ENSC 427 Communication Networks
Implementation of the Gnutella Protocol
Spring 2010

FINAL PROJECT

ZHIYU HU 301035686

zyh@sfu.ca

YUYUAN LIU 301046236

yla41@sfu.ca

Webpage:

www.sfu.ca/~yla41

Table of content

1. Abstract

A peer-to-peer, commonly abbreviated to P2P, is any distributed network architecture composed of participants that make a portion of their resources (such as processing power, disk storage or network bandwidth) directly available to other network participants, without the need for central coordination instances (such as servers or stable hosts) [1]. Gnutella Protocol is one of the peer-to-peer protocols developed by J.Frankel and T. Pepper in 2000. In late 2007, it was the most popular file sharing network on the Internet with an estimated market share of more than 40%[2]. In June 2005, Gnutella's population was 1.81 million computers[3]increasing to over three million nodes by January of 2006[4]. This project aims to investigate the characteristics of Gnutella Protocol v0.4 in OPNET. Packet format, node model and process model are built to assist the creation of the Gnutella Protocol. Note that only Ping, Pong, Query and Query Hit will be simulated in this project. The details of how to create them are presented in this report. Moreover, many simulations are conducted based on different topologies such as Hexagon, Duplicated Hexagon, Line and Tree etc. The simulation results as well as analysis will also be provided in this report.

2. Introduction and Background

2.1 P2P Networks Introduction

In recent years, Peer to Peer Computing (called P2P) quickly became one of the hot topics in computer sciences, Fortune magazine mark the Peer to Peer Computing as most influence Internet technology in the future.

P2P is a distributed network of participants to share their own part of the hardware resources (processing power, Storage capacity, network connectivity, printers, etc.)[5]. These shared resources needed by the P2P service can have other peer nodes (Peer) direct access without going through intermediate entities. Participations in this network are not only resources provider, but also resources downloader. These special identities are called servent [6].

P2P practical application is mainly reflected in the following areas:

2.1.1 P2P Distributed Storage

P2P is a distributed storage system for peer networks. It can provide an efficient file access and load-balancing saving features. There are several applications based on the super-point structure of semi-distributed P2P applications such as Kazza, Edonkey, and Morpheus [7]. Bit torrent is also part of a distributed storage areas and the number of users increased dramatically year by year.

2.1.2 The Sharing of Computing Power

Beside share storage capacity, joining the right-peer network may also be able to share the CPU processing power. We have already seen some computing power sharing system based on peer networks, such as SETI @ home. Although SETI @ home using NaPster [8] centralized directory Strategy, it still makes a step further to real equal share system. Such computing power-sharing system can be used for gene database directory or password cracking applications which require large-scale computing power.

2.1.3 P2P Application Layer Multicast

The application layer multicast is a P2P application which can be executed without the need of network layer support. This could avoid the delay in the deployment of network-layer multicast support for multicast applications.

Although P2P has broad application prospects, because of the lack of central control, as well as the dynamic nature of freedom and equality, self-organized P2P networks have many difficult issues at the technical level. P2P network node itself is often in large difference between computing powers, each node is given the same duties without regarding to its computing power and network bandwidth [9]. Local point of poor performance will lead to an overall network performance deterioration. So for the nodes in this heterogeneous environment, it is difficult to achieve optimal resource management and load balancing. Also, because users join or leave the P2P randomness allows users to obtain the target file with the uncertainty, resulting in a number of file download is not necessary, but a large number of bandwidth resources caused by the abuse.

2.2 Introduction to Gnutella Protocol

Gnutella is a protocol for distributed search. Although the Gnutella protocol supports a traditional client/centralized server search paradigm, Gnutella's distinction is its peer-to-peer, decentralized model [8]. 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. They provide client-side interfaces through which users can issue queries and view search results, while at the same time they also accept queries from other servents, check for matches against their local data set, and respond with applicable results. Due to its distributed nature, a network of servents that implements the Gnutella protocol is highly fault-tolerant, as operation of the network will not be interrupted if a subset of servents goes offline.

2.2.1 The mechanism of Gnutella Network

2.2.1.1 Descriptors

The Gnutella protocol consists of 5 descriptors. They are Ping, Pong, Query, Query Hit and Push [8]. This project will focus on first four descriptors. Ping is used for Gnutella Node to actively probe the network for other servents [8]. It will be flooded by every servent if TTL is greater than zero. A servent receiving a Ping descriptor may elect to respond with a Pong descriptor [8]. Pong descriptor is only sent in response to an incoming Ping descriptor. In our design, the replying Pong will reach the node sending the Ping in the reversed path of Ping. In other words, if Node A sends a Ping that reaches Node B along the path1, Node B will send a Pong backward to reach Node A along the path1. Query is used to search the appropriate content in the Gnutella network. It will be flooded by every node if TTL is greater than zero. If a servent has the right content, a Query Hit is generated and sent back. Query Hit will reach the node sending the Query in reversed path of Query, similar to Pong routing.

2.2.1.2 Packet Routing

Figure 2.2.1.2.1 shows how Ping and Pong work in Gnutella networks [8]. Notice that a Gnutella node will do two things if the received packet is Ping. First it will forward Ping to other nodes which are directly connected to itself. Second it will reply a Pong back. Two mechanisms are implemented to control the total traffic. First Ping will be controlled by TTL and Hops. TTL will decremented whenever the packet passes by a node. If TTL is equal or smaller than zero, Ping will not be further forwarded. Second every node has its own Ping cache, if the Ping received is in the cache, it will not be forwarded. Pong will reach the node that sends the Ping along the reversed direction of Ping. For example, in figure 2.2.1.2.1, Node7 receives a Ping from Node1. The path of Ping is node1->node2->node4->node7. Thus, Pong will go backward, from node7 to node4, to node2, finally reaches node1.

Figure 2.2.1.2.2 shows how Query and Query Hit work in Gnutella network. If a node receives a Query, the node will first check its own data pool. If the content in data

pool matches the search criteria, a Query Hit will be sent by the node. Second, the node will check TTL in Query, if TTL is greater than zero; Query will be forwarded to its neighbours. The routing rule of Query Hit is same as that for Pong.

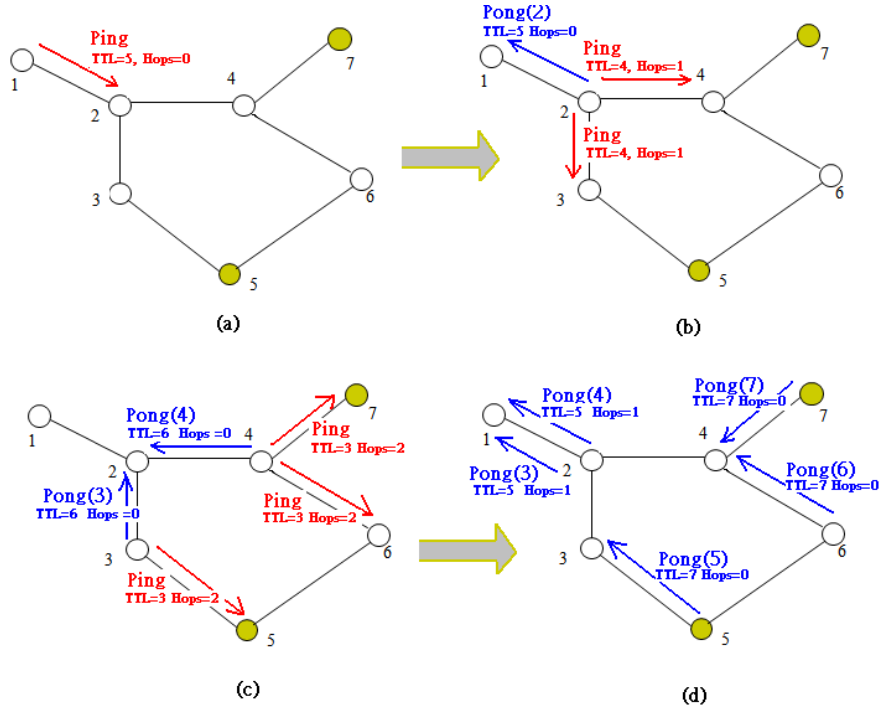


Image source:H.Su and K.Wu, "Gnutella Network Robustness," *ensc.sfu.ca*, report.

Figure 2.2.1.2.1

Once a Query Hit is received by a node, it will initiate data transfer from the sender of Query Hit [11]. The file described in Query Hit will be downloaded by HTTP protocol. Our project does not simulate file downloading.

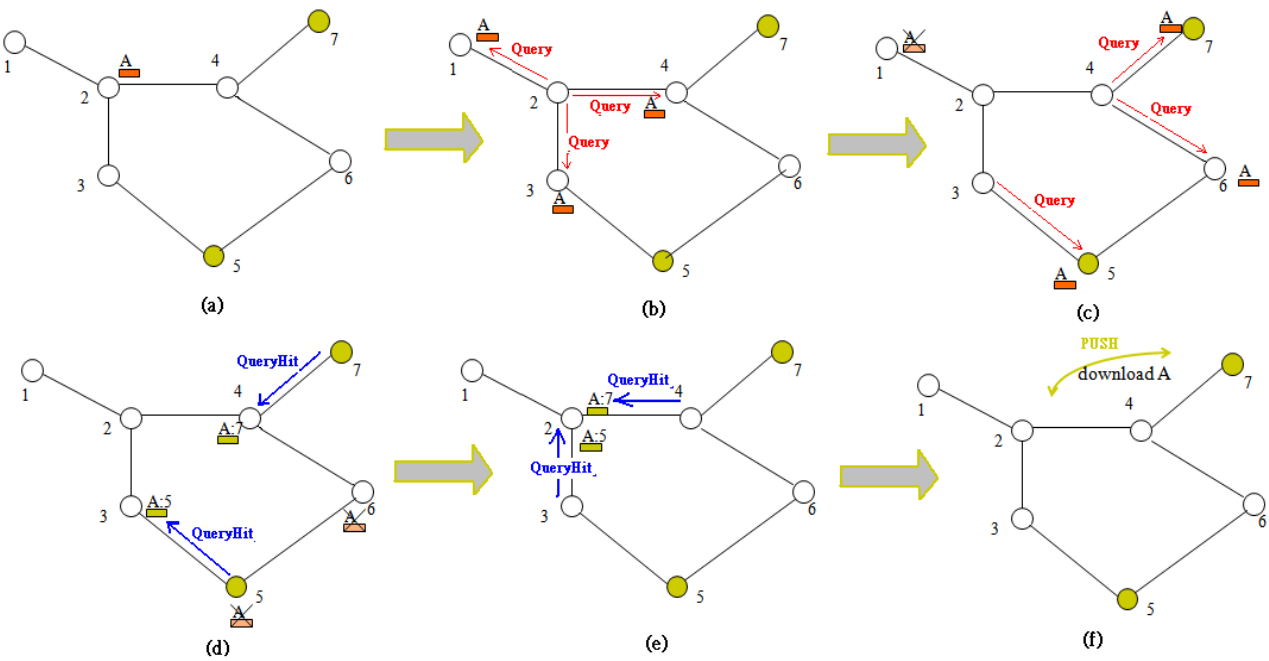


Image source:H.Su and K.Wu, "Gnutella Network Robustness," *ensc.sfu.ca*, report.

Figure 2.2.1.2.2

2.3 Scope of the Project

The goal of our project is to create a simplified Gnutella node in OPNET according to Gnutella Protocol V0.4. This Gnutella node will simulate the behaviours of four descriptors, namely, Ping, Pong, Query and Query Hit. In order to do that, we need to design Packet Format, Node model and Process model.

By using Gnutella node, we create different topologies, such as Hexagon, Duplicated Hexagon, Tree and Line etc. Many simulations will be done with these topologies to investigate the characteristics of our Gnutella node.

The real Gnutella network is much more complex than our model no matter from network scale or algorithm implemented. The packet format designed and used is not the same as the real world Gnutella Packet Format. Ours is a very simplified version. For more details about real Gnutella Protocol, please refer to the eighth entry of our reference.

3.0 Implementation of Project

3.1 Implementation of the Gnutella Node

3.1. 1 Packet Format

The packet format created is used for Gnutella node to communicate with each other. All of four descriptors, Ping, Pong, Query and Query Hit will share the same packet format.
The size of the packet format is 200 bits. There are 8 fields in the packet format. Table 3.1.1 clearly lists the name and the size of each field. Figure 3.1.1 shows the packet format in OPNET.

Field	Size (bit)
Packet_id	32
TTL	32
Hops	32
Node_id	32
Dest_addr	32
Sender_node_id	32
Payload Descriptor	4
search	4

Table 3.1.1

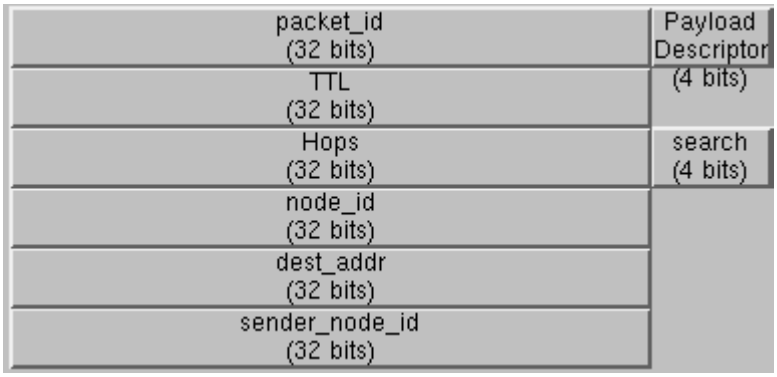


Figure 3.1.1 designed packet format for Gnutella Node

Packet_id is used to record the current packet identification number. The number is generated uniquely by OPNET when the packet is created. Payload Descriptor is used to indicate the types of the packet. We use 0x1 to indicate Ping, 0x2 to indicate Pong, 0x4 to indicate Query and 0x8 to indicate Query Hit. In the search field, as long as a Query packet created, an integer ranged from 0 to 2 will be generated according to the uniform distribution and it will be assigned to this field. If the packet is not Query, then 0 will be assigned to this filed. TTL and Hops are used to control the total traffic.TTL will be decremented and Hops will be incremented whenever a packet passes by a Gnutella node. For the packets which need to be flooded, such as Ping and Query, when TTL is smaller or equal to zero, the packet will be destroyed and the packet forwarding will not occur. Node_id is used to track the sender of the parent packet. The parent packet of Pong is the Ping packet which it replies to. The parent packet of Query Hit is the Query packet which it replies to. For example, node 2 receives a Ping packet from node 1; node 2 will reply a Pong to node 1. The parent packet in this example is the Ping packet generated by node 1. Since this packet is generated by node 1, the node_id of replied Pong is 1. Sender_node_id is used to track the node which generates the current packet. For the last example, since Pong is gnerated by node 2, sender_node_id is 2. Dest_addr is used to route Pong and Query Hit packet. After decoding this field, each node will know through which transmitter the packet should be sent. The detailed routing algorithm will be provided in the process model section.

3.1.2 Link Model

The link model is created to support point to point duplex and packet format we created [11]. This link model only supports our Gnutella packet format. The value of data rate is 1024. Table 3.1.2 shows the important attribute settings.

Attribute	Value
ecc model	Ecc_zero_err
error model	Error_zero_err
prodel model	Dpt_prodel
txdel model	Dpt_txdel
External file	Link_delay

Table 3.1.2

3.1.3 Node Model

The node model consists of 6 pairs of transmitters and receivers, src and proc. Each pair of transmitter and receiver is connected by logical Tx/Rx association. This will ensure the correct transmitter and receiver pairs are used when connecting links to nodes at the network levels [11].

Src is a ping source, which sends a Ping every second. The process model for src is simple_source. The packet format supported is our Gnutella packet format. For convenience, the Packet Inter-arrival Time is promoted to higher level. If necessary, we could disable src to make a node without sending Ping out. This is used in our simulation. Proc is a packet processor. It manipulates every received packet. The processor proc will be introduced later. The begsim intrpt is enabled so that at the beginning of the simulation, the interrupt occurred will cause the process model going to the initialization state.

Notice that the connection between receivers, proc and transmitters must be established in ordered fashion. In other words, the index of each connection must correspond to the code developed in processor proc. The method to check the connection index is to right click the proc processor then chooses show connectivity.

Some important settings should be done in node interface. All the Gnutella nodes we developed are fixed nodes. Thus Node type will not support mobile and satellite. The figure 3.1.3 below shows the node model.

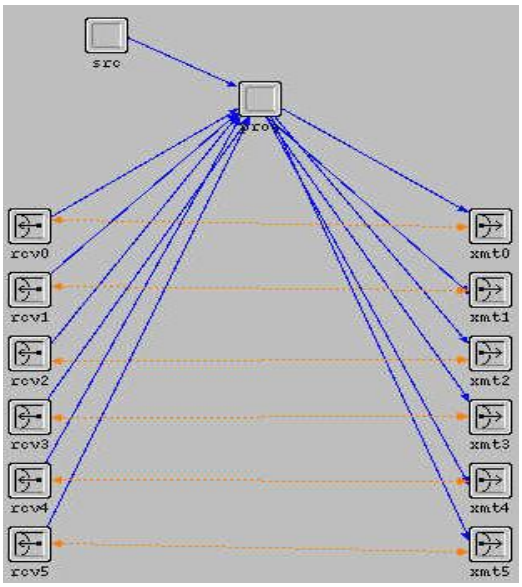


Figure 3.1.3 node model for the Gnutella node

3.1.4 Process Model

Process model is no more than FSM (finite state machine diagram). Figure 3.1.4.1 shows the designed process model. This process model goes into the proc in node model. There are three states in the model, namely, init, idle and proc. Init state is

initialization state. Idle state is used to wait for packet receiving. Proc state manipulates every received packet from one of the six receivers.

