ID) || (PING_cache5 == pk_ID) ||
(PING_cache6 == pk_ID) || (PING_cache7 == pk_ID) || (PING_cache8 == pk_ID) ||
(PING_cache9 == pk_ID) || (PING_cache10 == pk_ID) || (src_address == node_ID))
                        {
                                //Destroy and no further action required.
                                op_pk_destroy (pkptr1);


                        }

                else
                {

                //cache packet
                if ((num_PINGS_L_R % 10) == 0)
                        {
                        PING_cache1 = pk_ID;
                        }
                else if ((num_PINGS_L_R % 10) == 1)
                        {
                        PING_cache2 = pk_ID;
                        }
                else if ((num_PINGS_L_R % 10) == 2)
                        {
                        PING_cache3 = pk_ID;
                        }
                else if ((num_PINGS_L_R % 10) == 3)
                        {
                        PING_cache4 = pk_ID;
                        }
                else if ((num_PINGS_L_R % 10) == 4)
                        {
                        PING_cache5 = pk_ID;
```

```
                }
        else if ((num_PINGS_L_R % 10) == 5)
                {
                PING_cache6 = pk_ID;
                }
        else if ((num_PINGS_L_R % 10) == 6)
                {
                PING_cache7 = pk_ID;
                }
        else if ((num_PINGS_L_R % 10) == 7)
                {
                PING_cache8 = pk_ID;
                }
        else if ((num_PINGS_L_R % 10) == 8)
                {
                PING_cache9 = pk_ID;
                }
        else if ((num_PINGS_L_R % 10) == 9)
                {
                PING_cache10 = pk_ID;
                }



        ++num_PINGS_G_R;
        ++num_PINGS_L_R;
        op_stat_write (num_PINGS_G_R_h, num_PINGS_G_R);
        op_stat_write (num_PINGS_L_R_h, num_PINGS_L_R);


        //Respond with a PONG packet to where the packet came from
        pkptr4 = op_pk_create_fmt("gnutella_packet");
        op_pk_nfd_set_int32 (pkptr4, "Payload Descriptor", PONG);
        op_pk_nfd_set_int32 (pkptr4, "TTL", TTL + HOPS);
        op_pk_nfd_set_int32 (pkptr4, "HOPS",0);
        op_pk_nfd_set_int32 (pkptr4, "Source Address",src_address);
        op_pk_nfd_set_int32 (pkptr4, "File Name",0);
        op_pk_nfd_set_int32 (pkptr4, "Packet ID", 0); //We won't check
packet ID if it's a response. We can assume only one response will be sent
        op_pk_send (pkptr4, receive_strm);

        ++num_PONGS_G_C;
        ++num_PONGS_L_C;
        op_stat_write (num_PONGS_G_C_h, num_PONGS_G_C);
        op_stat_write (num_PONGS_L_C_h, num_PONGS_L_C);


        //Destroy PING packet if TTL has reached 0
        if (TTL  == 0)
                {
                        op_pk_destroy (pkptr1);
                }
        else
                {
```

```
		//Forward the packet to all the neighbours
				//Update TTL and HOPS field
				//We actually only need to update TTL and HOPS
fields here. Nothing else
				op_pk_nfd_set_int32 (pkptr1, "Payload
Descriptor", PING);
				op_pk_nfd_set_int32 (pkptr1, "TTL", TTL);
				op_pk_nfd_set_int32 (pkptr1, "HOPS",HOPS);
				op_pk_nfd_set_int32 (pkptr1, "Source
Address",src_address);
				op_pk_nfd_set_int32 (pkptr1, "File Name",0);
				op_pk_nfd_set_int32 (pkptr1, "Packet ID",
pk_ID);

