SIMON FRASER UNIVERSITY

Analysis of BitTorrent Protocol and Its Effect on the Network

ENSC 427: Final Project Report

Spring 2011

Group 11

www.sfu.ca/~kna5/ensc427

Ken Kyoungwoo Nam
301046747
Kna5 @sfu.ca

Yu Jie Xu
301083552
Xya14 @ sfu.ca

Abstract

The first version of the peer-to-peer file sharing protocol was invented in 1999, called Napster protocol. From then on, the application of peer-to-peer protocol has been widely spread in the internet. The advantage of the network with p2p protocol is that it needs much less server bandwidth compare to the basic client and server network. Moreover, in the p2p network, the client itself is the server, so they can communicate with each other without the central sever. Nowadays, there are two primary peer-to-peer file sharing protocol that dominate in the network: the Gnutella protocol and BitTorrent Protocol. In our project, we will focus on BitTorrent Protocol. To do this, we will create three different networks in OPNET, and investigate the network performance with and without BitTorrent nodes.

Table of contents

1. Introduction

Nowadays, Peer-to-peer (P2P) networking has become one of most popular concepts in the computing science field. P2P networking is a distributed application architecture that partitions tasks or workloads between peers. Networks such as BitTorrent make it easy for people to find what they want and share what they have.

BitTorrent is one of the most popular file sharing protocols that makes use of torrent files to find and share resources. It identifies content by URL and is designed to integrate seamlessly with the web. BitTorrent has an advantage that over plain HTTP which is when multiple downloads of the same file happen at the same time, the downloaders upload to each other, making it possible for the file source to support very large numbers of downloaders with only a modest increase in its load. Moreover, a user who wants to upload a file into the network called seed, and who downloads a file from a network called peers.

On the other hand, P2P protocols like BitTorrent today consumes very large portion of the network traffic which results network congestion. To reduce the congestion caused by the P2P traffic, some ISPs are throttling P2P traffic and University networks are blocking P2P traffic to provide better quality of service to non-P2P users.

The main goal of this project is investigating the BitTorrent protocol and how P2P file sharing congests different types of network. By doing this, we will create three different networks: normal client server network, network with p2p nodes and network with BitTorrent nodes, in OPNET and analyze each network's performance.

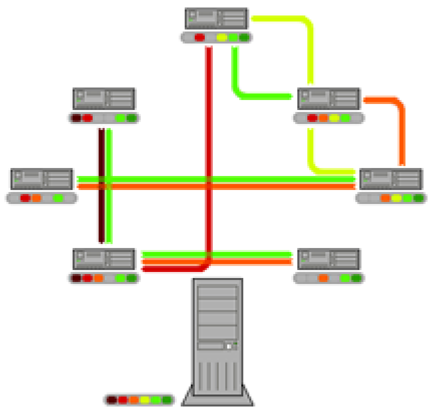


FIGURE1.1 BitTorrent demonstration

2. Theory

2.1 Terminology and Definition

The terminology used in the peer-to-peer application field is not standardized. For easy to explain the rarest first algorithm and choke algorithm in our paper, we defined the following terms, and also we gave the explanation of the important terms in our project as well.

Peer Set
In the peer set, there is a list of peers participating in the network.

Piece and Block
In the BitTorrent application , data is split into smaller pieces which sent between peers, and then the each piece can split into Blocks, which is transmission unit in the network, but BitTorrent protocol only accept transferred pieces.

Metainfo File
A metainfo file is a file which contains all the necessary details of the protocol to operate.

Torrent
A torrent is a type of meta-data that facilitates peer-to-peer file sharing.

Current Swarm
The piece held by the lowest number of peers.

Rare Pieces
We defined the rare piece that only existing on the original seed.

Rarest Pieces and Rarest Pieces set
We called the rarest pieces are the pieces that have the least number of flies copies in the peer set. Then the Rarest Pieces can form the rarest piece set.

2.2 Peer-to-Peer Protocol

In 1999, the first version of file sharing protocol called Napster protocol appeared. Since then, more and more people use the P2P protocol to request, prepare and transmit various resources over a network, so peer-to-peer file sharing has become the most popular application in the Internet. According to the network in the way sharing of the resources, the local area network has two organization forms: one is the peer-to-peer network, and another is the basic client server structure. However, in client server structure, all the resource requests from different workstations and then go to a central server. The server then receives each request and then sends back the resource which each client requested. Each client doesn’t mutually shared resource directly. As more and more clients request the resource from the central server, the more load will on the server, thus speed of downloading resource will decrease. In order to solve this problem, the peer-to-peer network was introduced. In P2P network, client can share the resources from each other, which mean each client can be the server. Thus, the capacity of service in the network will not overload, and also as more clients participating in the network, the more probability of successful connection will achieved.