The init state will initialize all the state variables, caches, cache pointers and data pool for searching. It will also obtain handles of statistics by using op_stat_reg command and write the initial values into all the statistics.

When finishing initialization, FSM will go to idle state. If the packet is received from src (transition macro SRC_ARRIVAL), then the function "transmit_src_pk()" will be called. This function will do the following:

1. Assign proper value to each field of the newly created packet.
2. Copy this packet 5 times
3. Send these packets through transmitter 0 to transmitter 6.

If the packet is received from one of the six receivers (transition macro RCV_ARRIVAL), FSM goes to proc state. This state will process the packet according to a specific algorithm which will be talked about later.

3.1.4.1 Algorithm in Proc State

In order to recognize the packet type, the Payload Descriptor field will be first read. According to different descriptors, the packet will be processed in different ways.
Appendix 1.1 shows the procedure of processing the packet with Ping descriptor.
Appendix 1.2 shows the procedure of processing the packet with Pong descriptor.
Appendix 1.3 shows the procedure of processing the packet with Query descriptor.
Appendix 1.4 shows the procedure of processing the packet with Query Hit descriptor.

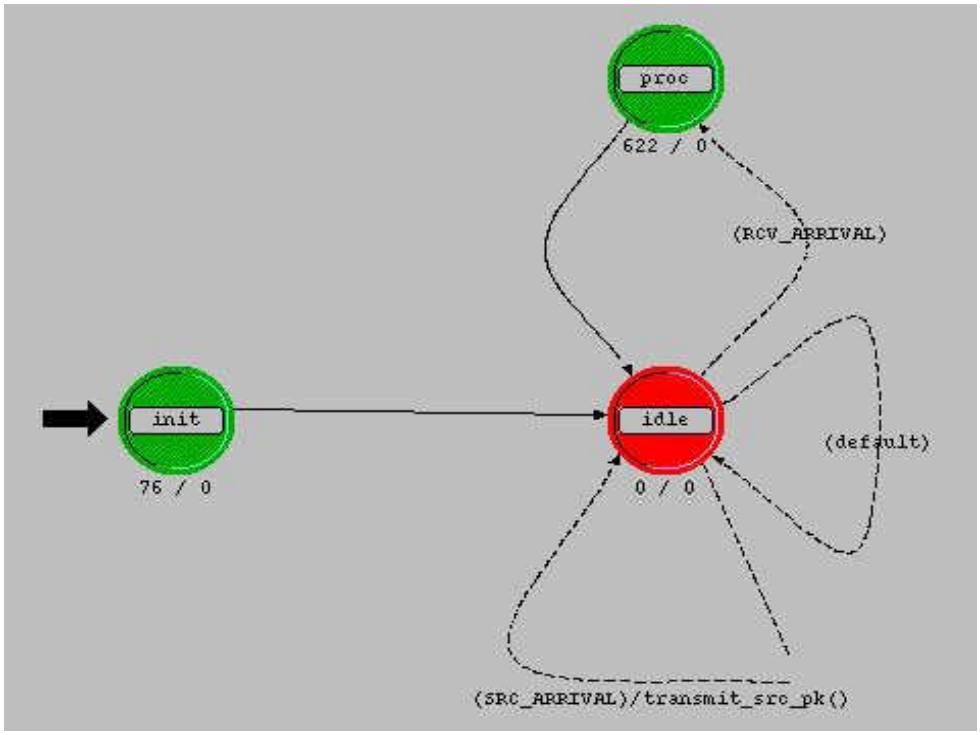


Figure 3.1.4.1 process model for the Gnutella Node

3.1.4.2 Routing Algorithm for Pong and Query Hit.

A specific example will be provided to demonstrate this algorithm. Suppose the network topology is a three nodes line. The duplex port 2 of first node is connected to the duplex port 4 of the second node. The duplex port 2 of second node is connected to the duplex port 3 of the third node. Figure 3.1.4.2 depicts this topology.

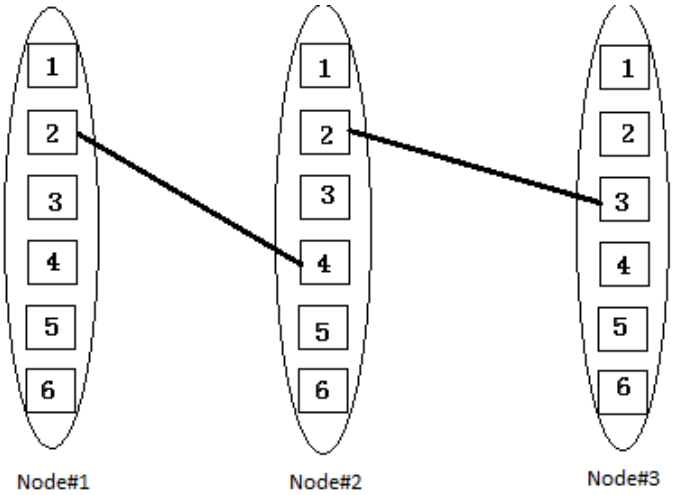


Figure 3.1.4.2 a specific topology for explaining routing algorithm

Suppose that node#1 sends a Ping to node#2, node#2 will forward this Ping to node#3 and node#3 will reply a Pong which should finally reach node#1 in the reversed path of Ping. In other words, Pong will start from node#3 port3 to node#2 port2, then from node#2 port4 to node#1 port2. The problem is how node#2 knows that this Pong should be forwarded through port4 while not other ports. Our algorithm is to solve this problem.

When Ping is about to send from node#1 port2, the field dest_addr will be update to the value of port number 2. When this Ping is about to forwarded from node#2 port2, we will update the dest_addr to 10*dest_addr+ duplex port number that receives this Ping. When node#3 receives this Ping, Pong will be replied. The dest_addr will be updated to the same value of parent Ping.

When Pong reaches node#2, dest_addr in Pong will be decoded to obtain the port number where Pong will be forwarded. The decoding scheme is that

Port number = dest_addr%10

Moreover, the dest_addr must be updated before the Pong forwarded. Dest_addr will be updated to (dest_addr-port number)/10.

In our specific example, when node#1 sends Ping to node#2, the dest_addr value is 2. When this Ping is forwarded from node#2 port2, dest_addr is updated to 10*2+4=24. The dest_addr value of Pong replied by node#3 will be the same as that in parent Ping. It is 24. When Pong arrives at node#2, 24 will be decoded to 24%10=4 and dest_addr will be updated to (24-4)/10=2. Thus, Pong will be forwarded through port4. In this way, Pong can be routed to the sender of its parent Ping in the reversed direction of Ping. Query Hit routing is the same as Pong routing.

4. Simulation Topology & Result:

In this project we have built up 4 distinguish topologies for different simulation aspects. Each topology is designed to test a particular functionality of Gnutella protocol in p2p network. All simulations were run 100 seconds and global packets Inter-arrival-time is set to be constant (1), which means SRC create "Ping" packets every other second. Other parameters are set as the default value. The statistical for every node are also promoted, but we only interesting in certain variables depend on the topologies

4.1 Hexagon Topology

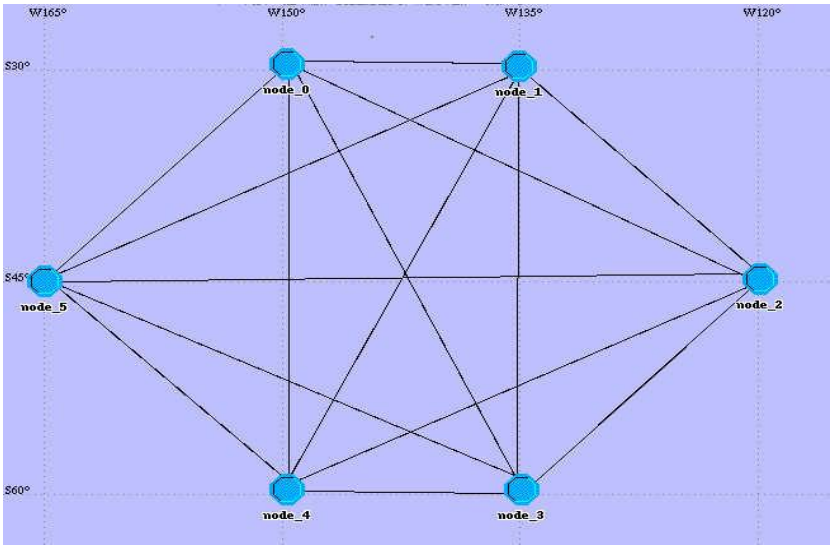


Figure 4.1.1 Hexagon Topology

Figure 4.1.1 shows the basic setup for Hexagon Topology simulations. This topology consists of six identical srevent nodes that inter connect with each other. Every node generates its own "PING" packets out. We also test the result of failed Node _3 situation of Hexagon Topology (Figure 4.1.2)

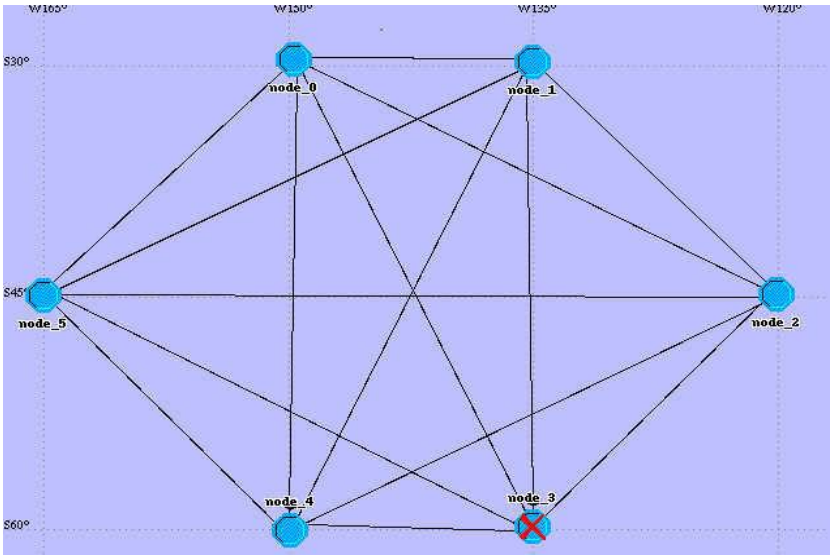


Figure 4.1.2 Failed Node _3 Hexagon Topology

Figure 4.1.3 and 4.1.4 shows the result of Hexagon Topology. Since all the non-broken nodes have similar results, we only focus on one particular node (Node_0). Figure 4.1.3 plotted out the packets numbers of income & outcome Ping, Pong, and Quarry hit format (Node_0 with fully connected Hexagon Topology). Compare to Figure 4.1.3, Figure 4.1.4 give us a similar statics but with node_3 failed in Hexagon topology. As expect all kinds of packet numbers are increasing linearly in both figures, the only difference between these two figures are the numbers of all packets received & transmitted by node_0 is about 1/6 less when Node_3 is failed. Obviously, a failed node in network makes the network smaller, so we expect decreasing numbers of received & transmitted packets. However we still can conclude that although failed node will decrease the numbers of received & transmitted packets, the functionality of the network remains the same.

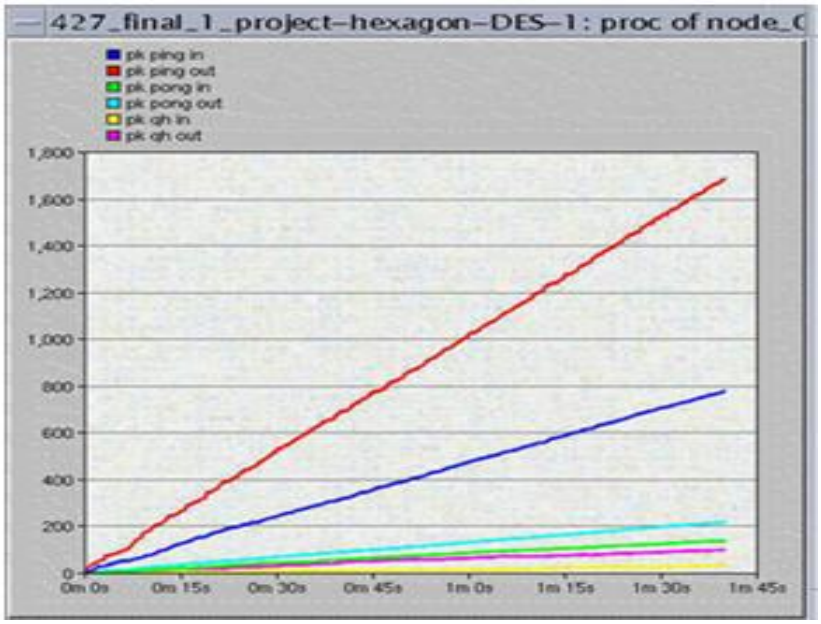


Figure 4.1.3 Node_0 results with no failed node

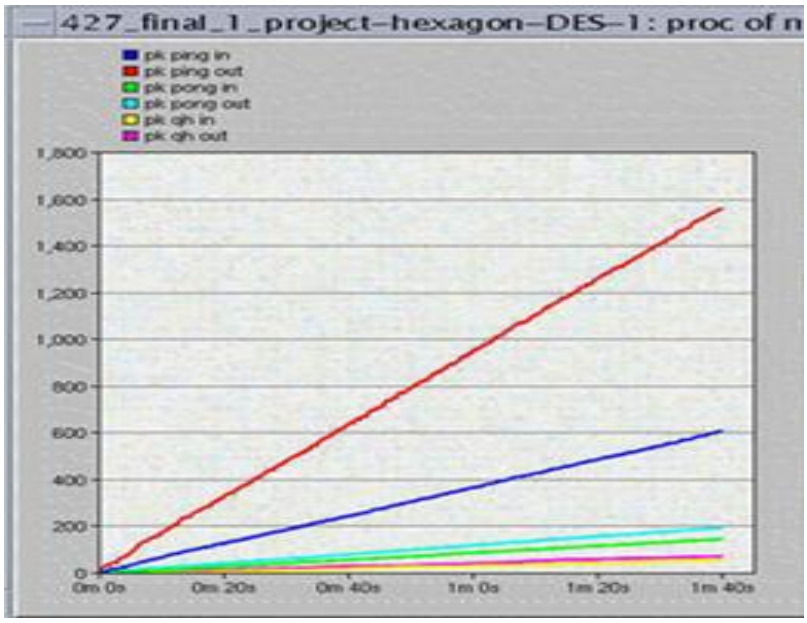


Figure 4.1.3 Node_0 results with Node_3 failed

4.2 Duplicated Hexagon Topology

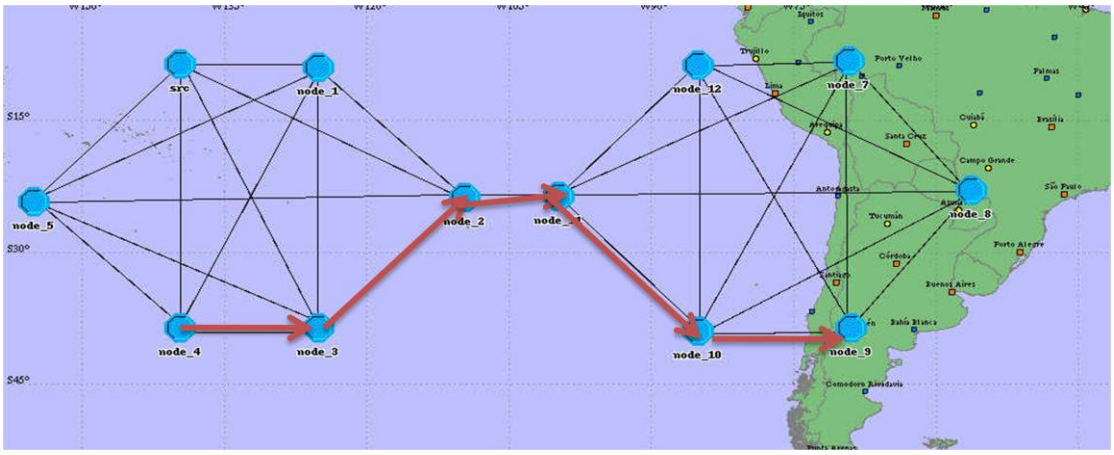


Figure 4.2.1 Duplicated Hexagon Topology

Figure 4.2.1 shows the basic setup for Duplicated Hexagon Topology simulations. This topology consists of eight identical servent nodes that dived into two sub-networks connect with each other. Only SRC node (circled in Figure) generates its own "PING" packets out. So expected the SRC_node, other nodes will not generate a search request.