				//Copy packet 2 more times and send to
appropiate nodes
				pkptr2 = op_pk_create_fmt("gnutella_packet");
				op_pk_nfd_set_int32 (pkptr2, "Payload
Descriptor", PING);
				op_pk_nfd_set_int32 (pkptr2, "TTL", TTL);
				op_pk_nfd_set_int32 (pkptr2, "HOPS",HOPS);
				op_pk_nfd_set_int32 (pkptr2, "Source
Address",src_address);
				op_pk_nfd_set_int32 (pkptr2, "File Name",0);
				op_pk_nfd_set_int32 (pkptr2, "Packet ID",
pk_ID);


				pkptr3 = op_pk_create_fmt("gnutella_packet");
				op_pk_nfd_set_int32 (pkptr3, "Payload
Descriptor", PING);
				op_pk_nfd_set_int32 (pkptr3, "TTL", TTL);
				op_pk_nfd_set_int32 (pkptr3, "HOPS",HOPS);
				op_pk_nfd_set_int32 (pkptr3, "Source
Address",src_address);
				op_pk_nfd_set_int32 (pkptr3, "File Name",0);
				op_pk_nfd_set_int32 (pkptr3, "Packet ID",
pk_ID);


				//Send them out to neighbouring nodes
				if (receive_strm == RCV_IN_STRM1)
					{
						op_pk_send (pkptr1,
XMT_OUT_STRM2);
						op_pk_send (pkptr2,
XMT_OUT_STRM3);
						op_pk_send (pkptr3,
XMT_OUT_STRM4);
					}
				else if (receive_strm == RCV_IN_STRM2)
					{
						op_pk_send (pkptr1,
XMT_OUT_STRM1);
```

```
                                        op_pk_send (pkptr2,
XMT_OUT_STRM3);
                                        op_pk_send (pkptr3,
XMT_OUT_STRM4);
                                }
                        else if (receive_strm == RCV_IN_STRM3)
                                {
                                        op_pk_send (pkptr1,
XMT_OUT_STRM1);
                                        op_pk_send (pkptr2,
XMT_OUT_STRM2);
                                        op_pk_send (pkptr3,
XMT_OUT_STRM4);
                                }
                        else if (receive_strm == RCV_IN_STRM4)
                                {
                                        op_pk_send (pkptr1,
XMT_OUT_STRM1);
                                        op_pk_send (pkptr2,
XMT_OUT_STRM2);
                                        op_pk_send (pkptr3,
XMT_OUT_STRM3);
                                }

                        }
                }
        }
///PING///PING///PING///PING///PING///PING///PING///PING///PING///PING///PING//
/PING///PING///PING///PING///PING///PING///PING///PING///PING///PING///PING

//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG/
/PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG
else if (PD == PONG)
        {
        //This is a PONG packet
        if (src_address == node_ID)
                {
                        //Destroy if packet was destined to this node
                        op_pk_destroy (pkptr1);
                        ++num_PONGS_G_R;
                        ++num_PONGS_L_R;
                        op_stat_write (num_PONGS_G_R_h, num_PONGS_G_R);
                        op_stat_write (num_PONGS_L_R_h, num_PONGS_L_R);
                        //Want to count number of received PONGS
                }
        else if (TTL == 0)
                {
                        //destroy packet and take no further action
                        op_pk_destroy (pkptr1);
                }

        else
                        //Forward PONG packet to appropiate route where packet
came from
```

```
                {
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
                        if (src_address == 0)
                                {
                                        op_pk_send (pkptr1, node_0);
                                }
                        else if (src_address == 4)
                                {
                                        op_pk_send (pkptr1, node_4);
                                }
                        else if (src_address == 10)
                                {
                                        op_pk_send (pkptr1, node_10);
                                }
                        else if (src_address == 11)
                                {
                                        op_pk_send (pkptr1, node_11);
                                }
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
                }
        }


//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG/
/PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG//PONG


//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//
QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QU
ERY
```

```
else if (PD == QUERY)
        {
                //This is a QUERY packet


                if ((QUERY_cache1 == pk_ID) || (QUERY_cache2 == pk_ID) ||
(QUERY_cache3 == pk_ID) || (QUERY_cache4 == pk_ID) || (QUERY_cache5 == pk_ID)
|| (QUERY_cache6 == pk_ID) || (QUERY_cache7 == pk_ID) || (QUERY_cache8 ==
pk_ID) || (QUERY_cache9 == pk_ID) || (QUERY_cache10 == pk_ID) || (src_address
== node_ID))
                        {
                                //Destroy and no further action required.
                                op_pk_destroy (pkptr1);