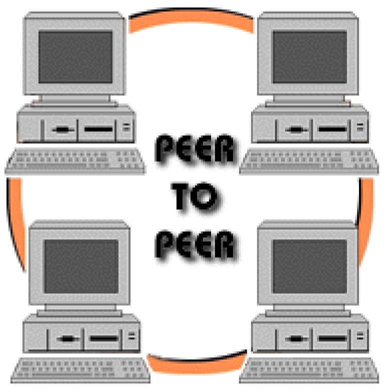


FIGURE 2.1 Peer-to-peer network

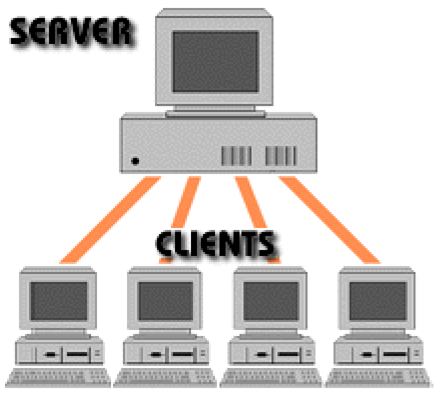


FIGURE 2.2 Server/client network

2.3 BitTorrent Protocol

The BitTorrent protocol was invented by Bram Cohen in 2001. In BitTorrent protocol, every client is capable of preparing, requesting, and transmitting any type of data over a network, using the protocol. As the BitTorrent application has been widely used, so the terminology called seed and peers appeared. The client that providing files are called seed and the client who are downloading the files are called peers.

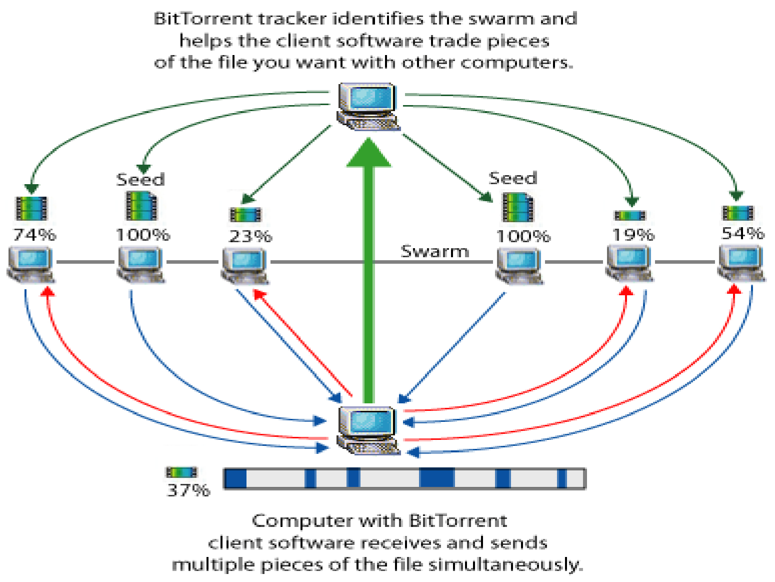

BitTorrent tracker identifies the swarm and helps the client software trade pieces of the file you want with other computers.
Seed
Seed
74%
100%
23%
100%
19%
54%
Swarm
37%
Computer with BitTorrent client software receives and sends multiple pieces of the file simultaneously.

FIGURE 2.3 Seeds and peer diagram

In order to deploy BitTorrent application, the seed should places an .torrent file onto an ordinary web server. In the .torrent file, contained the file information to share, including the filename, the size, the document disperses row information and aims at tracker URL. Tracker is responsible to help peers to be able to gain other seed information. Between tracker and peer, they use a very simple way to communicate with each other, which is based on HTTP agreement alternately. The peers tell tracker the type of file they want to download, the port type they are using and as well as similar information. Thus, tracker will make a list of clients that participating in the network in order to give related information to peers, so the peers this information to establish the connection between seeds. Moreover, after a peer downloaded the file, it can provide the file still available to become additional seeds.

2.4 BitTorrent Tracker

Track is a server that assists in the communication between peers and seeds using the BitTorrent protocol. Track can trace out a list of clients participating in the network. In addition, peers know nothing of each other until a response is received from the tracker. Thus, if the peers connect to the tracker server, so they can obtain the related information about the file that they want to download. Moreover, the role of the tracker ends once peers have known each other. From then on, communication is done directly between peers, and the tracker is not involved in the network anymore.

The tracker server plays an important role on BitTorrent application. A BitTorent client must communicate with the tracker before starting downloading the file as well as the downloading in progress to report their own downloading information and also can gain the new seed information. This kind of correspondence is carries on through the HTTP agreement, and also is called tracker the HTTP agreement.

Moreover, the role of the tracker ends once peers have known each other. From then on, communication is done directly between peers, and the tracker is not involved.