Figure 4.2.2 Duplicated Hexagon Topology SRC_node results

Figure 4.2.1 lists average number of “Ping” packet format transmitted by SRC_node (blue), average number of “Quarry” packet format generated by SRC_node (flowered not include, red), and average number of “Quarry-Hit” packet format received by SRC_node (green). All these numbers are linearly increased with time, and as expected, SRC_node transmitted a large scale of “Ping” packet format out. As “Quarry” (generated by SRC_node) flooded out, we received more “Quarry-Hit” packets from the net work.

In order to see whether the SRC_node have the accessibility form sub net 2 we need the result from Node_9 to Node_12 (nodes in sub-net 2)

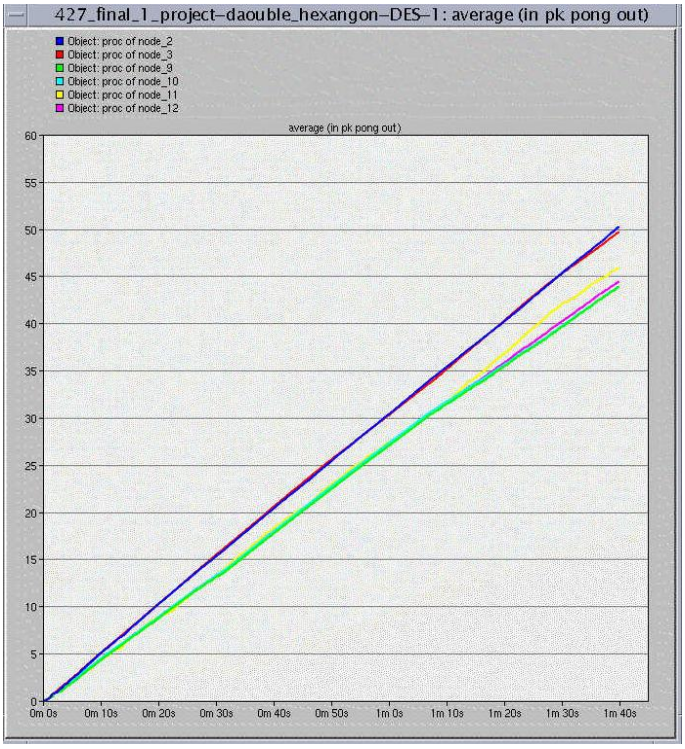


Figure 4.2.3 “PONG” out numbers

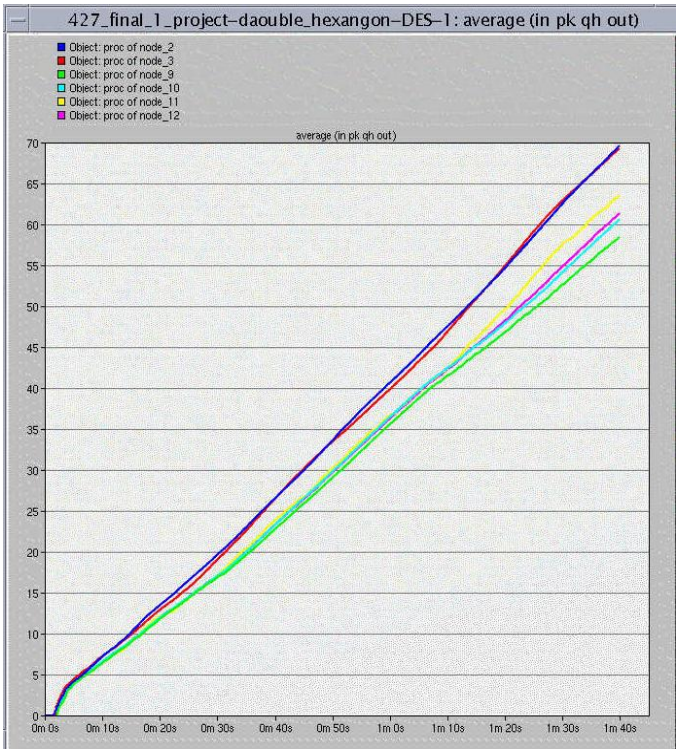


Figure 4.2.4 “Quarry Hit” out

Figure 4.2.3 collected “Pong” signal numbers transmitted by node_2 (blue), node_3(red), node_9-12 (from green to pink listed on upper left of Figure) respectively. Figure 4.2.4 collected “Quarry hit” signal transmitted by the same node listed in Figure 4.2.3. Because SRC_node is the only node that grated its own “Ping” massage out to the network, all the transmitted “Pong” and “Quarry hit” signals are received only by SRC_node. So if there exists “Pong” and “Quarry hit” transmitted by one node, this node is accessible by SRC_node. As one can see from figures, both “Pong” and “Quarry hit” numbers are increasing linearly with time which means SRC_node have the accessibility of the entire node listed in the figure. These figures also shows more “Pong” and “Quarry hit” signals can be received by SRC_node from Node_2 and Node_3 then other nodes. This is reasonable because Node_2 and Node_3 are laid in the same network with SRC_node, where as Node_9 to Node_12 are in subnet-B. However we can still conclude that SRC_node have fully connected

with subnet-B as the simulation results indicate.
We also recode two throughput results for the link **Node_2<->Node_3 (blue)** and link **Node_2<->Node_11 (red)**. Results show that both links are reached to a steady state as time pass by. **Node_2<->Node_11** have higher throughput as expect because Node**_2<->Node_11** connected the two subnet together so we expect higher information communicate rate than **Node_2<->Node_11 (**connection between two node with a same subnet**).**

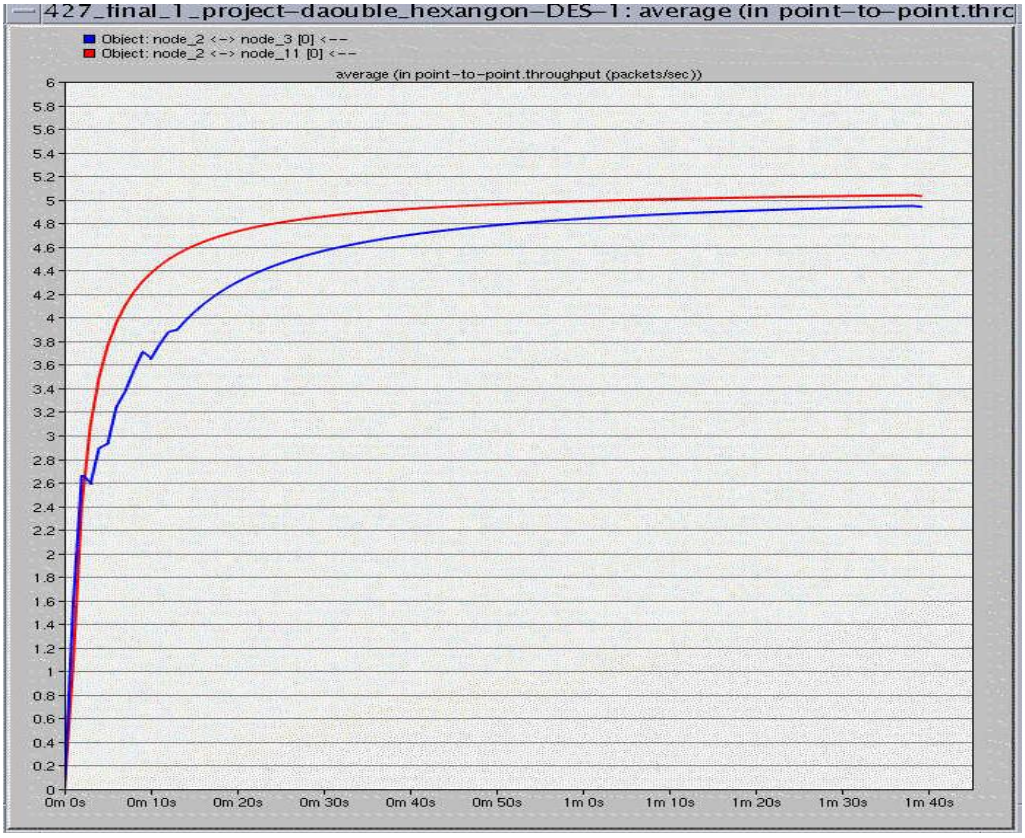


Figure 4.2.4 Throughput Compare

4.3 Line Topology

Figure 4.3.1 Line Topology[1]

We setup the line Topology as showed in Figure 4.3.1, 8 nodes connect by a single

[1] This line topology was originally created by H.Su and K.Wu, we modified it for our simulation

line. Only the SRC node in the beginning generates its own “PING” packets out. We use this simple topology to test the functionality and the importance of the TTL (Time to live). We run two TTL simulations, TTL = 5 and TTL = 50. Simulation results are showed in the following figures.

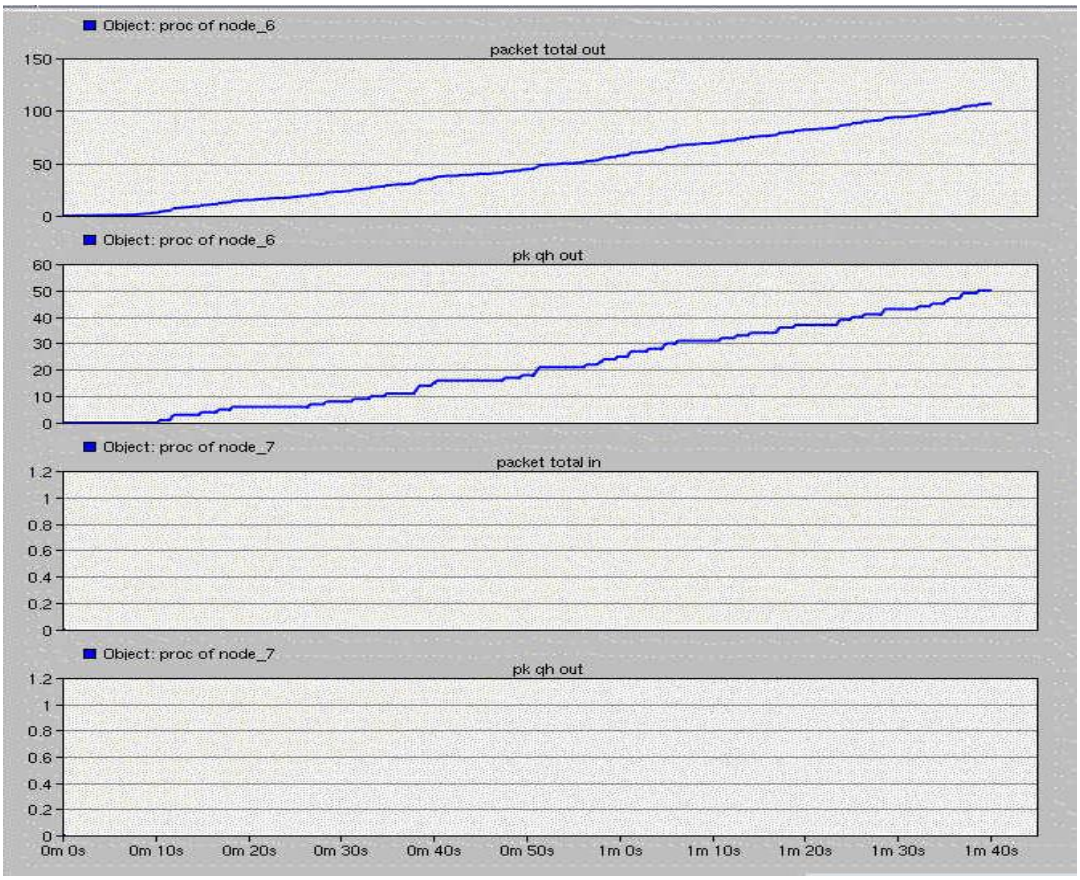


Figure 4.3.2 TTL=5 Node_6 and Node_7 results

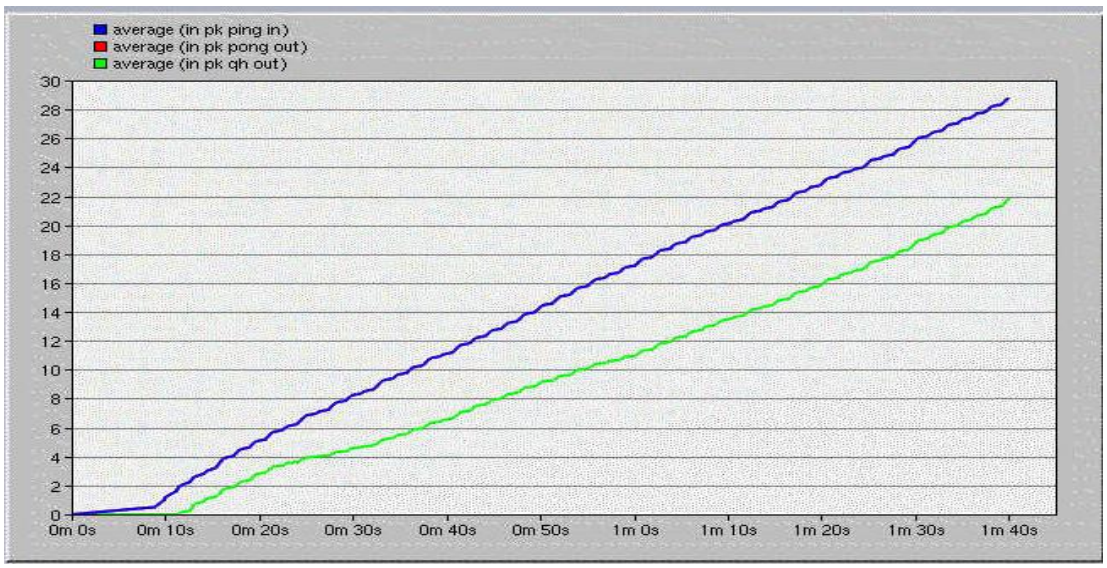


Figure 4.3.3 TTL = 50 Node_7 packets statistics

Figure 4.3.2 plotted out the total packets output and the “Quarry Hit” output numbers

for both Node_7 and Node_6. If TLL = 5 then “Ping” packets send by SRC_node cannot reached node _7 before TTL = 0, so we expect no packets in or out from Node_7. The simulation results coincide with our expectations. Compare to Figure 4.3.3 TTL= 50, node 7 is fully functioned under this situation (both “QH out” and “pong out” increasing linearly with the received Ping packages in). So we can conclude TTL plays a very important role in P2P net works.

4.4 Tree Topology

The Tree Topology has two kinds of topology integral in it, the star and the linear way of connecting to nodes. The tree like structure allows we to have many servants on the network and we can branch out the network in many ways. This is particularly helpful for colleges, universities and schools. We set up a 4 level tree Topology as graphed in Figure 4.4.1

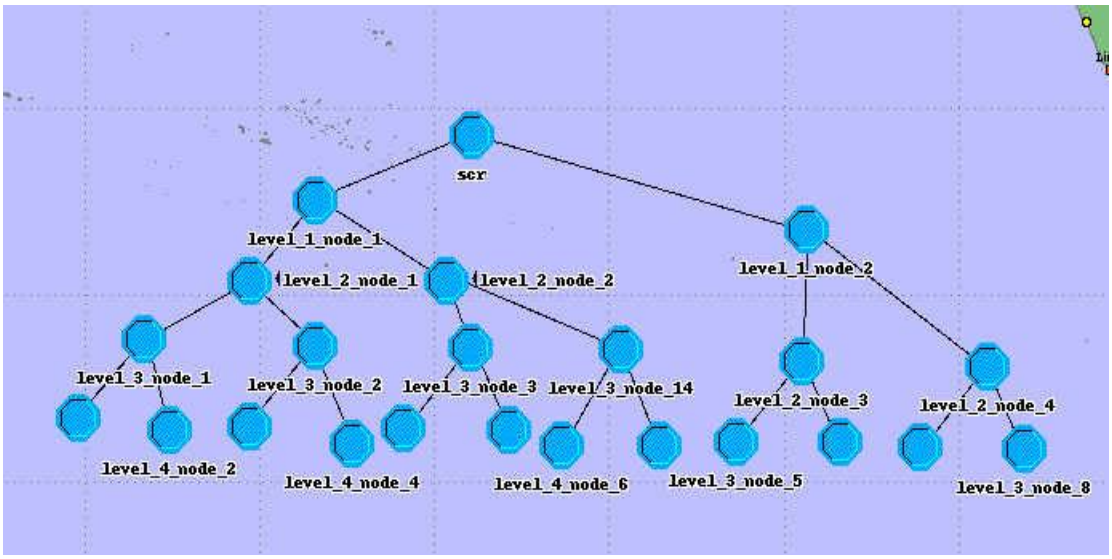


Figure 4.4.1 Tree Topology

In order to simulate quarry-flooding searching mechanism, we made the SRC_node as the only “Ping” source node in tree topology. So nodes in other levels will not send the reaching request.

Figure 4.4.2 provides the number of “PING” packets received & transmitted, “PONG” packets received, and the “Quarry Hit” packets received. As SRC_node linearly “ping” out the information, it is receiving “Pong” and “Quarry-Hit” continually. So SRC_node have successfully reached data it wanted.