                        }

                else
                {

                //THIS PART IS ROYALLY MESSING UP FOR SOME REASON AND I CANT
FIGURE OUT WHY
                //cache packet
                if ((num_QUERIES_L_R % 10) == 0)
                        {
                        QUERY_cache1 = pk_ID;
                        }
                else if ((num_QUERIES_L_R % 10) == 1)
                        {
                        QUERY_cache2 = pk_ID;
                        }
                else if ((num_QUERIES_L_R % 10) == 2)
                        {
                        QUERY_cache3 = pk_ID;
                        }
                else if ((num_QUERIES_L_R % 10) == 3)
                        {
                        QUERY_cache4 = pk_ID;
                        }
                else if ((num_QUERIES_L_R % 10) == 4)
                        {
                        QUERY_cache5 = pk_ID;
                        }
                else if ((num_QUERIES_L_R % 10) == 5)
                        {
                        QUERY_cache6 = pk_ID;
                        }
                else if ((num_QUERIES_L_R % 10) == 6)
                        {
                        QUERY_cache7 = pk_ID;
                        }
                else if ((num_QUERIES_L_R % 10) == 7)
                        {
                        QUERY_cache8 = pk_ID;
```

```
                }
        else if ((num_QUERIES_L_R % 10) == 8)
                {
                QUERY_cache9 = pk_ID;
                }
        else if ((num_QUERIES_L_R % 10) == 9)
                {
                QUERY_cache10 = pk_ID;
                }


        ++num_QUERIES_G_R;
        ++num_QUERIES_L_R;
        op_stat_write (num_QUERIES_G_R_h, num_QUERIES_G_R);
        op_stat_write (num_QUERIES_L_R_h, num_QUERIES_L_R);


        //Check file name with data base and send QUERY_HIT if file
found
        if ((FILE == file1) || (FILE == file2) || (FILE == file3) ||
(FILE == file4) || (FILE == file5) || (FILE == file6) || (FILE == file7) ||
(FILE == file8) || (FILE == file9) || (FILE == file10))

                {
                        //reply with a QUERY_HIT
                        pkptr4 = op_pk_create_fmt("gnutella_packet");
                        op_pk_nfd_set_int32 (pkptr4, "Payload
Descriptor", QUERY_HIT);
                        op_pk_nfd_set_int32 (pkptr4, "TTL", TTL +
HOPS);
                        op_pk_nfd_set_int32 (pkptr4, "HOPS",0);
                        op_pk_nfd_set_int32 (pkptr4, "Source
Address",src_address);
                        op_pk_nfd_set_int32 (pkptr4, "File Name",0);
                        op_pk_nfd_set_int32 (pkptr4, "Packet ID", 0);
//We won't check packet ID if it's a response. We can assume only one response
will be sent
                        op_pk_send (pkptr4, receive_strm);

                        ++num_QUERY_HITS_G_C;
                        ++num_QUERY_HITS_L_C;
                        op_stat_write (num_QUERY_HITS_G_C_h,
num_QUERY_HITS_G_C);
                        op_stat_write (num_QUERY_HITS_L_C_h,
num_QUERY_HITS_L_C);
                }



        //Destroy QUERY packet if TTL has reached 0

        if (TTL == 0)
                {
                        //destroy packet
```

```
                        op_pk_destroy (pkptr1);
                }
        else
                {
                        //Forward the packet to all the neighbours
                        //Update TTL and HOPS field
                        op_pk_nfd_set_int32 (pkptr1, "Payload
Descriptor", QUERY);
                        op_pk_nfd_set_int32 (pkptr1, "TTL", TTL);
                        op_pk_nfd_set_int32 (pkptr1, "HOPS",HOPS);
                        op_pk_nfd_set_int32 (pkptr1, "Source
Address",src_address);
                        op_pk_nfd_set_int32 (pkptr1, "File Name",FILE);
                        op_pk_nfd_set_int32 (pkptr1, "Packet ID",
pk_ID);