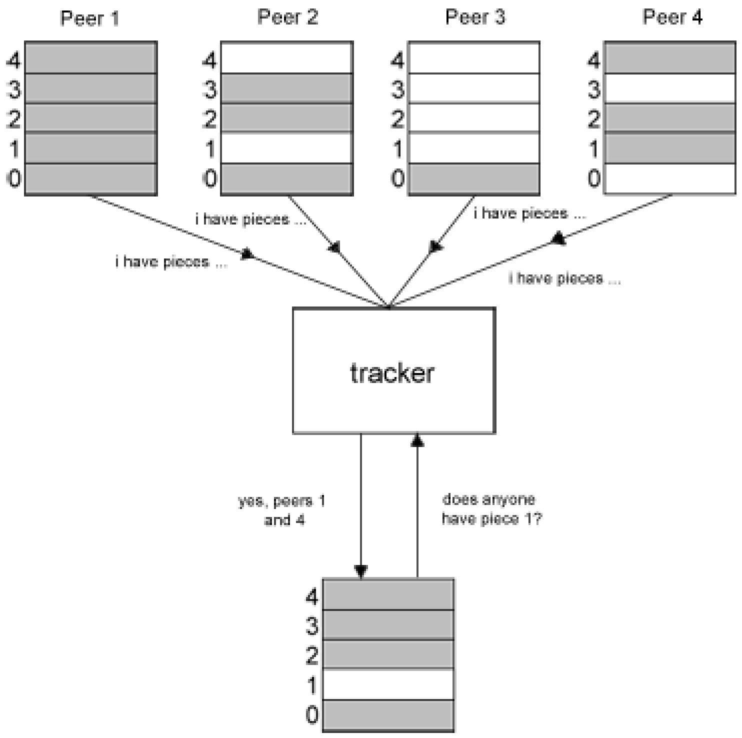


FIGURE 2.4 A tracker providing a list of peers with the require files

2.5 Rarest First Algorithm

The rarest first algorithm is the piece selection strategy used in BitTorrent. It works as follows. As a peer selects which piece to download next, the rarest piece will be chosen from the current swarm. This means that the most common pieces are left, and focus goes to replication of rarer pieces. At the beginning of starting a torrent, it will be only one seed which have the complete file so that it will cause a problem if multiple downloaders were trying to access the same piece. However, the rarest first algorithm avoids this problem because different peers keep different pieces. When more and more peers connect to the network, rarest first will not need tracker any more, as peers begin to download from each other. Finally, the original seed will disappear from a torrent so that it will cause a potential of losing pieces if no current peers have them. Rarest first algorithm works to prevent this loss of pieces by replicating the pieces most likely to be lost.

Moreover, the behavior of the rarest first algorithm can be modified by three additional policies: random first policy, strict priority policy and end game mode.

Firstly, if a peer has downloaded less than 4 pieces, it will choose the next piece to be requested randomly. This is called the random first policy. After the peer has downloaded at least 4 pieces, it switches to the rarest first algorithm. The functionality of the random first policy is to permit a peer to download its first pieces faster than with the rarest first policy, as it is important to have some pieces to reciprocate for the choke algorithm. In fact, a piece chosen at random is likely to be more replicated than the rarest pieces so that its downloading time will be shorter.

Secondly, a strict priority policy will be applied at the block level of BitTorrent. It works as follows. When at least one block of a piece has been requested, the other blocks of the same piece are requested with the highest priority. The functionality of the strict priority policy is to complete the download of a piece as fast as possible. As only complete pieces can be sent, it is important to minimize the number of pieces that has been received.

Finally, the last policy is the end game mode. This policy works as follows. It starts as a peer has requested all blocks, for example, all blocks have either been already received or requested. In addition, in this mode, the peer requests all blocks not yet received to all the peers in its peer set that have the corresponding blocks. Each time a block is received, it cancels the request for the received block to all the peers in its peer set.

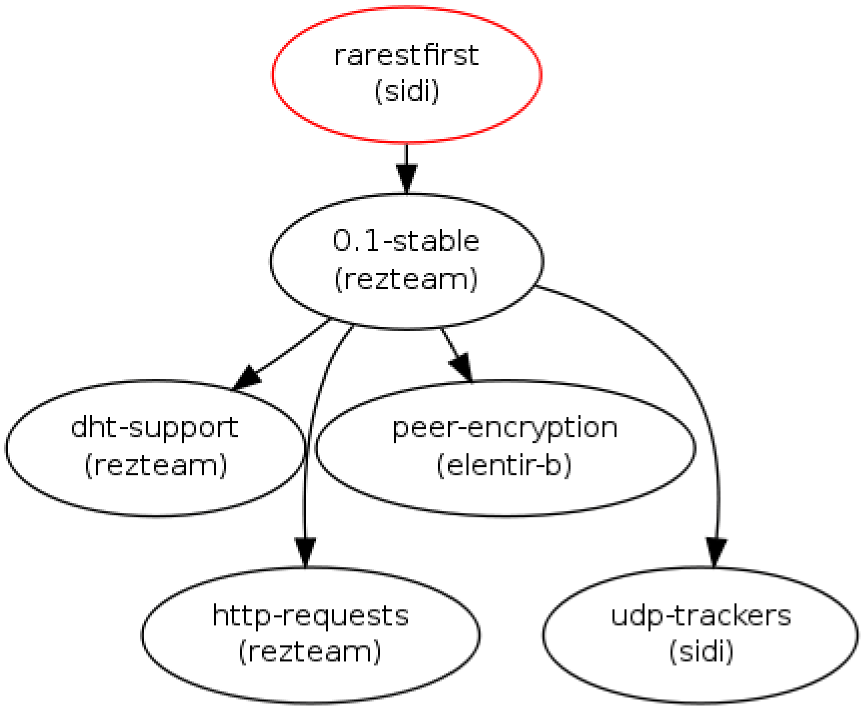


FIGURE 2.5 The flow chart of the rarest first algorithm

2.6 Chocking Algorithm

The rarest first algorithm is the piece selection strategy used in BitTorrent. It works as follows. Peers will continue to download files from all available peers that have required file pieces, and peers can block others from downloading data if necessary. This is called as choking algorithm.