Figure 4.4.3 curves the statics of “Quarry-Hit” packet replied by a random selected node of each level. As graph point out, all level nodes have “Quarry-Hit” packet replied with a minor number difference. So this simulation result indicated SRC have successfully reached data it wanted from every level of the tree topology

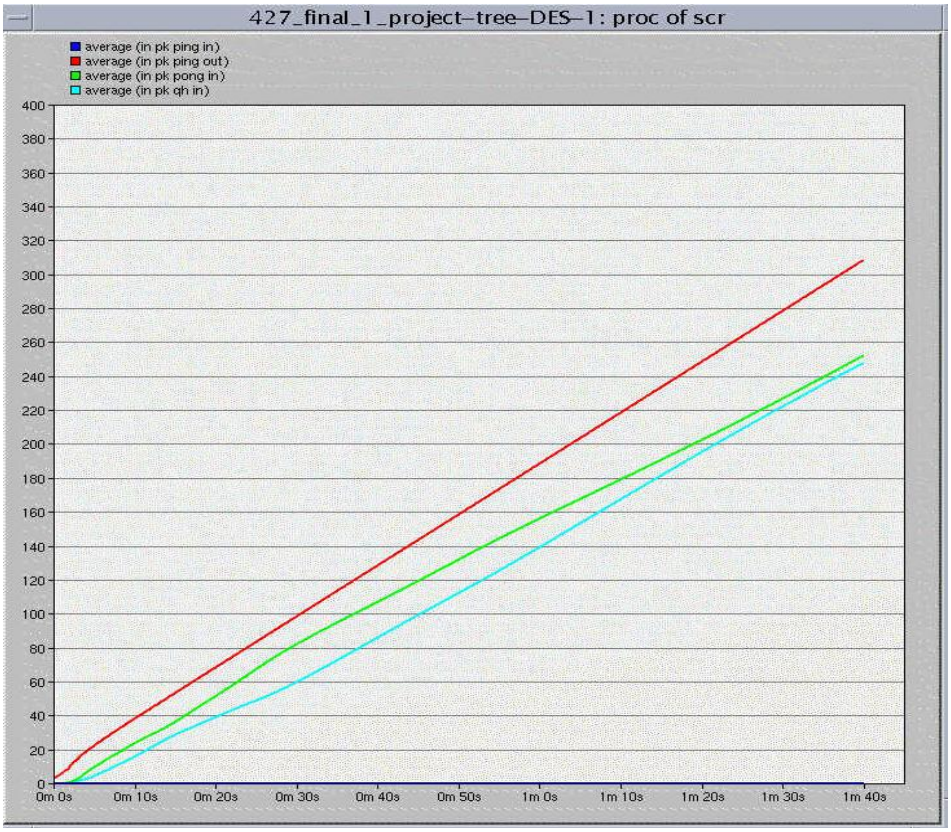


Figure 4.4.2 SRC_Node result

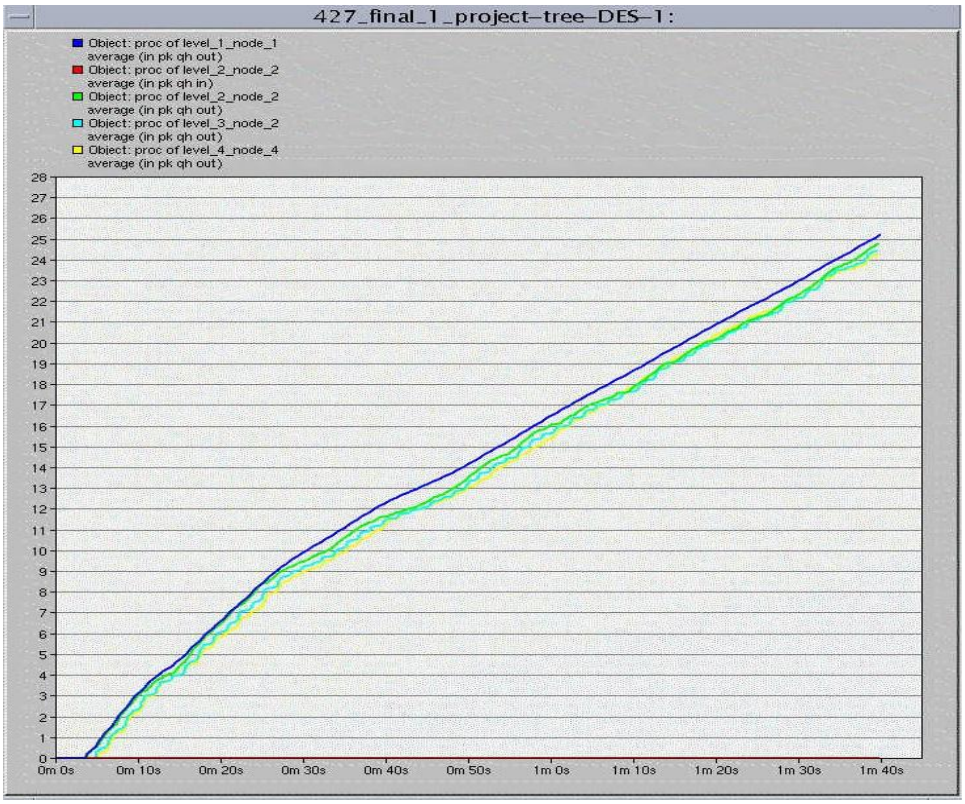


Figure 4.4.3 "Quarry-Hit" packet replied by a random selected node of each level

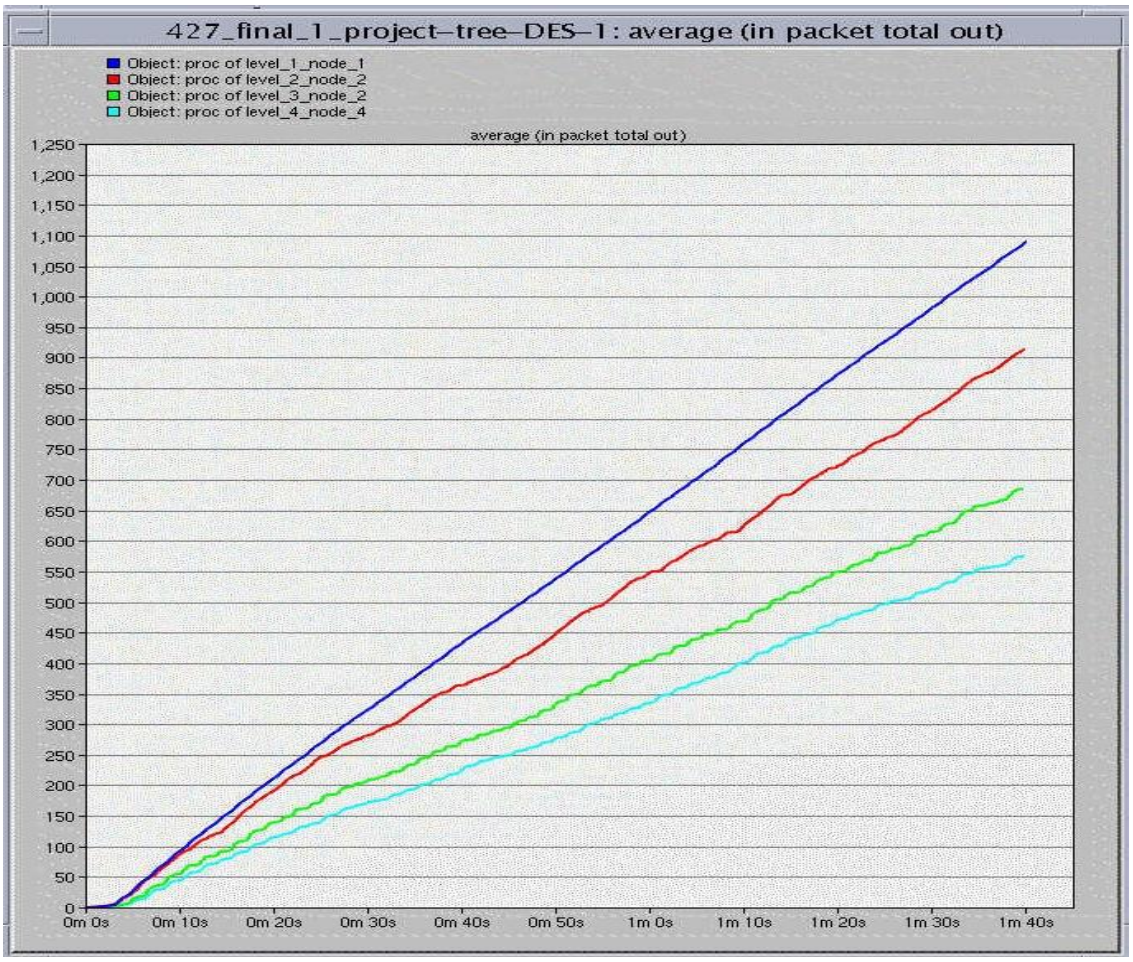


Figure 4.4.4 Total Packets Output

Beside above statistics we also did a total packets output recording for each node of our tree network (Figure 4.4.4). Because the SRC_node is the only "Ping" source node in tree topology during the 100 second simulation time, only 100 "PING" packets goes out from SRC node. However, we can observe at least 600 packets output for any node in level 4 (Sky blue line in Figure 4.4.4). As node level goes up, we can see a dramatic increasing in node packets output number, printed numbers are listed in flowing table.

Table 4.4.1 PK_out numbers for different levels

LEVEL	Single node PK out	Total node inside this level	Total PK out for this level
1	1100	2	2200
2	900	4	3600
3	700	8	5600
4	600	8	4800
Total packets number for whole net work **16200**			

Above table also indicate that there are 16200 packets created by just 100 "Ping" in

100 seconds simulation time. So Quarry-Flood searching mechanism can generate a larger amount of traffic follow with in very short time.

5. Conclusions and Discussions

5.1 Conclusions

This project aims to simulate Gnutella Protocol v0.4. Due to the complexity of the real Gnutella servent, only Ping, Pong, Query and Query Hit will be simulated in OPNET. In order to do that, we create a packet format, node model and process model. We verify the behaviours of our model by using debugging mode of simulation. We also build 4 topologies to investigate the characteristics of the Gnutella node. They are Hexagon, Duplicated Hexagon, Line and Tree.

From the simulation result we can conclude that compared to traditional network, P2P network are more reliable and easier to set up. A single node failure will not affect on the functionality of the whole network.

Query flooding is simple to implement and is practical for small networks with few requests. It contacts all reachable nodes in the network and so can precisely determine whether a resource can be found in the network

On the other hand, every request may cause every node to be contacted. Each node might generate a small number of queries; however, each such Query is flooded to the network. Thus, a larger network would generate far more traffic per node than a smaller one, making it inherently not scalable. Additionally, because a node can flood the network simply by issuing a request for a nonexistent resource, it could make the network paralytic.

5.2 Difficulties

Designing the algorithm in process model, especially the one in proc state is the most challenging part. This design involves many times of refining. To test the algorithm, we start from a very simple line topology with only one node sending Ping. We trace every packet's behaviours in the model to see whether some unexpected behaviours occurs. Then we check it with more complex topologies to further verify our algorithm. The other difficulty is debugging the code. There are more than 800 lines of code. OPNET cannot debug the code by step-to-step run. So in order to debug, we have to trace some packets and set breakpoints in the code. It is really a hard work to debug 800 lines of code by that way.

5.2 Future Works

The Gnutella network can be created as that every node can be dynamically join in

and quit the network. This is actually not hard to achieve. At the beginning of the project, we have had the intention to do that. However, considering the controllability of the whole project, we give up that idea. The rough idea is that we could create and maintain a logic connection list in every Gnutella node. If some node intends to quit the network, a special packet will be flooded to inform every node in the network to gracefully close the connection. The informed nodes will update their logic connection list by deleting that node.

The other work can be done in the future is to dynamically update the logic connection list. For example, if node A has not receive any information from node B for about 5 minutes. Node A should delete the logic connection record between node A and node B from the logic connection list of node A. To achieve this, we need a timer in every node. When timer times out, the logic connection list has to be updated.

In addition, we could make an adjustable TTL. In other words, TTL can be adjusted automatically according to the situation of the network. This situation is the combination of many elements, such as the amount of nodes in the network, the transmission speed and the distance between each node etc. An appropriate TTL is a very important parameter for the performance of Gnutella network. If TTL is too big, an unnecessary traffic load will occur. If TTL is too small, the amount of logic connections is limited. Thus, developing an algorithm for adjusting TTL automatically is crucial.

Moreover, our simulations and project model are base on 0.4 version, latest versions of Gnutella Protocol uses the notion of super-peers or ultra-peers. It is very valuable to build new version Gnutella node and investigate the usefulness of super-peers and ultra-peers.

Reference

[1] R.Schollmeier, "*A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Application"s*, Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE (2002).

[2] E.Bangeman, "*Ars Technica Study: BitTorrent sees big growth*" [Online]. Available: http://arstechnica.com/old/content/2008/04/study-bittorren-sees-big-growth-limewire-still-1-p2p-app.ars. Accessed: March. 15, 2010

[3] T. Mennecke *"Slyck News- eDonkey 2000 Nearly Double the Size of FastTrack"* [Online]. Available: http://www.slyck.com/news.php?story=814 Retrieved: March 15th, 2010.

[4] A.Rasti, D.Stutzbach and R.Rejaie "*On the Long-term Evolution of the Two-Tier Gnutella Overlay*" University of Oregon. P.2.

[5] J. Cardoso and M. Lytras, *Semantic Web engineering in the knowledge society,* Hershey, PA ,2009.

[6] L. Alfred and W. Sing, *Peer-to-peer computing : building supercomputers with Web technologies / Alfred Wai-Sing Loo.* Springer : London, 2007.

[7] A . Dufour, *Improving the performance of the Gnutella network.* Simon Fraser University : Burnaby B.C 2006.

[8] "The Gnutella specification v0.4". www9.limewire.com/developer/gnutella_protocol_0.4.pdf. [Accessed: March. 15,2010].

[9] S. Osama, *Modeling and caching of peer-to-peer traffic.* Burnaby B.C: Simon Fraser University, 2006.

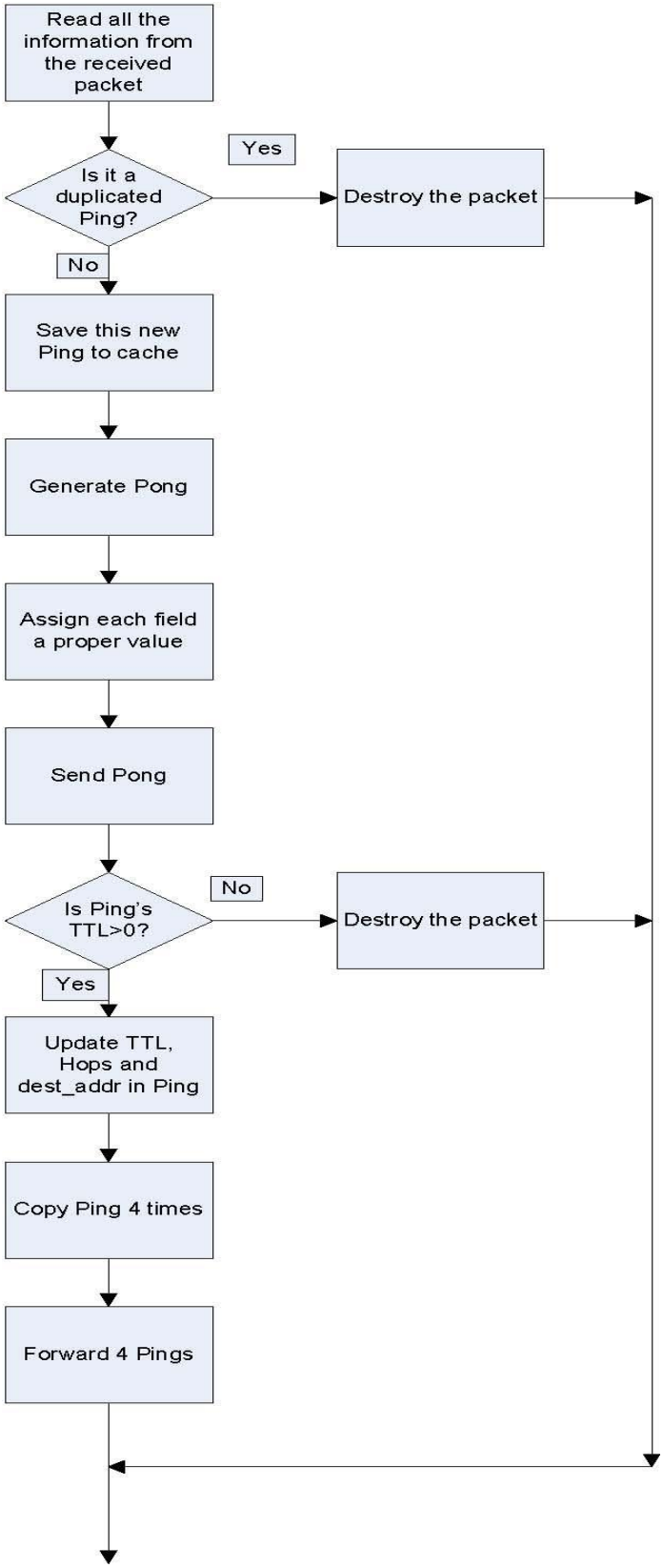
[10] D. Worthington; M. Nate (25 May 2005). "BitTorrent Creator Opens Online Search".BetaNews.[Online] Available: http://www.betanews.com/article/BitTorrent_Creator_Opens_Online_Search/1117065427. [Accessed: March. 05,2010].