                        //Copy packet 2 more times and send to
appropiate nodes
                        pkptr2 = op_pk_create_fmt("gnutella_packet");
                        op_pk_nfd_set_int32 (pkptr2, "Payload
Descriptor", QUERY);
                        op_pk_nfd_set_int32 (pkptr2, "TTL", TTL);
                        op_pk_nfd_set_int32 (pkptr2, "HOPS",HOPS);
                        op_pk_nfd_set_int32 (pkptr2, "Source
Address",src_address);
                        op_pk_nfd_set_int32 (pkptr2, "File Name",FILE);
                        op_pk_nfd_set_int32 (pkptr2, "Packet ID",
pk_ID);


                        pkptr3 = op_pk_create_fmt("gnutella_packet");
                        op_pk_nfd_set_int32 (pkptr3, "Payload
Descriptor", QUERY);
                        op_pk_nfd_set_int32 (pkptr3, "TTL", TTL);
                        op_pk_nfd_set_int32 (pkptr3, "HOPS",HOPS);
                        op_pk_nfd_set_int32 (pkptr3, "Source
Address",src_address);
                        op_pk_nfd_set_int32 (pkptr3, "File Name",FILE);
                        op_pk_nfd_set_int32 (pkptr3, "Packet ID",
pk_ID);


                        //Send them out to neighbouring nodes
                        if (receive_strm == RCV_IN_STRM1)
                                {
                                        op_pk_send (pkptr1,
XMT_OUT_STRM2);
                                        op_pk_send (pkptr2,
XMT_OUT_STRM3);
                                        op_pk_send (pkptr3,
XMT_OUT_STRM4);
                                }
                        else if (receive_strm == RCV_IN_STRM2)
                                {
```

```
                                        op_pk_send (pkptr1,
XMT_OUT_STRM1);
                                        op_pk_send (pkptr2,
XMT_OUT_STRM3);
                                        op_pk_send (pkptr3,
XMT_OUT_STRM4);
                                }
                        else if (receive_strm == RCV_IN_STRM3)
                                {
                                        op_pk_send (pkptr1,
XMT_OUT_STRM1);
                                        op_pk_send (pkptr2,
XMT_OUT_STRM2);
                                        op_pk_send (pkptr3,
XMT_OUT_STRM4);
                                }
                        else if (receive_strm == RCV_IN_STRM4)
                                {
                                        op_pk_send (pkptr1,
XMT_OUT_STRM1);
                                        op_pk_send (pkptr2,
XMT_OUT_STRM2);
                                        op_pk_send (pkptr3,
XMT_OUT_STRM3);
                                }
                        }
                }
                }

//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//
QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QUERY//QU
ERY


//QUERY HIT//QUERY HIT//QUERY HIT//QUERY HIT//QUERY HIT//QUERY HIT//QUERY
HIT//QUERY HIT//QUERY HIT//QUERY HIT//QUERY HIT//QUERY HIT//QUERY HIT//QUERY
HIT//QUERY HIT
else if (PD == QUERY_HIT)
        {
                //This is a QUERY HIT packet

                if (src_address == node_ID)
                {
                        //Destroy if packet was destined to this node
                        op_pk_destroy (pkptr1);
                        ++num_QUERY_HITS_G_R;
                        ++num_QUERY_HITS_L_R;
                        op_stat_write (num_QUERY_HITS_G_R_h,
num_QUERY_HITS_G_R);
                        op_stat_write (num_QUERY_HITS_L_R_h,
num_QUERY_HITS_L_R);
                        //Want to count number of received HITS
                }
                else if (TTL == 0)
```