When a peer receives a request for a piece from another peer, it can choose to refuse to transmit that piece. If it happens, the peer is said to be choked and this can be caused at some reasons, but the most common is by default, a client will only maintain a default number of maximum uploads so that all further requests to the client will be marked as choked. Moreover, the default for maximum uploads is 4.

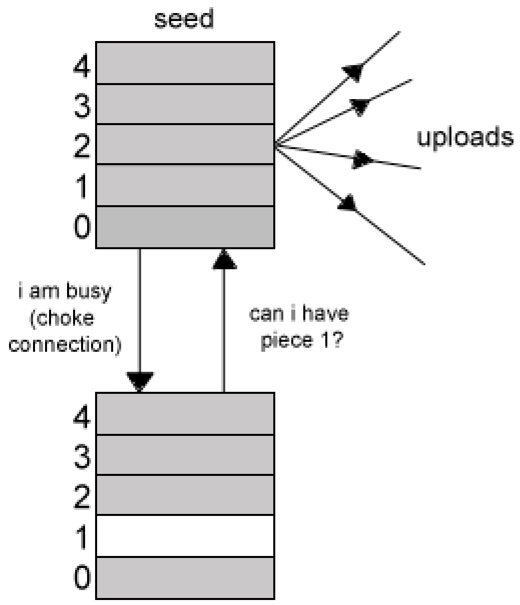


FIGURE 2.6 Choke algorithm demonstration

Showing in the above figure that a seed chokes the connection to a peer because it has reached its maximum uploads and the peer will keep choked until an unchoke message has been sent.

Another example that related to Chocking Algorithm is that optimistic unchoking, which ensure fairness between peers. It works as follows. When a peer is choked it is downloading from a seed, and the seed requires no pieces.

3. Implementation Details

To simulate the effect of BitTorrent traffic on the network, we used OPNET version 14 to implement different node types and network topologies. We first built a normal client and a server node models, and we built a plain P2P node model and a BitTorrent node model. Using the node models, we built two different networks: a small network and a large network. Finally, we created three scenarios for each network.

3.1 Packet Formats

There are two packet types transmitted and received from the nodes in our network models: normal packet and P2P packet.

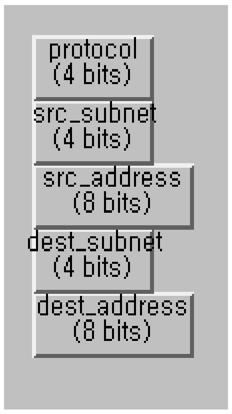


FIGURE 3.1 Normal Packet Format

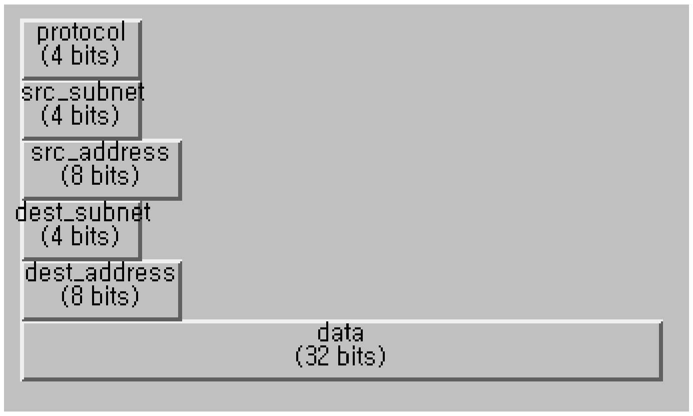


FIGURE 3.2 Peer-to-Peer Packet Format

Figure 3.1 is the packet format used for normal client and server nodes. It is consisted of a protocol value, source and destination subnets, and source and destination addresses. Figure 3.2 is the packet format used for both the plain P2P nodes and the BitTorrent nodes. An additional field used for storing data is added from the normal packet format. The protocol value is used to indicate the packet type. If the protocol value is set to zero, the packet type is request. The request packet is used by P2P nodes to request a piece of data. If the value is set to one, the packet type is reply. This packet type is also used by P2P nodes to reply to request transmitted by other P2P nodes. If the value is set to two, the packet type is normal. This packet type is used to model the traffic between client and server nodes.

3.2 Normal Client and Server Node Models

Normal client and server node models are used to simulate regular services on the network such as ftp and web services.