[11] H.Su and K.Wu, "Gnutella Network Robustness," *ensc.sfu.ca*, report. Spring 2009. [Online]. Available: http://www.ensc.sfu.ca/~ljilja/ENSC427/Spring09/Projects/team12/Gnutella_Network_Robustness_Final_Version.pdf [Accessed: March. 05, 2010].

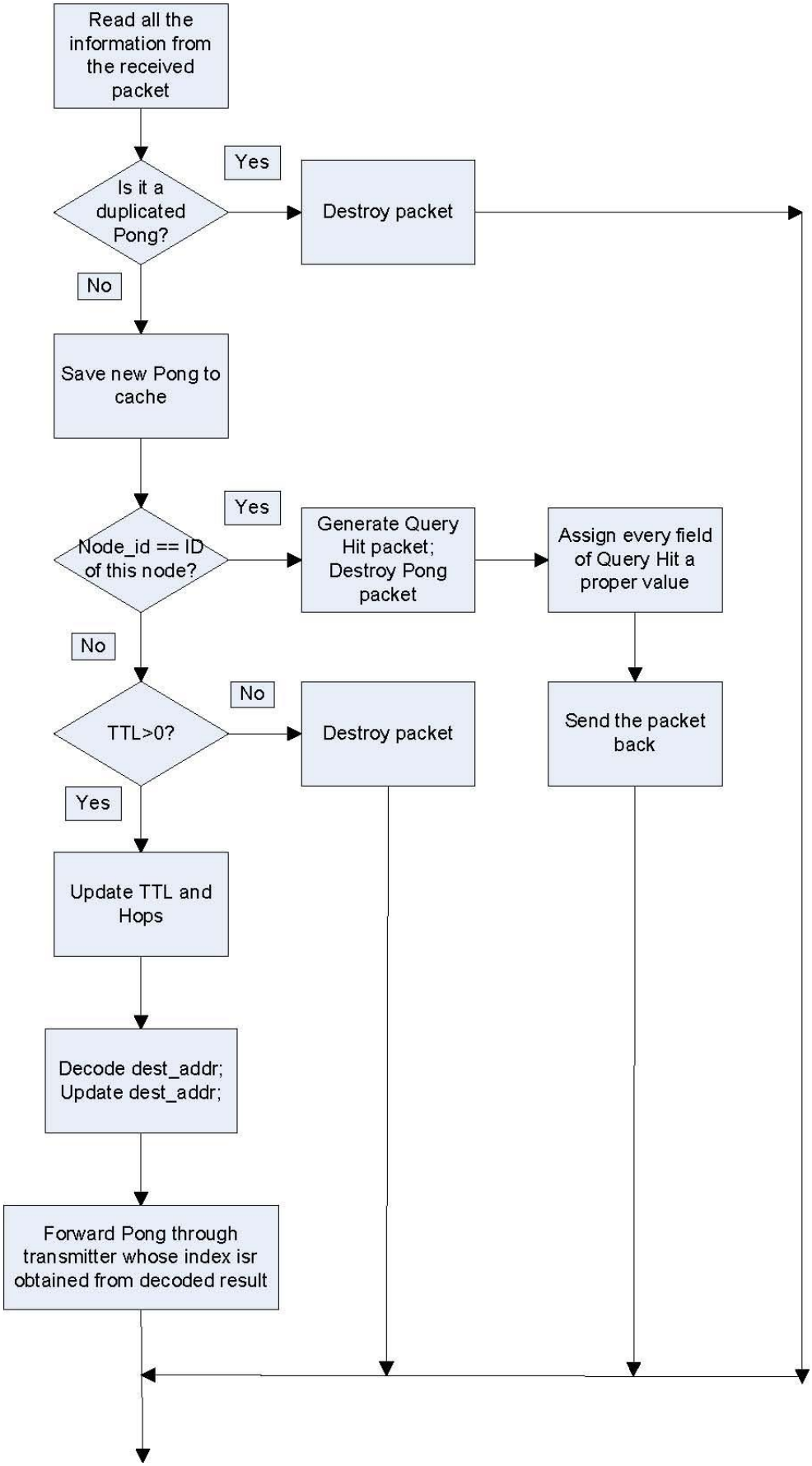
Appendix

Appendix 1 Processing flow chart

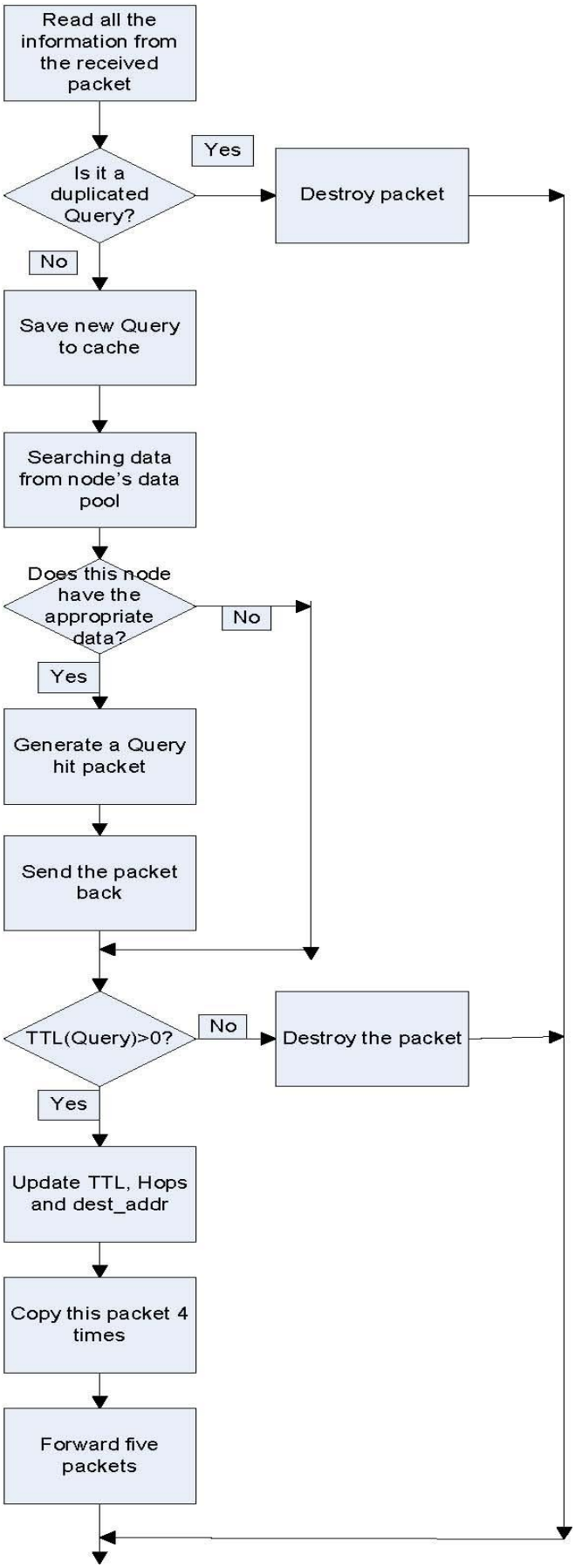
Appendix 1.1 the procedure of processing the packet with Ping descriptor



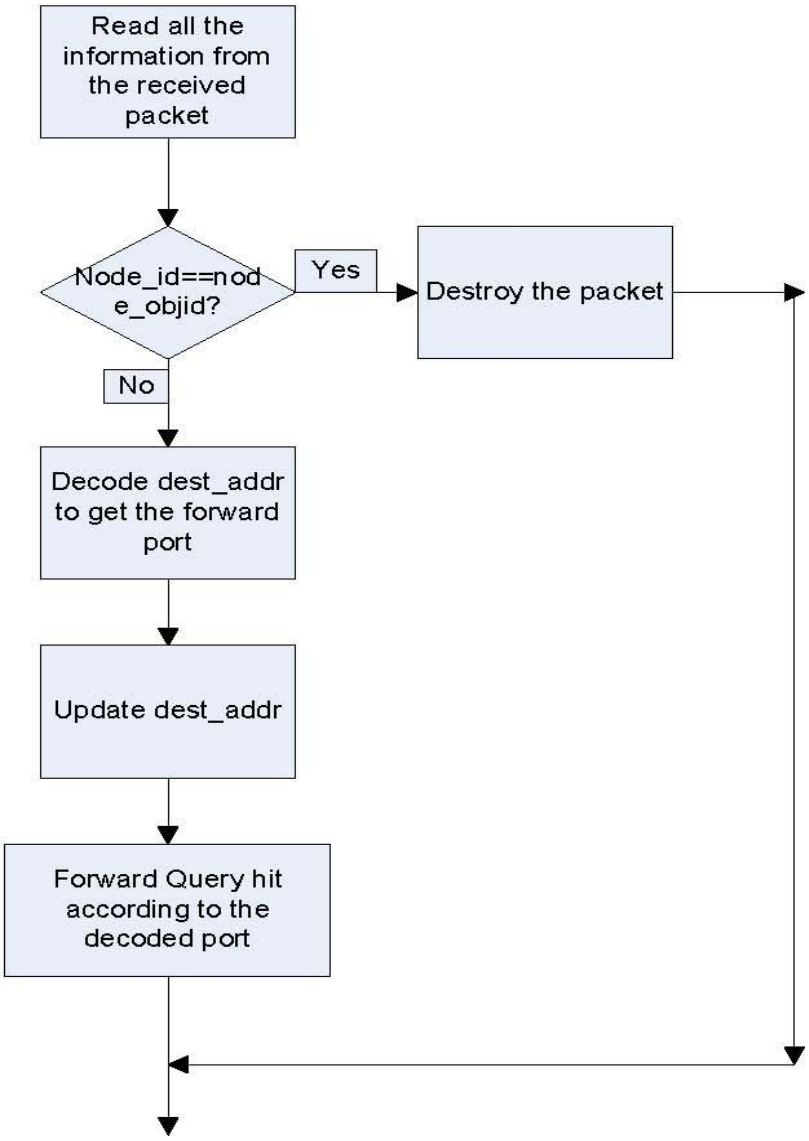
Appendix 1.2 Procedure of processing the packet with Pong descriptor