```
                {
                        //destroy packet and take no further action
                        op_pk_destroy (pkptr1);
                }

                else
                        //Forward PONG packet to appropiate route where packet
came from
                {
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
                        if (src_address == 0)
                                {
                                        op_pk_send (pkptr1, node_0);
                                }
                        else if (src_address == 4)
                                {
                                        op_pk_send (pkptr1, node_4);
                                }
                        else if (src_address == 10)
                                {
                                        op_pk_send (pkptr1, node_10);
                                }
                        else if (src_address == 11)
                                {
                                        op_pk_send (pkptr1, node_11);
                                }
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION//THIS IS THE ROUTING PROCEDURE SECTION//THIS IS THE ROUTING PROCEDURE
SECTION
                }

        }
//QUERY HIT//QUERY HIT//QUERY HIT//QUERY HIT//QUERY HIT//QUERY HIT//QUERY
```

```
HIT//QUERY HIT//QUERY HIT//QUERY HIT//QUERY HIT//QUERY HIT//QUERY HIT//QUERY
HIT//QUERY HIT
```

State Variables

```
/* Randomly assigns a file to search for */
Distribution *        \file_search_dist;

Distribution *        \files_stored_dist;

/* Denotes which files are stored */
int         \file1;

int         \file2;

int         \file3;

int         \file4;

int         \file5;

/* This value is used to identify each node. */
/* We'll need it for routing purposes        */
int         \node_ID;

/* ACCUMULATORS */
int         \num_PINGS_L_C;

/* ACCUMULATORS */
int         \num_PINGS_L_R;

/* ACCUMULATORS */
int         \num_PONGS_L_C;

/* ACCUMULATORS */
int         \num_PONGS_L_R;

/* ACCUMULATORS */
int         \num_QUERIES_L_C;

/* ACCUMULATORS */
int         \num_QUERIES_L_R;

/* ACCUMULATORS */
int         \num_QUERY_HITS_L_C;

/* ACCUMULATORS */
int         \num_QUERY_HITS_L_R;

int         \PING_cache1;
```

```
int         \PING_cache10;

int         \PING_cache2;

int         \PING_cache3;

int         \PING_cache4;

int         \PING_cache5;

int         \PING_cache6;

int         \PING_cache7;

int         \PING_cache8;

int         \PING_cache9;

int         \QUERY_cache1;

int         \QUERY_cache10;

int         \QUERY_cache2;

int         \QUERY_cache3;

int         \QUERY_cache4;

int         \QUERY_cache5;

int         \QUERY_cache6;

int         \QUERY_cache7;

int         \QUERY_cache8;

int         \QUERY_cache9;

/* STAT HANDLES */
Stathandle         \num_PINGS_G_C_h;

/* STAT HANDLES */
Stathandle         \num_PINGS_G_R_h;

/* STAT HANDLES */
Stathandle         \num_PINGS_L_C_h;

/* STAT HANDLES */
Stathandle         \num_PINGS_L_R_h;

/* STAT HANDLES */
Stathandle         \num_PONGS_G_C_h;
```

```
/* STAT HANDLES */
Stathandle        \num_PONGS_G_R_h;

/* STAT HANDLES */
Stathandle        \num_PONGS_L_C_h;

/* STAT HANDLES */
Stathandle        \num_PONGS_L_R_h;

/* STAT HANDLES */
Stathandle        \num_QUERIES_G_C_h;

/* STAT HANDLES */
Stathandle        \num_QUERIES_G_R_h;

/* STAT HANDLES */
Stathandle        \num_QUERIES_L_C_h;

/* STAT HANDLES */
Stathandle        \num_QUERIES_L_R_h;

/* STAT HANDLES */
Stathandle        \num_QUERY_HITS_G_C_h;

/* STAT HANDLES */
Stathandle        \num_QUERY_HITS_G_R_h;

/* STAT HANDLES */
Stathandle        \num_QUERY_HITS_L_C_h;

/* STAT HANDLES */
Stathandle        \num_QUERY_HITS_L_R_h;

int        \file6;

int        \file7;

int        \file8;

int        \file9;

int        \file10;

int        \node_0;

int        \node_4;

int        \node_10;

int        \node_11;
```