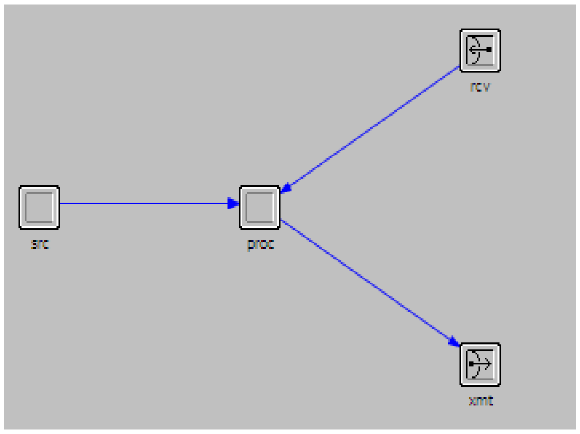


FIGURE 3.3 Client and Server Node Model

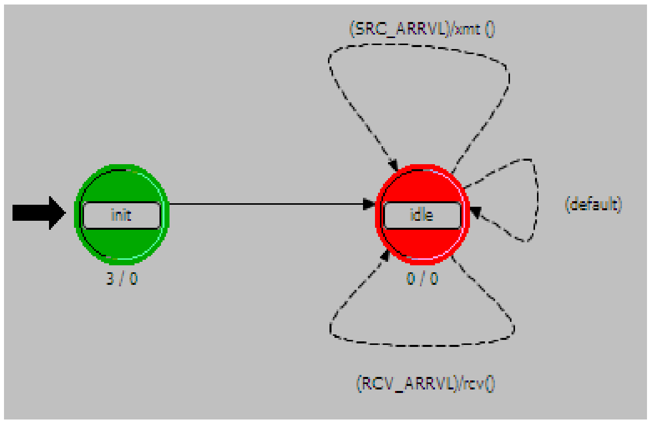


FIGURE 3.4 Client and Server Process Model

For the normal client and server nodes, we started building from the peripheral node in the OPNET tutorial, packet switching I. Figure 3.3 is the node model for both client and server, and figure 3.4 is the process model for both client and server. To accommodate the client and server behaviour, the client node is modified to send the packets to the server node only. Packet interarrival time for client is set to ten for the client and one for the server. Traffic generated from the server is larger than the client which is usually true in real world. Also, end-to-end delay is calculated in the normal client and server nodes only since we are interested in how these nodes are affected by the P2P traffic.

3.3 Plain Peer-to-Peer Node Model

Plain P2P node is a P2P node without any algorithms built in. The purpose of building this node is to see the effectiveness of the algorithms implemented in the BitTorrent node. From normal client node, two additional P2P source generators are added: request and reply.

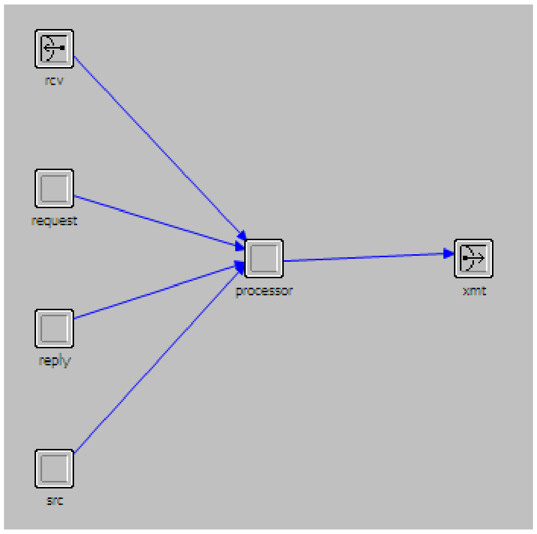


FIGURE 3.5 P2P Node Model

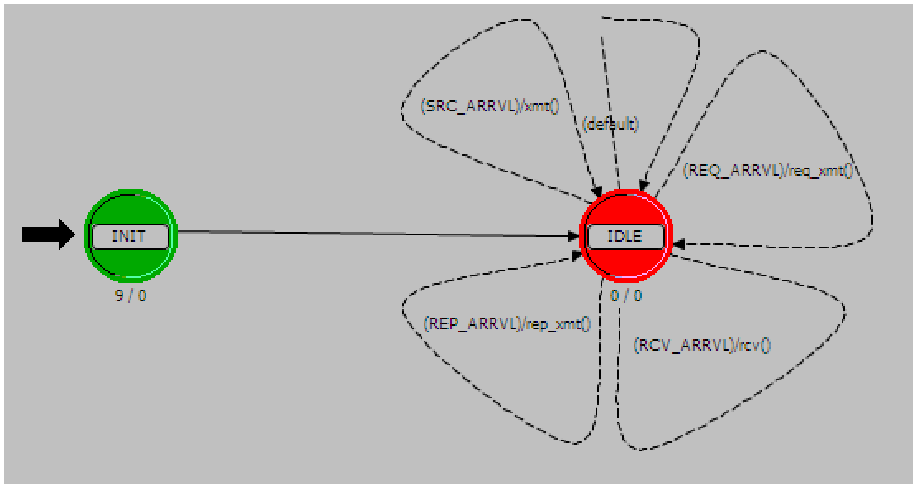


FIGURE 3.6 P2P Process Model

Figure 3.5 is the node model for the P2P node and figure 3.6 is the process model for the p2P node. The P2P node transmits request and reply to random addresses.

3.4 BitTorrent Node Model

For the BitTorrent node, we implemented a simplified version of tracker and choke algorithm. The node model for BitTorrent is exactly same as the P2P node model and the process model for BitTorrent has same states and transitions as the P2P node model. However, additional code is added to the functions to enable tracker and choke algorithm. With a presence of the tracker, BitTorrent peers know the address of the other peers. Therefore, a BitTorrent node now sends request and reply to only the BitTorrent nodes. Also, we enabled one key feature of the choke algorithm: limiting the number of connections each node. A BitTorrent node now only replies to the request sent from one fixed node.