Appendix 1.3 Procedure of processing the packet with Query descriptor



Appendix 1.4 Procedure of processing the packet with Query hit descriptor



Appendix 2

Project Codes

```
/* Process model C form file: liu_testgnutella_node_proc_comment_code.pr.c */
/* Portions of this file copyright 1992-2007 by OPNET Technologies, Inc. */



/* This variable carries the header into the object file */
const char liu_testgnutella_node_proc_comment_code_pr_c [] = "MIL_3_Tfile_Hdr_ 140A 30A opnet
7 4BC96826 4BC96826 1 payette yla41 0 0 none none 0 0 none 0 0 0 0 0 0 0 0 18a9 3
";
#include <string.h>



/* OPNET system definitions */
#include <opnet.h>



/* Header Block */

#include <math.h>
/*packet stream definition*/
//in stream
#define src_in 0
#define rcv0_in 1
#define rcv1_in 2
#define rcv2_in 3
#define rcv3_in 4
#define rcv4_in 5
#define rcv5_in 6
//out stream
#define xmt0_out 0
#define xmt1_out 1
#define xmt2_out 2
#define xmt3_out 3
#define xmt4_out 4
#define xmt5_out 5

/*define message ids*/
//payload descriptor value
#define PING 1
```

```
#define PONG 2
#define Query 4
#define Query_hit 8

//TTL init
#define TTL_INIT 5

//ping pong cache size
#define CACHE_SIZE 1000
//data cache size
#define DATA_CACHE 1

//packet size
#define PK_SIZE 200;
//define cache structure
typedef struct
     {
          int packet_id;
          //int TTL;
          int Hops;
          int node_id;
          int dest_addr;
          int Payload_des;
          int search;

     }msg_cache;

//transition macro
//SRC_ARRIVAL
#define SRC_ARRIVAL (op_intrpt_type()== \
OPC_INTRPT_STRM && op_intrpt_strm()==src_in)

#define RCV_ARRIVAL (op_intrpt_type () == OPC_INTRPT_STRM \
&& ((op_intrpt_strm () == rcv0_in) \
|| (op_intrpt_strm () == rcv1_in) \
||(op_intrpt_strm () == rcv2_in) \
||(op_intrpt_strm () == rcv3_in)\
||(op_intrpt_strm () == rcv4_in)\
||(op_intrpt_strm () == rcv5_in)))

/* End of Header Block */

#if !defined (VOSD_NO_FIN)
#undef   BIN
```

```
#undef   BOUT
#define  BIN      FIN_LOCAL_FIELD(_op_last_line_passed) = __LINE__ - _op_block_origin;
#define  BOUT BIN
#define  BINIT    FIN_LOCAL_FIELD(_op_last_line_passed) = 0; _op_block_origin = __LINE__;
#else
#define  BINIT
#endif /* #if !defined (VOSD_NO_FIN) */



/* State variable definitions */
typedef struct
    {
    /* Internal state tracking for FSM */
    FSM_SYS_STATE
    /* State Variables */
    int                             pk_total_out                                    ;  /*
total number of packets out of node */
    int                             pk_total_in                                     ;  /*
total number of packet into this node */
    int                             pk_ping_out                                     ;  /*
total number of pings out of the node */
    int                             pk_ping_in                                      ;  /*
total number of pings into this node */
    int                             pk_pong_out                                     ;  /*
Pong used to replies ping. fwd pong does not include */
    int                             pk_pong_in                                      ;  /*
Pong with node_id==node_objid */
    int                             pk_q_out                                        ;  /*
total number of query out of node */
    int                             pk_q_in                                         ;  /*
total query into this node */
    int                             pk_qh_out                                       ;  /*
query hit generated by this node. fwd query hit does not include */
    int                             pk_qh_in                                        ;  /*
query hit with node_id==node_objid */
    msg_cache                       ping_cache[CACHE_SIZE]                          ;  /*
used to save received ping and gerenated ping */
    msg_cache                       pong_cache[CACHE_SIZE]                          ;  /*
used to save received pong */
    int                             ava_data                                        ;  /*
data to be searched, only used when developing the project */
    int                             ping_ptr                                        ;  /*
ping pointer used in cache */
```

```
    int                             pong_ptr                                    ;   /*
pong pointer used in pong cache */
    Stathandle                      pk_total_out_stat                           ;
    Stathandle                      pk_total_in_stat                            ;
    Stathandle                      pk_ping_out_stat                            ;
    Stathandle                      pk_ping_in_stat                             ;
    Stathandle                      pk_pong_out_stat                            ;
    Stathandle                      pk_pong_in_stat                             ;
    Stathandle                      pk_q_out_stat                               ;
    Stathandle                      pk_q_in_stat                                ;
    Stathandle                      pk_qh_out_stat                              ;
    Stathandle                      pk_qh_in_stat                               ;
    int                             node_objid                                  ;   /*
the id of this node */
    int                             src_packet_id[CACHE_SIZE]                   ;   /*
used to track which source ping send only used when developing the project */
    int                             src_ping_ptr                                ;
    int                             data_pool[20]                               ;   /*
data that can be searched in this node */
    int                             pk_q_own_out                                ;   /*
query generated by this node */
    Stathandle                      pk_q_own_out_stat                           ;
    msg_cache                       query_cache[CACHE_SIZE]                     ;   /*
used to save received Query */
    int                             query_ptr                                   ;   /*
query pointer used in query cache */
    } liu_testgnutella_node_proc_comment_code_state;

#define pk_total_out                op_sv_ptr->pk_total_out
#define pk_total_in                 op_sv_ptr->pk_total_in
#define pk_ping_out                 op_sv_ptr->pk_ping_out
#define pk_ping_in                  op_sv_ptr->pk_ping_in
#define pk_pong_out                 op_sv_ptr->pk_pong_out
#define pk_pong_in                  op_sv_ptr->pk_pong_in
#define pk_q_out                    op_sv_ptr->pk_q_out
#define pk_q_in                     op_sv_ptr->pk_q_in
#define pk_qh_out                   op_sv_ptr->pk_qh_out
#define pk_qh_in                    op_sv_ptr->pk_qh_in
#define ping_cache                  op_sv_ptr->ping_cache
#define pong_cache                  op_sv_ptr->pong_cache
#define ava_data                    op_sv_ptr->ava_data
#define ping_ptr                    op_sv_ptr->ping_ptr
#define pong_ptr                    op_sv_ptr->pong_ptr
#define pk_total_out_stat           op_sv_ptr->pk_total_out_stat
```

```
#define pk_total_in_stat            op_sv_ptr->pk_total_in_stat
#define pk_ping_out_stat            op_sv_ptr->pk_ping_out_stat
#define pk_ping_in_stat             op_sv_ptr->pk_ping_in_stat
#define pk_pong_out_stat            op_sv_ptr->pk_pong_out_stat
#define pk_pong_in_stat             op_sv_ptr->pk_pong_in_stat
#define pk_q_out_stat               op_sv_ptr->pk_q_out_stat
#define pk_q_in_stat                op_sv_ptr->pk_q_in_stat
#define pk_qh_out_stat              op_sv_ptr->pk_qh_out_stat
#define pk_qh_in_stat               op_sv_ptr->pk_qh_in_stat
#define node_objid                  op_sv_ptr->node_objid
#define src_packet_id               op_sv_ptr->src_packet_id
#define src_ping_ptr                op_sv_ptr->src_ping_ptr
#define data_pool                   op_sv_ptr->data_pool
#define pk_q_own_out                op_sv_ptr->pk_q_own_out
#define pk_q_own_out_stat           op_sv_ptr->pk_q_own_out_stat
#define query_cache                 op_sv_ptr->query_cache
#define query_ptr                   op_sv_ptr->query_ptr

/* These macro definitions will define a local variable called   */
/* "op_sv_ptr" in each function containing a FIN statement.  */
/* This variable points to the state variable data structure,    */
/* and can be used from a C debugger to display their values.    */
#undef FIN_PREAMBLE_DEC
#undef FIN_PREAMBLE_CODE
#define FIN_PREAMBLE_DEC    liu_testgnutella_node_proc_comment_code_state *op_sv_ptr;
#define FIN_PREAMBLE_CODE   \
        op_sv_ptr = ((liu_testgnutella_node_proc_comment_code_state
*)(OP_SIM_CONTEXT_PTR->_op_mod_state_ptr));


/* Function Block */

#if !defined (VOSD_NO_FIN)
enum { _op_block_origin = __LINE__ + 2};
#endif

static void transmit_src_pk(void)
    {
        Packet* pkptr_0;
        Packet* pkptr_1;
        Packet* pkptr_2;
        Packet* pkptr_3;
        Packet* pkptr_4;
        Packet* pkptr_5;
```

```
OpT_Packet_Id pk_id;

FIN(transmit_src_pk())
//assign proper value to each field
pkptr_0=op_pk_get(src_in);
pk_id=op_pk_id(pkptr_0);
op_pk_nfd_set_pkid (pkptr_0, "packet_id", pk_id);
op_pk_nfd_set_int32 (pkptr_0, "Payload Descriptor", PING);
op_pk_nfd_set_int32 (pkptr_0, "TTL", TTL_INIT);
op_pk_nfd_set_int32 (pkptr_0, "Hops", 0);
op_pk_nfd_set_int32 (pkptr_0,"node_id", node_objid);
op_pk_nfd_set_int32 (pkptr_0, "dest_addr", xmt0_out);
op_pk_nfd_set_int32 (pkptr_0,"search",0);
op_pk_nfd_set_pkid (pkptr_0,"pong_qh_parentID",-1);
op_prg_odb_bkpt ("lll");

//copy this packet 5 times
pkptr_1=op_pk_create_fmt("gnutella_simple_pk_format");
op_pk_nfd_set_pkid (pkptr_1, "packet_id", pk_id);
op_pk_nfd_set_int32 (pkptr_1, "Payload Descriptor", PING);
op_pk_nfd_set_int32 (pkptr_1, "TTL", TTL_INIT);
op_pk_nfd_set_int32 (pkptr_1, "Hops", 0);
op_pk_nfd_set_int32 (pkptr_1,"node_id", node_objid);
op_pk_nfd_set_int32 (pkptr_1, "dest_addr", xmt1_out);
op_pk_nfd_set_int32 (pkptr_1,"search",0);
op_pk_nfd_set_pkid (pkptr_1,"pong_qh_parentID",-1);

pkptr_2=op_pk_create_fmt("gnutella_simple_pk_format");
op_pk_nfd_set_pkid (pkptr_2, "packet_id", pk_id);
op_pk_nfd_set_int32 (pkptr_2, "Payload Descriptor", PING);
op_pk_nfd_set_int32 (pkptr_2, "TTL", TTL_INIT);
op_pk_nfd_set_int32 (pkptr_2, "Hops", 0);
op_pk_nfd_set_int32 (pkptr_2,"node_id", node_objid);
op_pk_nfd_set_int32 (pkptr_2, "dest_addr", xmt2_out);
op_pk_nfd_set_int32 (pkptr_2,"search",0);
op_pk_nfd_set_pkid (pkptr_2,"pong_qh_parentID",-1);

pkptr_3=op_pk_create_fmt("gnutella_simple_pk_format");
op_pk_nfd_set_pkid (pkptr_3, "packet_id", pk_id);
op_pk_nfd_set_int32 (pkptr_3, "Payload Descriptor", PING);
op_pk_nfd_set_int32 (pkptr_3, "TTL", TTL_INIT);
op_pk_nfd_set_int32 (pkptr_3, "Hops", 0);
op_pk_nfd_set_int32 (pkptr_3,"node_id", node_objid);
op_pk_nfd_set_int32 (pkptr_3, "dest_addr", xmt3_out);
```

```
op_pk_nfd_set_int32 (pkptr_3,"search",0);
op_pk_nfd_set_pkid (pkptr_3,"pong_qh_parentID",-1);

pkptr_4=op_pk_create_fmt("gnutella_simple_pk_format");
op_pk_nfd_set_pkid (pkptr_4, "packet_id", pk_id);
op_pk_nfd_set_int32 (pkptr_4, "Payload Descriptor", PING);
op_pk_nfd_set_int32 (pkptr_4, "TTL", TTL_INIT);
op_pk_nfd_set_int32 (pkptr_4, "Hops", 0);
op_pk_nfd_set_int32 (pkptr_4,"node_id", node_objid);
op_pk_nfd_set_int32 (pkptr_4, "dest_addr", xmt4_out);
op_pk_nfd_set_int32 (pkptr_4,"search",0);
op_pk_nfd_set_pkid (pkptr_4,"pong_qh_parentID",-1);

pkptr_5=op_pk_create_fmt("gnutella_simple_pk_format");
op_pk_nfd_set_pkid (pkptr_5, "packet_id", pk_id);
op_pk_nfd_set_int32 (pkptr_5, "Payload Descriptor", PING);
op_pk_nfd_set_int32 (pkptr_5, "TTL", TTL_INIT);
op_pk_nfd_set_int32 (pkptr_5, "Hops", 0);
op_pk_nfd_set_int32 (pkptr_5,"node_id", node_objid);
op_pk_nfd_set_int32 (pkptr_5, "dest_addr", xmt5_out);
op_pk_nfd_set_int32 (pkptr_5,"search",0);
op_pk_nfd_set_pkid (pkptr_5,"pong_qh_parentID",-1);

//send ping packets to transmitter
op_pk_send(pkptr_0,xmt0_out);
op_pk_send(pkptr_1,xmt1_out);
op_pk_send(pkptr_2,xmt2_out);
op_pk_send(pkptr_3,xmt3_out);
op_pk_send(pkptr_4,xmt4_out);
op_pk_send(pkptr_5,xmt5_out);

int i=0;
for (i=0;i<=5;i++)
    {
        pk_total_out++;
        pk_ping_out++;
        }

//save ping to cache
ping_cache[ping_ptr].packet_id=(int)pk_id;
ping_cache[ping_ptr].Hops=0;
ping_cache[ping_ptr].node_id=(int)node_objid;
ping_cache[ping_ptr].dest_addr=0;
ping_cache[ping_ptr].Payload_des=PING;
```

```
        ping_cache[ping_ptr].search=0;
        ping_ptr++;
        if(ping_ptr>=CACHE_SIZE)
            {
                ping_ptr=0;
            }

        //save source ping to cahce. only used when developing
        src_packet_id[src_ping_ptr]=(int)pk_id;
        src_ping_ptr++;
        if(src_ping_ptr>=CACHE_SIZE)
            {
                src_ping_ptr=0;
            }
        //write statistics
        op_stat_write(pk_total_out_stat,pk_total_out);
        op_stat_write(pk_total_in_stat,pk_total_in);
        op_stat_write(pk_ping_out_stat,pk_ping_out);
        op_stat_write(pk_ping_in_stat,pk_ping_in);
        op_stat_write(pk_pong_out_stat,pk_pong_out);
        op_stat_write(pk_pong_in_stat,pk_pong_in);
        op_stat_write(pk_q_out_stat,pk_q_out);
        op_stat_write(pk_q_in_stat,pk_q_in);
        op_stat_write(pk_qh_out_stat,pk_qh_out);
        op_stat_write(pk_qh_in_stat,pk_qh_in);

        FOUT;

    }

/* End of Function Block */

/* Undefine optional tracing in FIN/FOUT/FRET */
/* The FSM has its own tracing code and the other */
/* functions should not have any tracing.        */
#undef FIN_TRACING
#define FIN_TRACING

#undef FOUTRET_TRACING
#define FOUTRET_TRACING

#if defined (__cplusplus)
extern "C" {
#endif
```

```
    void liu_testgnutella_node_proc_comment_code (OP_SIM_CONTEXT_ARG_OPT);
    VosT_Obtype _op_liu_testgnutella_node_proc_comment_code_init (int * init_block_ptr);
    void _op_liu_testgnutella_node_proc_comment_code_diag (OP_SIM_CONTEXT_ARG_OPT);
    void _op_liu_testgnutella_node_proc_comment_code_terminate (OP_SIM_CONTEXT_ARG_OPT);
    VosT_Address _op_liu_testgnutella_node_proc_comment_code_alloc (VosT_Obtype, int);
    void _op_liu_testgnutella_node_proc_comment_code_svar (void *, const char *, void **);


#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif




/* Process model interrupt handling procedure */


void
liu_testgnutella_node_proc_comment_code (OP_SIM_CONTEXT_ARG_OPT)
    {
#if !defined (VOSD_NO_FIN)
    int _op_block_origin = 0;
#endif
    FIN_MT (liu_testgnutella_node_proc_comment_code ());

        {


        FSM_ENTER ("liu_testgnutella_node_proc_comment_code")

        FSM_BLOCK_SWITCH
            {
            /*---------------------------------------------------------*/
            /** state (init) enter executives **/
            FSM_STATE_ENTER_FORCED_NOLABEL (0, "init",
"liu_testgnutella_node_proc_comment_code [init enter execs]")
                FSM_PROFILE_SECTION_IN ("liu_testgnutella_node_proc_comment_code [init
enter execs]", state0_enter_exec)
                {
                //initialize state variables
                pk_total_out =0;
                pk_total_in=0;
                pk_ping_out=0;
```

```
                    pk_ping_in =0;
                    pk_pong_out=0;
                    pk_pong_in=0;
                    pk_q_out=0;
                    pk_q_in =0;
                    pk_qh_out=0;
                    pk_qh_in =0;
                    pk_q_own_out=0     ;
                    //ava_data=0;
                    ping_ptr=0;
                    pong_ptr=0;
                    query_ptr=0;
                    src_ping_ptr=0;
                    int j=0;
                    for (j=0;j++;j<CACHE_SIZE)
                         {
                              ping_cache[j].packet_id=-1;
                              pong_cache[j].packet_id=-1;
                              query_cache[j].packet_id=-1;
                              src_packet_id[j]=-1;
                         }

                    //generate node_objid
                    node_objid = (int)op_topo_parent (op_id_self());

                    //generate data pool for searching
                    for(j=0;j++;j<20)
                         {

     data_pool[j]=(int)op_dist_outcome(op_dist_load("uniform_int",0,4));
                         }

                    //statistic register
                    pk_total_out_stat  = op_stat_reg ("packet total out", OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
                    pk_total_in_stat = op_stat_reg ("packet total in",
OPC_STAT_INDEX_NONE,OPC_STAT_LOCAL);
                    pk_ping_out_stat  = op_stat_reg ("pk ping out", OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
                    pk_ping_in_stat = op_stat_reg ("pk ping in",
OPC_STAT_INDEX_NONE,OPC_STAT_LOCAL);
                    pk_pong_out_stat  = op_stat_reg ("pk pong out", OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
                    pk_pong_in_stat = op_stat_reg ("pk pong in",
```

```
OPC_STAT_INDEX_NONE,OPC_STAT_LOCAL);
                    pk_q_out_stat= op_stat_reg ("pk q out", OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
                    pk_q_in_stat = op_stat_reg ("pk q in", OPC_STAT_INDEX_NONE,OPC_STAT_LOCAL);
                    pk_qh_out_stat= op_stat_reg ("pk qh out", OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
                    pk_qh_in_stat = op_stat_reg ("pk qh in",
OPC_STAT_INDEX_NONE,OPC_STAT_LOCAL);
                    pk_q_own_out_stat = op_stat_reg ("pk q own out",
OPC_STAT_INDEX_NONE,OPC_STAT_LOCAL);

                    //write the initial values to statistics
                    op_stat_write(pk_total_out_stat,pk_total_out);
                    op_stat_write(pk_total_in_stat,pk_total_in);
                    op_stat_write(pk_ping_out_stat,pk_ping_out);
                    op_stat_write(pk_ping_in_stat,pk_ping_in);
                    op_stat_write(pk_pong_out_stat,pk_pong_out);
                    op_stat_write(pk_pong_in_stat,pk_pong_in);
                    op_stat_write(pk_q_out_stat,pk_q_out);
                    op_stat_write(pk_q_in_stat,pk_q_in);
                    op_stat_write(pk_qh_out_stat,pk_qh_out);
                    op_stat_write(pk_qh_in_stat,pk_qh_in);
                    op_stat_write(pk_q_own_out_stat,pk_q_own_out);
                    }
                    FSM_PROFILE_SECTION_OUT (state0_enter_exec)

               /** state (init) exit executives **/
               FSM_STATE_EXIT_FORCED (0, "init", "liu_testgnutella_node_proc_comment_code [init
exit execs]")


               /** state (init) transition processing **/
               FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "init", "idle", "tr_0",
"liu_testgnutella_node_proc_comment_code [init -> idle : default / ]")
                    /*------------------------------------------------------------*/



               /** state (idle) enter executives **/
               FSM_STATE_ENTER_UNFORCED (1, "idle", state1_enter_exec,
"liu_testgnutella_node_proc_comment_code [idle enter execs]")

               /** blocking after enter executives of unforced state. **/
               FSM_EXIT (3,"liu_testgnutella_node_proc_comment_code")
```

```
/** state (idle) exit executives **/
FSM_STATE_EXIT_UNFORCED (1, "idle", "liu_testgnutella_node_proc_comment_code
[idle exit execs]")


/** state (idle) transition processing **/
FSM_PROFILE_SECTION_IN ("liu_testgnutella_node_proc_comment_code [idle trans
conditions]", state1_trans_conds)
FSM_INIT_COND (RCV_ARRIVAL)
FSM_TEST_COND (SRC_ARRIVAL)
FSM_DFLT_COND
FSM_TEST_LOGIC ("idle")
FSM_PROFILE_SECTION_OUT (state1_trans_conds)

FSM_TRANSIT_SWITCH
    {
    FSM_CASE_TRANSIT (0, 2, state2_enter_exec, ;, "RCV_ARRIVAL", "", "idle",
"proc", "tr_9", "liu_testgnutella_node_proc_comment_code [idle -> proc : RCV_ARRIVAL / ]")
    FSM_CASE_TRANSIT (1, 1, state1_enter_exec, transmit_src_pk();,
"SRC_ARRIVAL", "transmit_src_pk()", "idle", "idle", "tr_1",
"liu_testgnutella_node_proc_comment_code [idle -> idle : SRC_ARRIVAL / transmit_src_pk()]")
    FSM_CASE_TRANSIT (2, 1, state1_enter_exec, ;, "default", "", "idle", "idle",
"tr_2", "liu_testgnutella_node_proc_comment_code [idle -> idle : default / ]")
    }
    /*-------------------------------------------------------*/



/** state (proc) enter executives **/
FSM_STATE_ENTER_FORCED (2, "proc", state2_enter_exec,
"liu_testgnutella_node_proc_comment_code [proc enter execs]")
    FSM_PROFILE_SECTION_IN ("liu_testgnutella_node_proc_comment_code [proc
enter execs]", state2_enter_exec)
    {
    //input packet field[Universal]
    OpT_Packet_Id o_pk_id;//correspond to packet_id in the packet
    int ttl;
    int hops;
    int node_id;
    int dest_addr;
    OpT_Packet_Id sender_node_id;
    int payload_des;
```

```
int search;

//for generated pong[1st if]
Packet* pong_pkptr;
OpT_Packet_Id pong_o_pk_id;

//used for generating query[2nd if]
Packet* q_pkptr;
OpT_Packet_Id q_o_pk_id;



//used for generating queryhit[3rd if]
OpT_Packet_Id Query_hit_o_pk_id;
Packet* qh_pkptr;


//packet pointer used for fwd ping pong etc.[Universal]
Packet* pkptr_1;
Packet* pkptr_2;
Packet* pkptr_3;
Packet* pkptr_4;
Packet* pkptr_5;

//input receiver number[Universal]
int input_rcv_num;

//FLAG.used for check whether it is a duplicated ping[1st if]
int duplicate_ping;
//FLAG.used to check whether it is a duplicated pong[2st if]
int duplicate_pong;
//FLAG.uesd to detect wether this node sent a parent packet.[2nd if]
int is_parent_flag;
//FLAG. check searching data
int search_hit_flag;
//incoming pakcet's pointer[Universal]
Packet* pkptr;
//-----------------------------------------------------------------------


//get packet from receiver
input_rcv_num=op_intrpt_strm();
pkptr=op_pk_get(input_rcv_num);
```

```
//read payload_descriptor
op_pk_nfd_get_int32(pkptr,"Payload Descriptor",&payload_des);

//check which command it is
if (payload_des==1)//PING
     {
          pk_total_in++;
          pk_ping_in++;

          //read all the information from packet
          op_pk_nfd_get_pkid(pkptr,"packet_id",&o_pk_id);
          op_pk_nfd_get_int32(pkptr,"TTL",&ttl);
          op_pk_nfd_get_int32(pkptr,"Hops",&hops);
          op_pk_nfd_get_int32(pkptr,"node_id",&node_id);
          op_pk_nfd_get_int32(pkptr,"dest_addr",&dest_addr);
          op_pk_nfd_get_pkid(pkptr,"sender_node_id",&sender_node_id);
          op_pk_nfd_get_int32(pkptr,"search",&search);

          //if this is duplicated ping.destroy packet.set flag duplicate_ping
          duplicate_ping=0;
          int i=0;
          while((duplicate_ping==0) && (i<=CACHE_SIZE))
               {
                    if((int)o_pk_id==ping_cache[i].packet_id)
                         {duplicate_ping=1;}
                    i=i+1;
               }
          if (duplicate_ping==1)
               {
                    op_prg_odb_bkpt ("pk0");
                    op_pk_destroy(pkptr);
               }
          else if(duplicate_ping==0)//it is not duplicated ping
               {
                    //generate pong packet
                  pong_pkptr =
op_pk_create_fmt("gnutella_simple_pk_format");
                    pong_o_pk_id=op_pk_id(pong_pkptr);

                    //assign each field a proper value
                  op_pk_nfd_set_pkid (pong_pkptr,"packet_id", pong_o_pk_id);

                    op_prg_odb_bkpt ("1");
                  op_pk_nfd_set_int32 (pong_pkptr,"Payload Descriptor",
```

```
PONG);
                                    op_pk_nfd_set_int32 (pong_pkptr,"TTL", TTL_INIT);
//TTL=HOPS+1
                                    op_pk_nfd_set_int32 (pong_pkptr,"Hops", 0);
                                    op_pk_nfd_set_int32 (pong_pkptr,"node_id", node_id);
                                      op_pk_nfd_set_pkid (pong_pkptr,"sender_node_id",
o_pk_id);
                                      op_prg_odb_bkpt ("2");
                                      op_pk_nfd_set_int32(pong_pkptr,"dest_addr",dest_addr);
                                      op_pk_nfd_set_int32 (pong_pkptr,"search", 0);

                                      //reply ping by using pong
                                      if(input_rcv_num==rcv0_in)
                                           {op_pk_send(pong_pkptr,xmt0_out);}
                                      else if(input_rcv_num==rcv1_in)
                                           {op_pk_send(pong_pkptr,xmt1_out);}
                                      else if(input_rcv_num==rcv2_in)
                                           {op_pk_send(pong_pkptr,xmt2_out);}
                                      else if(input_rcv_num==rcv3_in)
                                           {op_pk_send(pong_pkptr,xmt3_out);}
                                      else if(input_rcv_num==rcv4_in)
                                           {op_pk_send(pong_pkptr,xmt4_out);}
                                      else if(input_rcv_num==rcv5_in)
                                           {op_pk_send(pong_pkptr,xmt5_out);}
                                      pk_pong_out++;
                                      pk_total_out++;

                                      //save new ping
                                      ping_cache[ping_ptr].packet_id=(int)o_pk_id;
                                      ping_ptr++;
                                      if(ping_ptr>=CACHE_SIZE)
                                           {
                                                ping_ptr=0;
                                           }

                                      //if ttl is greater than zero, update ttl, hops, dest_addr
and fwd ping
                                      if(ttl>0)
                                           {
                                                //update ttl hops dest_addr
                                                int new_dest_addr;
                                                new_dest_addr=dest_addr*10+input_rcv_num;
                                                op_pk_nfd_set_int32 (pkptr, "dest_addr",
new_dest_addr);
```

```
                                        op_pk_nfd_set_int32(pkptr,"TTL",ttl-1);
                                        op_pk_nfd_set_int32(pkptr,"Hops",hops+1);

                                        //copy packet 4 times

    pkptr_1=op_pk_create_fmt("gnutella_simple_pk_format");
                                        op_pk_nfd_set_pkid (pkptr_1, "packet_id",
o_pk_id);
                                        op_prg_odb_bkpt ("3");

                                        op_pk_nfd_set_int32 (pkptr_1, "Payload
Descriptor", PING);
                                        op_pk_nfd_set_int32 (pkptr_1, "TTL", ttl-1);
                                        op_pk_nfd_set_int32 (pkptr_1, "Hops", hops+1);
                                        op_pk_nfd_set_int32 (pkptr_1,"node_id",
node_id);
                                        op_pk_nfd_set_int32 (pkptr_1, "dest_addr",
new_dest_addr);
                                       op_pk_nfd_set_int32 (pkptr_1,"search",search);

                                        op_pk_nfd_set_pkid
(pkptr_1,"sender_node_id",sender_node_id);
                                        op_prg_odb_bkpt ("4");


    pkptr_2=op_pk_create_fmt("gnutella_simple_pk_format");
                                        op_pk_nfd_set_pkid (pkptr_2, "packet_id",
o_pk_id);
                                        op_pk_nfd_set_int32 (pkptr_2, "Payload
Descriptor", PING);
                                        op_pk_nfd_set_int32 (pkptr_2, "TTL", ttl-1);
                                        op_pk_nfd_set_int32 (pkptr_2, "Hops", hops+1);
                                        op_pk_nfd_set_int32 (pkptr_2,"node_id",
node_id);
                                        op_pk_nfd_set_int32 (pkptr_2, "dest_addr",
new_dest_addr);
                                        op_pk_nfd_set_int32
(pkptr_2,"search",search);
                                        op_pk_nfd_set_pkid
(pkptr_2,"sender_node_id",sender_node_id);


    pkptr_3=op_pk_create_fmt("gnutella_simple_pk_format");
                                        op_pk_nfd_set_pkid (pkptr_3, "packet_id",
```

```
o_pk_id);
                                        op_pk_nfd_set_int32 (pkptr_3, "Payload
Descriptor", PING);
                                        op_pk_nfd_set_int32 (pkptr_3, "TTL", ttl-1);
                                        op_pk_nfd_set_int32 (pkptr_3, "Hops", hops+1);
                                        op_pk_nfd_set_int32 (pkptr_3,"node_id",
node_id);
                                        op_pk_nfd_set_int32 (pkptr_3, "dest_addr",
new_dest_addr);
                                        op_pk_nfd_set_int32
(pkptr_3,"search",search);
                                        op_pk_nfd_set_pkid
(pkptr_3,"sender_node_id",sender_node_id);


    pkptr_4=op_pk_create_fmt("gnutella_simple_pk_format");
                                        op_pk_nfd_set_pkid (pkptr_4, "packet_id",
o_pk_id);
                                        op_pk_nfd_set_int32 (pkptr_4, "Payload
Descriptor", PING);
                                        op_pk_nfd_set_int32 (pkptr_4, "TTL", ttl-1);
                                        op_pk_nfd_set_int32 (pkptr_4, "Hops", hops+1);
                                        op_pk_nfd_set_int32 (pkptr_4,"node_id",
node_id);
                                        op_pk_nfd_set_int32 (pkptr_4, "dest_addr",
new_dest_addr);
                                        op_pk_nfd_set_int32
(pkptr_4,"search",search);
                                        op_pk_nfd_set_pkid
(pkptr_4,"sender_node_id",sender_node_id);

                                        //fwd ping
                                        if(input_rcv_num==rcv0_in)
                                            {op_pk_send(pkptr,xmt1_out);
                                            op_pk_send(pkptr_1,xmt2_out);
                                            op_pk_send(pkptr_2,xmt3_out);
                                            op_pk_send(pkptr_3,xmt4_out);
                                            op_pk_send(pkptr_4,xmt5_out);}
                                        else if(input_rcv_num==rcv1_in)
                                            {op_pk_send(pkptr,xmt0_out);
                                            op_pk_send(pkptr_1,xmt2_out);
                                            op_pk_send(pkptr_2,xmt3_out);
                                            op_pk_send(pkptr_3,xmt4_out);
                                            op_pk_send(pkptr_4,xmt5_out);}
```

```
                    else if(input_rcv_num==rcv2_in)
                        {op_pk_send(pkptr,xmt1_out);
                        op_pk_send(pkptr_1,xmt0_out);
                        op_pk_send(pkptr_2,xmt3_out);
                        op_pk_send(pkptr_3,xmt4_out);
                        op_pk_send(pkptr_4,xmt5_out);}
                    else if(input_rcv_num==rcv3_in)
                        {op_pk_send(pkptr,xmt1_out);
                        op_pk_send(pkptr_1,xmt2_out);
                        op_pk_send(pkptr_2,xmt0_out);
                        op_pk_send(pkptr_3,xmt4_out);
                        op_pk_send(pkptr_4,xmt5_out);}
                    else if(input_rcv_num==rcv4_in)
                        {op_pk_send(pkptr,xmt1_out);
                        op_pk_send(pkptr_1,xmt2_out);
                        op_pk_send(pkptr_2,xmt3_out);
                        op_pk_send(pkptr_3,xmt0_out);
                        op_pk_send(pkptr_4,xmt5_out);}
                    else if(input_rcv_num==rcv5_in)
                        {op_pk_send(pkptr,xmt1_out);
                        op_pk_send(pkptr_1,xmt2_out);
                        op_pk_send(pkptr_2,xmt3_out);
                        op_pk_send(pkptr_3,xmt4_out);
                        op_pk_send(pkptr_4,xmt0_out);}
                    pk_total_out=pk_total_out+5;
                    pk_ping_out=pk_ping_out+5;
                }
            else//ttl<=0
                {
                op_pk_destroy(pkptr);
                }

        }

    }
else if (payload_des==2)//Pong
    {
        pk_total_in++;

        //read all the information from packet
        op_pk_nfd_get_pkid(pkptr,"packet_id",&o_pk_id);
        op_pk_nfd_get_int32(pkptr,"TTL",&ttl);
        op_pk_nfd_get_int32(pkptr,"Hops",&hops);
        op_pk_nfd_get_int32(pkptr,"node_id",&node_id);
```

```
op_pk_nfd_get_int32(pkptr,"dest_addr",&dest_addr);
op_pk_nfd_get_pkid(pkptr,"sender_node_id",&sender_node_id);
op_pk_nfd_get_int32(pkptr,"search",&search);

//check if it is a duplicated pong and set flag duplicate_pong
duplicate_pong=0;
int i=0;
while((duplicate_pong==0) && (i<CACHE_SIZE))
    {
        if((int)o_pk_id==pong_cache[i].packet_id)
            {duplicate_pong=1;}
        i=i+1;
    }

//if it is duplicated, destroy it.
if (duplicate_pong==1)
    {
        op_prg_odb_bkpt ("02");
        op_pk_destroy(pkptr);
    }
else //else fwd pong if ttl ok, save pong.detect node_id in pong,
if it matches
     //node_objid, send query, save query
    {
        //save new pong
        pong_cache[pong_ptr].packet_id=(int)o_pk_id;
        pong_ptr++;
        if(pong_ptr>=CACHE_SIZE)
            {
                pong_ptr=0;
            }

        //check if node_id from packet is equal to node_objid
        is_parent_flag=0;
        if(node_id==node_objid)
            {
                is_parent_flag=1;
                op_prg_odb_bkpt ("03");
                pk_pong_in++;
                op_pk_destroy(pkptr);
            }
        else if(ttl>0)
            {
                //continue sending pong through decoded port
```

```
                                        //update ttl and hops
                                        op_pk_nfd_set_int32(pkptr,"TTL",ttl-1);
                                        op_pk_nfd_set_int32(pkptr,"Hops",hops+1);

                                        //decode dest_addr
                                        int decoded_send_port;
                                        decoded_send_port=dest_addr%10;

                                        //update dest_addr
                                        op_pk_nfd_set_int32 (pkptr, "dest_addr",
(dest_addr-decoded_send_port)/10);

                                        //continue send pong through decoded port
                                        op_prg_odb_bkpt ("04");
                                        op_pk_send(pkptr,decoded_send_port-1);
                                        op_prg_odb_bkpt ("05");

                                        //write statistics
                                        pk_total_out++;
                                    }
                                else
                                    {
                                        op_pk_destroy(pkptr);
                                    }

                                if (is_parent_flag==1)//the case node_id==node_objid
                                    {
                                        //generate query packet

    q_pkptr=op_pk_create_fmt("gnutella_simple_pk_format");
                                        q_o_pk_id=op_pk_id(q_pkptr);
                                        op_pk_nfd_set_pkid (q_pkptr, "packet_id",
q_o_pk_id);
                                        op_pk_nfd_set_int32 (q_pkptr, "Payload
Descriptor", Query);
                                        op_pk_nfd_set_int32 (q_pkptr, "TTL",
TTL_INIT);
                                        op_pk_nfd_set_int32 (q_pkptr, "Hops", 0);
                                        op_pk_nfd_set_int32 (q_pkptr,
"dest_addr",input_rcv_num);
                                        op_pk_nfd_set_int32
(q_pkptr,"search",(int)op_dist_outcome(op_dist_load("uniform_int",0,2)));//data to be
searched is uniformly distributed int from 0 to 2
```

```
                                        op_prg_odb_bkpt ("12");
                                        op_pk_nfd_set_pkid
(q_pkptr,"sender_node_id",-1);
                                        op_pk_nfd_set_int32
(q_pkptr,"node_id",node_id);
                                        op_prg_odb_bkpt ("7");

                                        //save new query

    query_cache[query_ptr].packet_id=(int)q_o_pk_id;
                                        query_ptr++;
                                        if(query_ptr>=CACHE_SIZE)
                                             {query_ptr=0;}

                                        //send query packet
                                        if(input_rcv_num==rcv0_in)
                                             {op_pk_send(q_pkptr,xmt0_out);}
                                        else if(input_rcv_num==rcv1_in)
                                             {op_pk_send(q_pkptr,xmt1_out);}
                                        else if(input_rcv_num==rcv2_in)
                                             {op_pk_send(q_pkptr,xmt2_out);}
                                        else if(input_rcv_num==rcv3_in)
                                             {op_pk_send(q_pkptr,xmt3_out);}
                                        else if(input_rcv_num==rcv4_in)
                                             {op_pk_send(q_pkptr,xmt4_out);}
                                        else if(input_rcv_num==rcv5_in)
                                             {op_pk_send(q_pkptr,xmt5_out);}
                                        pk_total_out++;
                                        pk_q_out++;
                                        pk_q_own_out++;
                                   }
                              }
                         }
                    else if (payload_des==4)//Query
                         {

                         pk_total_in++;
                         pk_q_in++;
                         //read all the information from the packet
                              op_pk_nfd_get_pkid(pkptr,"packet_id",&o_pk_id);
                              op_pk_nfd_get_int32(pkptr,"TTL",&ttl);
                              op_pk_nfd_get_int32(pkptr,"Hops",&hops);
                              op_pk_nfd_get_int32(pkptr,"node_id",&node_id);
                              op_pk_nfd_get_int32(pkptr,"dest_addr",&dest_addr);
```

```
                    op_pk_nfd_get_pkid(pkptr,"sender_node_id",&sender_node_id);
                    op_pk_nfd_get_int32(pkptr,"search",&search);

                //check if it is duplicated query,if it is , destroy it, if not, save
it in cache
                  int i=0;
                  int query_dup=0;
                    while(query_dup==0 && i<CACHE_SIZE)
                    {
                        if(query_cache[i].packet_id==o_pk_id)
                            {op_prg_odb_bkpt ("qq");
                            query_dup=1;}
                        i=i+1;
                }
                  if(query_dup==1)
                      {op_pk_destroy(pkptr);}
                  else
                      {
                        query_cache[query_ptr].packet_id=o_pk_id;
                        query_ptr++;
                      }
                  if(query_ptr>=CACHE_SIZE)
                        {query_ptr=0;}

                  //if it is not duplicated query, search data pool.
                  if(query_dup==0)
                      {
                        search_hit_flag=0;
                        i=0;
                            op_prg_odb_bkpt ("13");
                            while(search_hit_flag==0 && i<20)
                                {
                                    if (data_pool[i] ==search)
                                        {search_hit_flag=1;}
                                    i=i+1;

                                }
                    if (search_hit_flag==1)//data is found
                            {
                                    //generate queryhit packet

    qh_pkptr=op_pk_create_fmt("gnutella_simple_pk_format");
                                    Query_hit_o_pk_id=op_pk_id(qh_pkptr);
                                    op_pk_nfd_set_pkid (qh_pkptr, "packet_id",
```

```
Query_hit_o_pk_id);
                                        op_pk_nfd_set_int32 (qh_pkptr, "Payload
Descriptor", Query_hit);
                                        op_pk_nfd_set_int32 (qh_pkptr, "TTL",
TTL_INIT);
                                        op_pk_nfd_set_int32 (qh_pkptr, "Hops", 0);
                                        op_pk_nfd_set_int32 (qh_pkptr,"node_id",
node_id);
                                        op_pk_nfd_set_int32 (qh_pkptr, "dest_addr",
dest_addr);
                                        op_pk_nfd_set_int32 (qh_pkptr,"search",0);
                                        op_pk_nfd_set_pkid
(qh_pkptr,"sender_node_id",o_pk_id);

                                        //send queryhit packet
                                        if(input_rcv_num==rcv0_in)
                                             {op_pk_send(qh_pkptr,xmt0_out);}
                                        else if(input_rcv_num==rcv1_in)
                                             {op_pk_send(qh_pkptr,xmt1_out);}
                                        else if(input_rcv_num==rcv2_in)
                                             {op_pk_send(qh_pkptr,xmt2_out);}
                                        else if(input_rcv_num==rcv3_in)
                                             {op_pk_send(qh_pkptr,xmt3_out);}
                                        else if(input_rcv_num==rcv4_in)
                                             {op_pk_send(qh_pkptr,xmt4_out);}
                                        else if(input_rcv_num==rcv5_in)
                                             {op_pk_send(qh_pkptr,xmt5_out);}
                                        pk_total_out++;
                                        pk_qh_out++;
                                   }
                        //fwd query
                        //if ttl is ok, update ttl, hops, dest_addr and fwd query
                        if(ttl>0)
                             {
                                        //update ttl hops dest_addr
                                        int new_dest_addr;
                                        new_dest_addr=dest_addr*10+input_rcv_num;
                                        op_pk_nfd_set_int32 (pkptr, "dest_addr",
new_dest_addr);
                                        op_pk_nfd_set_int32(pkptr,"TTL",ttl-1);
                                        op_pk_nfd_set_int32(pkptr,"Hops",hops+1);

                                        //copy packet 4 times
```

```
    pkptr_1=op_pk_create_fmt("gnutella_simple_pk_format");

                                                op_pk_nfd_set_pkid (pkptr_1, "packet_id",
o_pk_id);
                                                op_prg_odb_bkpt ("3");

                                                op_pk_nfd_set_int32 (pkptr_1, "Payload
Descriptor", Query);
                                                op_pk_nfd_set_int32 (pkptr_1, "TTL", ttl-1);
                                                op_pk_nfd_set_int32 (pkptr_1, "Hops", hops+1);
                                                op_pk_nfd_set_int32 (pkptr_1,"node_id",
node_id);
                                                op_pk_nfd_set_int32 (pkptr_1, "dest_addr",
new_dest_addr);
                                               op_pk_nfd_set_int32 (pkptr_1,"search",search);

                                                op_pk_nfd_set_pkid
(pkptr_1,"sender_node_id",sender_node_id);

                                                op_prg_odb_bkpt ("4");


    pkptr_2=op_pk_create_fmt("gnutella_simple_pk_format");
                                                op_pk_nfd_set_pkid (pkptr_2, "packet_id",
o_pk_id);
                                                op_pk_nfd_set_int32 (pkptr_2, "Payload
Descriptor", Query);
                                                op_pk_nfd_set_int32 (pkptr_2, "TTL", ttl-1);
                                                op_pk_nfd_set_int32 (pkptr_2, "Hops", hops+1);
                                                op_pk_nfd_set_int32 (pkptr_2,"node_id",
node_id);
                                                op_pk_nfd_set_int32 (pkptr_2, "dest_addr",
new_dest_addr);
                                                op_pk_nfd_set_int32
(pkptr_2,"search",search);
                                                op_pk_nfd_set_pkid
(pkptr_2,"sender_node_id",sender_node_id);


    pkptr_3=op_pk_create_fmt("gnutella_simple_pk_format");
                                                op_pk_nfd_set_pkid (pkptr_3, "packet_id",
o_pk_id);
                                                op_pk_nfd_set_int32 (pkptr_3, "Payload
Descriptor", Query);
```

```
                                        op_pk_nfd_set_int32 (pkptr_3, "TTL", ttl-1);
                                        op_pk_nfd_set_int32 (pkptr_3, "Hops", hops+1);
                                        op_pk_nfd_set_int32 (pkptr_3,"node_id",
node_id);
                                        op_pk_nfd_set_int32 (pkptr_3, "dest_addr",
new_dest_addr);
                                        op_pk_nfd_set_int32
(pkptr_3,"search",search);
                                        op_pk_nfd_set_pkid
(pkptr_3,"sender_node_id",sender_node_id);


    pkptr_4=op_pk_create_fmt("gnutella_simple_pk_format");
                                        op_pk_nfd_set_pkid (pkptr_4, "packet_id",
o_pk_id);
                                        op_pk_nfd_set_int32 (pkptr_4, "Payload
Descriptor", Query);
                                        op_pk_nfd_set_int32 (pkptr_4, "TTL", ttl-1);
                                        op_pk_nfd_set_int32 (pkptr_4, "Hops", hops+1);
                                        op_pk_nfd_set_int32 (pkptr_4,"node_id",
node_id);
                                        op_pk_nfd_set_int32 (pkptr_4, "dest_addr",
new_dest_addr);
                                        op_pk_nfd_set_int32
(pkptr_4,"search",search);
                                        op_pk_nfd_set_pkid
(pkptr_4,"sender_node_id",sender_node_id);
                            //fwd query
                                        if(input_rcv_num==rcv0_in)
                                            {op_pk_send(pkptr,xmt1_out);
                                            op_pk_send(pkptr_1,xmt2_out);
                                            op_pk_send(pkptr_2,xmt3_out);
                                            op_pk_send(pkptr_3,xmt4_out);
                                            op_pk_send(pkptr_4,xmt5_out);}
                                        else if(input_rcv_num==rcv1_in)
                                            {op_pk_send(pkptr,xmt0_out);
                                            op_pk_send(pkptr_1,xmt2_out);
                                            op_pk_send(pkptr_2,xmt3_out);
                                            op_pk_send(pkptr_3,xmt4_out);
                                            op_pk_send(pkptr_4,xmt5_out);}
                                        else if(input_rcv_num==rcv2_in)
                                            {op_pk_send(pkptr,xmt1_out);
                                            op_pk_send(pkptr_1,xmt0_out);
                                            op_pk_send(pkptr_2,xmt3_out);
```

```
                                    op_pk_send(pkptr_3,xmt4_out);
                                    op_pk_send(pkptr_4,xmt5_out);}
                                else if(input_rcv_num==rcv3_in)
                                    {op_pk_send(pkptr,xmt1_out);
                                    op_pk_send(pkptr_1,xmt2_out);
                                    op_pk_send(pkptr_2,xmt0_out);
                                    op_pk_send(pkptr_3,xmt4_out);
                                    op_pk_send(pkptr_4,xmt5_out);}
                                else if(input_rcv_num==rcv4_in)
                                    {op_pk_send(pkptr,xmt1_out);
                                    op_pk_send(pkptr_1,xmt2_out);
                                    op_pk_send(pkptr_2,xmt3_out);
                                    op_pk_send(pkptr_3,xmt0_out);
                                    op_pk_send(pkptr_4,xmt5_out);}
                                else if(input_rcv_num==rcv5_in)
                                    {op_pk_send(pkptr,xmt1_out);
                                    op_pk_send(pkptr_1,xmt2_out);
                                    op_pk_send(pkptr_2,xmt3_out);
                                    op_pk_send(pkptr_3,xmt4_out);
                                    op_pk_send(pkptr_4,xmt0_out);}
                                pk_total_out=pk_total_out+5;
                                pk_q_out=pk_q_out+5;
                                }
            if(ttl<=0)
                {
                    op_pk_destroy(pkptr);
                }



            }
        }
    else if (payload_des==8)//Query hit
        {
          pk_total_in++;
        //read all the information from packet
            op_pk_nfd_get_pkid(pkptr,"packet_id",&o_pk_id);
            op_pk_nfd_get_int32(pkptr,"TTL",&ttl);
            op_pk_nfd_get_int32(pkptr,"Hops",&hops);
            op_pk_nfd_get_int32(pkptr,"node_id",&node_id);
            op_pk_nfd_get_int32(pkptr,"dest_addr",&dest_addr);
            op_pk_nfd_get_pkid(pkptr,"sender_node_id",&sender_node_id);
            op_pk_nfd_get_int32(pkptr,"search",&search);

        //see whether its node_id matches node_objid. if match, it is this node's
```