3.5 Building the Small Network

The small network is used to simulate and also used for testing before we create more complicated and larger network. The small network uses router, server, and clients to set up a start topology. To examine the effects of the P2P and BitTorrent users on the network, we created three scenarios. Details on the router used in the network are available in the appendix at the end of the report.

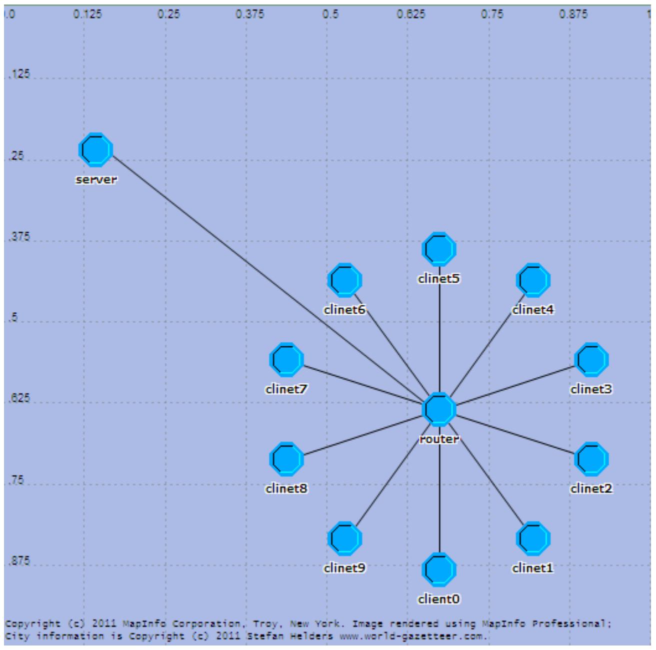

server
router
clinet5
clinet6
clinet4
clinet7
clinet3
clinet8
clinet2
clinet9
client0
clinet1
Copyright (c) 2011 MapInfo Corporation, Troy, New York. Image rendered using MapInfo Professional;
City information is Copyright (c) 2011 Stefan Helders www.world-gazetteer.com.

FIGURE 3.7 Small Network without P2P

Figure 3.7 is the small network with only normal server and client nodes. There are ten client nodes sending traffic to a server node. This scenario is used to check then performance without any P2P users.

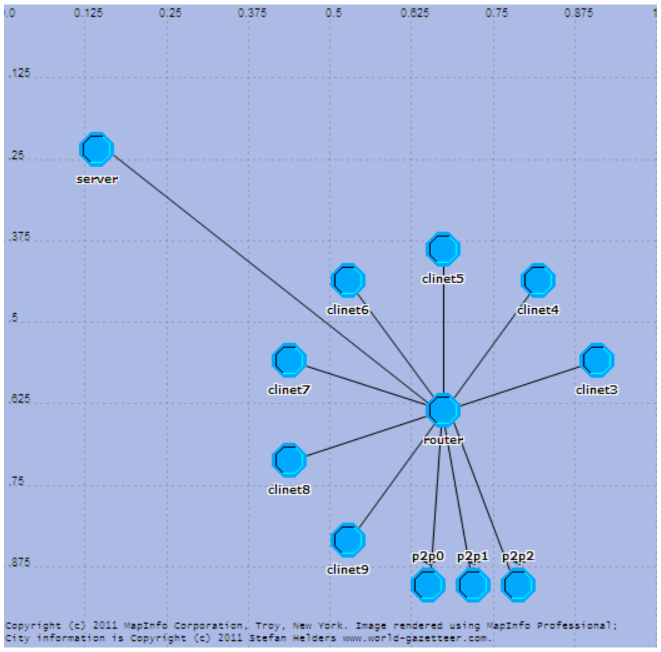

server
clinet5
clinet6
clinet4
clinet7
clinet3
router
clinet8
clinet9
p2p0
p2p1
p2p2
Copyright (c) 2011 MapInfo Corporation, Troy, New York. Image rendered using MapInfo Professional;
City information is Copyright (c) 2011 Stefan Helders www.world-gazetteer.com.

FIGURE 3.8 Small Network with P2P

Figure 3.8 is the small network with P2P nodes introduced. There are seven normal client nodes and three P2P nodes. This scenario is used to see how performance changes with introduction of plain P2P nodes.