queryhit. update statistics and destroy it. if not, decode dest_addr, update dest_addr field in pakcet and send it, update statistics

```
                    if (node_id==node_objid)
                    {
                        op_prg_odb_bkpt ("06");
                        op_pk_destroy(pkptr);
                        pk_qh_in ++;
                    }
                    else
                    {
                        //decode port
                        int decoded_send_port;
                        decoded_send_port=dest_addr%10;

                        //update dest_addr
                        op_pk_nfd_set_int32 (pkptr, "dest_addr",
(dest_addr-decoded_send_port)/10);

                        //continue send pong through decoded port
                        //printf("dest_addr: %d\n",dest_addr);
                        //printf("decoded_send_port: %d\n",decoded_send_port);
                        op_pk_send(pkptr,decoded_send_port-1);

                        //update statistics
                        pk_total_out++;

                    }




                }
            else
                {
                op_pk_destroy(pkptr);
                }
            //write statistics
            op_stat_write(pk_total_out_stat,pk_total_out);
            op_stat_write(pk_total_in_stat,pk_total_in);
            op_stat_write(pk_ping_out_stat,pk_ping_out);
            op_stat_write(pk_ping_in_stat,pk_ping_in);
            op_stat_write(pk_pong_out_stat,pk_pong_out);
            op_stat_write(pk_pong_in_stat,pk_pong_in);
            op_stat_write(pk_q_out_stat,pk_q_out);
```

```
                op_stat_write(pk_q_in_stat,pk_q_in);
                op_stat_write(pk_qh_out_stat,pk_qh_out);
                op_stat_write(pk_qh_in_stat,pk_qh_in);
                op_stat_write(pk_q_own_out_stat,pk_q_own_out);
                }
                FSM_PROFILE_SECTION_OUT (state2_enter_exec)

            /** state (proc) exit executives **/
            FSM_STATE_EXIT_FORCED (2, "proc", "liu_testgnutella_node_proc_comment_code [proc
exit execs]")


            /** state (proc) transition processing **/
           FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "proc", "idle", "tr_10",
"liu_testgnutella_node_proc_comment_code [proc -> idle : default / ]")
                /*---------------------------------------------------------*/



            }


        FSM_EXIT (0,"liu_testgnutella_node_proc_comment_code")
        }
    }




void
_op_liu_testgnutella_node_proc_comment_code_diag (OP_SIM_CONTEXT_ARG_OPT)
    {
    /* No Diagnostic Block */
    }




void
_op_liu_testgnutella_node_proc_comment_code_terminate (OP_SIM_CONTEXT_ARG_OPT)
    {

    FIN_MT (_op_liu_testgnutella_node_proc_comment_code_terminate ())
```

```
    /* No Termination Block */

    Vos_Poolmem_Dealloc (op_sv_ptr);

    FOUT
    }


/* Undefine shortcuts to state variables to avoid */
/* syntax error in direct access to fields of */
/* local variable prs_ptr in _op_liu_testgnutella_node_proc_comment_code_svar function. */
#undef pk_total_out
#undef pk_total_in
#undef pk_ping_out
#undef pk_ping_in
#undef pk_pong_out
#undef pk_pong_in
#undef pk_q_out
#undef pk_q_in
#undef pk_qh_out
#undef pk_qh_in
#undef ping_cache
#undef pong_cache
#undef ava_data
#undef ping_ptr
#undef pong_ptr
#undef pk_total_out_stat
#undef pk_total_in_stat
#undef pk_ping_out_stat
#undef pk_ping_in_stat
#undef pk_pong_out_stat
#undef pk_pong_in_stat
#undef pk_q_out_stat
#undef pk_q_in_stat
#undef pk_qh_out_stat
#undef pk_qh_in_stat
#undef node_objid
#undef src_packet_id
#undef src_ping_ptr
#undef data_pool
#undef pk_q_own_out
#undef pk_q_own_out_stat
#undef query_cache
```

```
#undef query_ptr

#undef FIN_PREAMBLE_DEC
#undef FIN_PREAMBLE_CODE

#define FIN_PREAMBLE_DEC
#define FIN_PREAMBLE_CODE

VosT_Obtype
_op_liu_testgnutella_node_proc_comment_code_init (int * init_block_ptr)
	{
	VosT_Obtype obtype = OPC_NIL;
	FIN_MT (_op_liu_testgnutella_node_proc_comment_code_init (init_block_ptr))

	obtype = Vos_Define_Object_Prstate ("proc state vars
(liu_testgnutella_node_proc_comment_code)",
		sizeof (liu_testgnutella_node_proc_comment_code_state));
	*init_block_ptr = 0;

	FRET (obtype)
	}

VosT_Address
_op_liu_testgnutella_node_proc_comment_code_alloc (VosT_Obtype obtype, int init_block)
	{
#if !defined (VOSD_NO_FIN)
	int _op_block_origin = 0;
#endif
	liu_testgnutella_node_proc_comment_code_state * ptr;
	FIN_MT (_op_liu_testgnutella_node_proc_comment_code_alloc (obtype))

	ptr = (liu_testgnutella_node_proc_comment_code_state *)Vos_Alloc_Object (obtype);
	if (ptr != OPC_NIL)
		{
		ptr->_op_current_block = init_block;
#if defined (OPD_ALLOW_ODB)
		ptr->_op_current_state = "liu_testgnutella_node_proc_comment_code [init enter
execs]";
#endif
		}
	FRET ((VosT_Address)ptr)
	}
```

```
void
_op_liu_testgnutella_node_proc_comment_code_svar (void * gen_ptr, const char * var_name, void
** var_p_ptr)
    {
    liu_testgnutella_node_proc_comment_code_state      *prs_ptr;

    FIN_MT (_op_liu_testgnutella_node_proc_comment_code_svar (gen_ptr, var_name, var_p_ptr))

    if (var_name == OPC_NIL)
        {
        *var_p_ptr = (void *)OPC_NIL;
        FOUT
        }
    prs_ptr = (liu_testgnutella_node_proc_comment_code_state *)gen_ptr;

    if (strcmp ("pk_total_out" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pk_total_out);
        FOUT
        }
    if (strcmp ("pk_total_in" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pk_total_in);
        FOUT
        }
    if (strcmp ("pk_ping_out" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pk_ping_out);
        FOUT
        }
    if (strcmp ("pk_ping_in" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pk_ping_in);
        FOUT
        }
    if (strcmp ("pk_pong_out" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pk_pong_out);
        FOUT
        }
    if (strcmp ("pk_pong_in" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->pk_pong_in);
```

```
    FOUT
    }
if (strcmp ("pk_q_out" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_q_out);
    FOUT
    }
if (strcmp ("pk_q_in" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_q_in);
    FOUT
    }
if (strcmp ("pk_qh_out" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_qh_out);
    FOUT
    }
if (strcmp ("pk_qh_in" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_qh_in);
    FOUT
    }
if (strcmp ("ping_cache" , var_name) == 0)
    {
    *var_p_ptr = (void *) (prs_ptr->ping_cache);
    FOUT
    }
if (strcmp ("pong_cache" , var_name) == 0)
    {
    *var_p_ptr = (void *) (prs_ptr->pong_cache);
    FOUT
    }
if (strcmp ("ava_data" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->ava_data);
    FOUT
    }
if (strcmp ("ping_ptr" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->ping_ptr);
    FOUT
    }
if (strcmp ("pong_ptr" , var_name) == 0)
    {
```

```
    *var_p_ptr = (void *) (&prs_ptr->pong_ptr);
    FOUT
    }
if (strcmp ("pk_total_out_stat" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_total_out_stat);
    FOUT
    }
if (strcmp ("pk_total_in_stat" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_total_in_stat);
    FOUT
    }
if (strcmp ("pk_ping_out_stat" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_ping_out_stat);
    FOUT
    }
if (strcmp ("pk_ping_in_stat" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_ping_in_stat);
    FOUT
    }
if (strcmp ("pk_pong_out_stat" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_pong_out_stat);
    FOUT
    }
if (strcmp ("pk_pong_in_stat" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_pong_in_stat);
    FOUT
    }
if (strcmp ("pk_q_out_stat" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_q_out_stat);
    FOUT
    }
if (strcmp ("pk_q_in_stat" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_q_in_stat);
    FOUT
    }
if (strcmp ("pk_qh_out_stat" , var_name) == 0)
```

```
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_qh_out_stat);
    FOUT
    }
if (strcmp ("pk_qh_in_stat" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_qh_in_stat);
    FOUT
    }
if (strcmp ("node_objid" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->node_objid);
    FOUT
    }
if (strcmp ("src_packet_id" , var_name) == 0)
    {
    *var_p_ptr = (void *) (prs_ptr->src_packet_id);
    FOUT
    }
if (strcmp ("src_ping_ptr" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->src_ping_ptr);
    FOUT
    }
if (strcmp ("data_pool" , var_name) == 0)
    {
    *var_p_ptr = (void *) (prs_ptr->data_pool);
    FOUT
    }
if (strcmp ("pk_q_own_out" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_q_own_out);
    FOUT
    }
if (strcmp ("pk_q_own_out_stat" , var_name) == 0)
    {
    *var_p_ptr = (void *) (&prs_ptr->pk_q_own_out_stat);
    FOUT
    }
if (strcmp ("query_cache" , var_name) == 0)
    {
    *var_p_ptr = (void *) (prs_ptr->query_cache);
    FOUT
    }
```

```
    if (strcmp ("query_ptr" , var_name) == 0)
        {
        *var_p_ptr = (void *) (&prs_ptr->query_ptr);
        FOUT
        }
    *var_p_ptr = (void *)OPC_NIL;
FOUT }
```