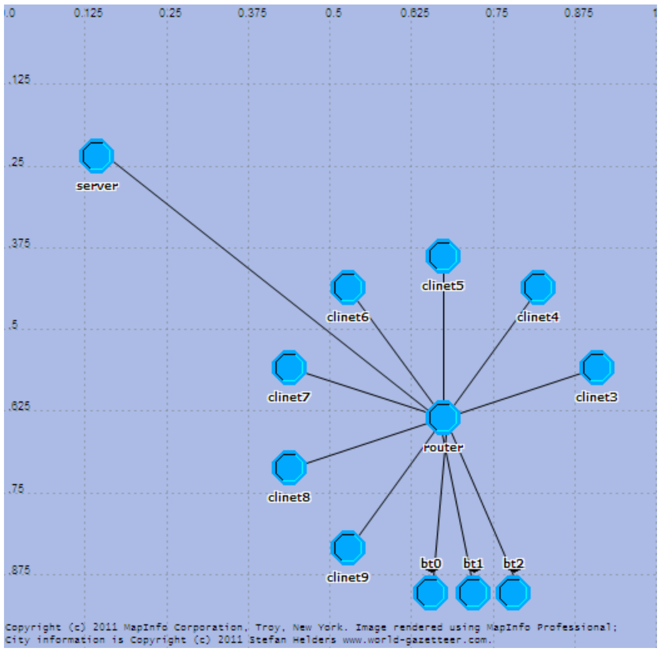
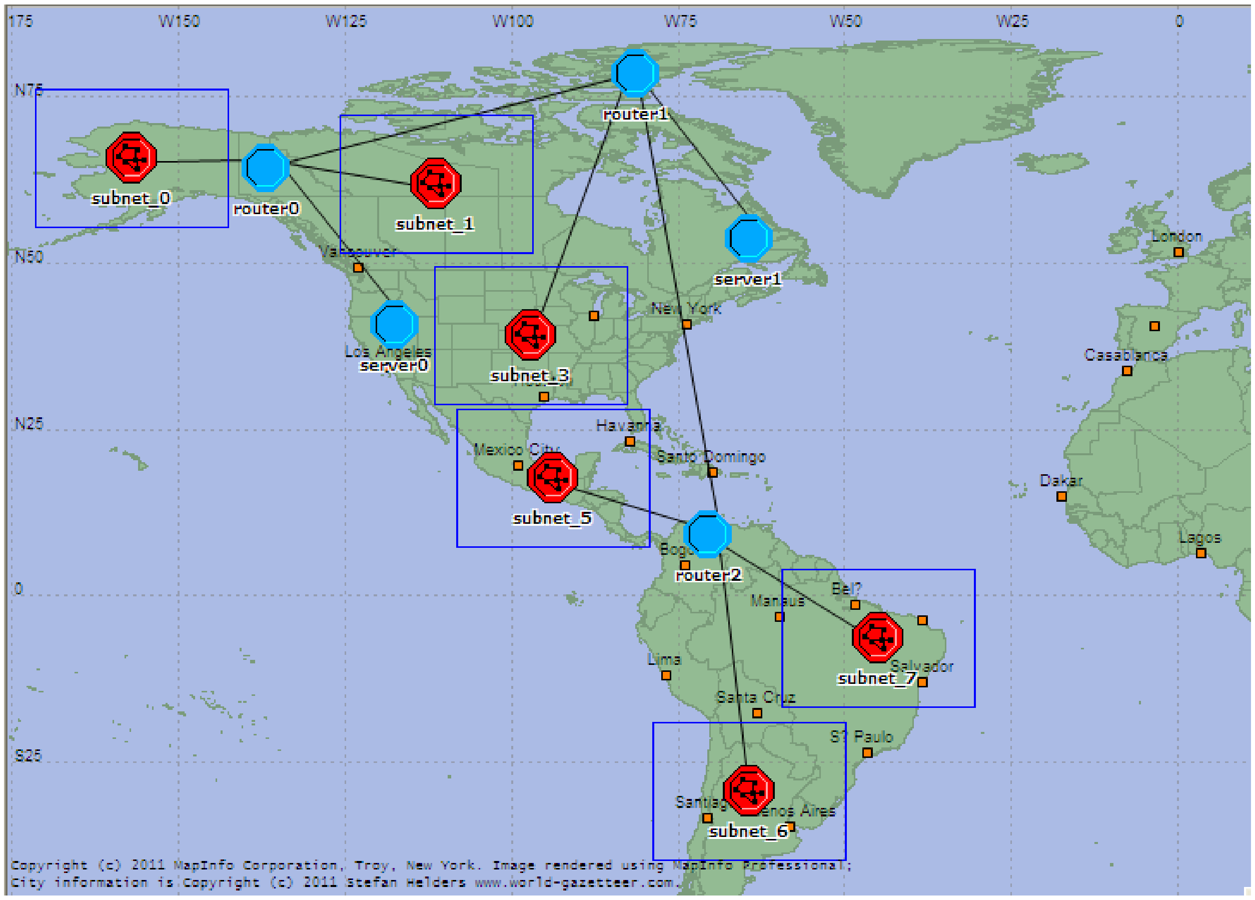

server
clinet5
clinet6
clinet4
clinet7
clinet3
router
clinet8
clinet9
bt0
bt1
bt2
Copyright (c) 2011 MapInfo Corporation, Troy, New York. Image rendered using MapInfo Professional;
City information is Copyright (c) 2011 Stefan Helders www.world-gazetteer.com.

FIGURE 3.9 Small Network with BitTorrent

Figure 3.9 is the small network with BitTorrent nodes introduced. There are seven normal client nodes and three BitTorrent nodes. This scenario is used to examine the effects of the BitTorrent nodes on the network.

3.6 Building the Large Network

After we successively built and tested the small network, we built a more realistic network containing larger number of nodes. Details on then routers used in the network are available in the appendix at the end of the report.



router1
router0
subnet_0
subnet_1
server1
server0
subnet_3
subnet_5
router2
subnet_7
subnet_6
Copyright (c) 2011 MapInfo Corporation, Troy, New York. Image rendered using MapInfo Professional;
City information is Copyright (c) 2011 Stefan Helders www.world-gazetteer.com.

FIGURE 3.10 Large network

Figure 3.10 is the large network we created. It has five subnets, two servers, and three routers. Inside each subnet is identical to the small network we built in the previous section. The large network also has three different scenarios: without P2P, with P2P, and with BitTorrent.

4. Result Analysis

To examine the effect of the presence of the BitTorrent nodes, we looked at the average end-to-end delay of server and clients, and throughput and utilization of client nodes for all three scenarios. We first analyzed the results from the small network, and then we analyzed at the large network.

4.1 Small Network

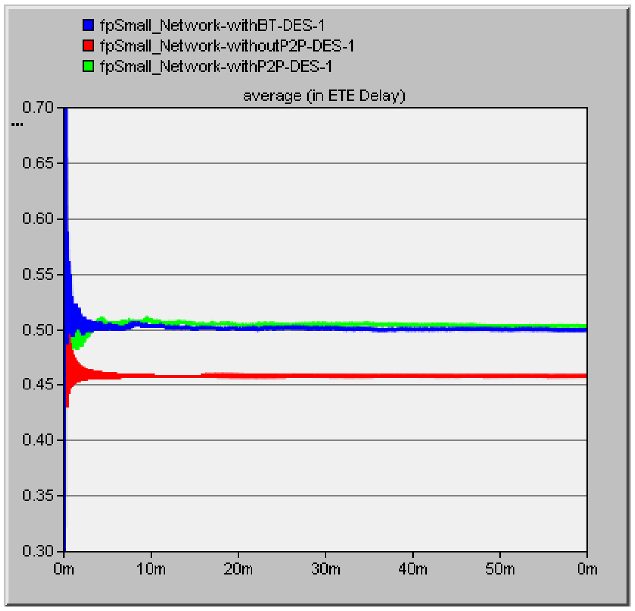


FIGURE 3.11 Average end-to-end delay between server and clients

Figure 3.11 displays the end-to-end delay graph for all three scenarios. The network without any P2P node has the smallest end-to-end delay and the network with plain P2P nodes has the largest end-to-end delay. The end-to-end delay for the scenario with the BitTorrent nodes is little bit smaller than the one with the plain P2P. This is because the BitTorrent nodes limit the P2P connection which also limits the transmitted traffic. However, the delay is still larger compare to the scenario without P2P.

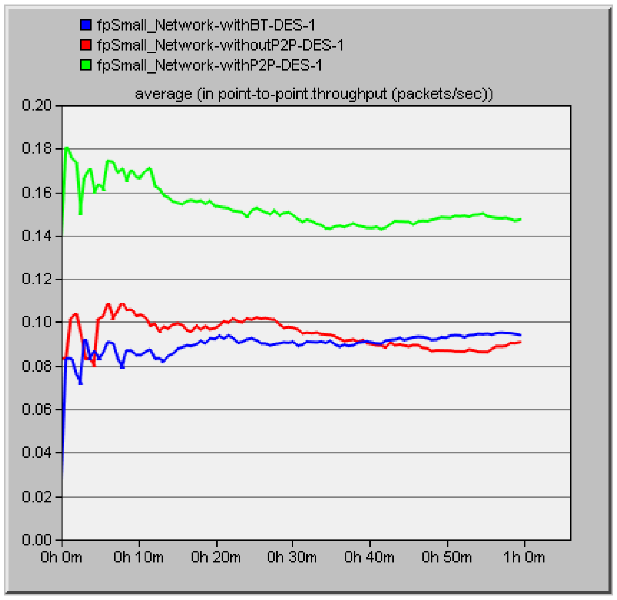


FIGURE 3.12 Throughput from the router to a client

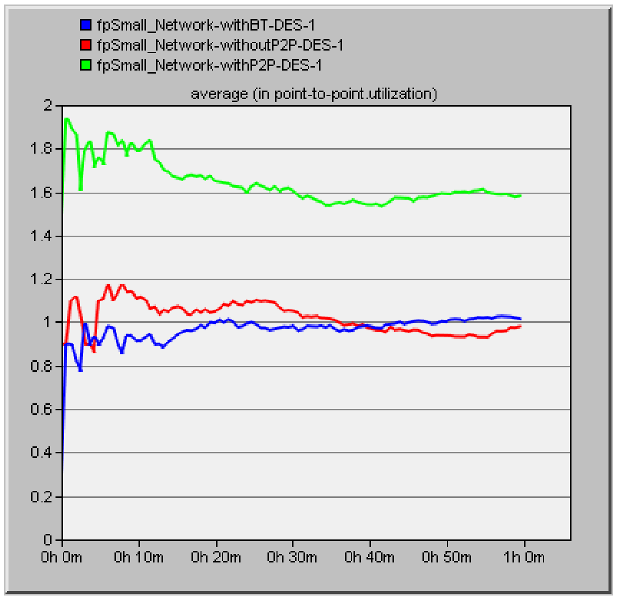


FIGURE 3.13 Utilization from the router to a client

Figure 3.12 and 3.13 shows the point-to-point throughput and utilization from the router to a client. The scenarios without P2P nodes and with BitTorrent nodes have similar results but the scenario with plain P2P nodes has much larger value. This is because the plain P2P nodes use flooding method to find other peers and therefore, transmit the P2P packets everywhere. Nodes that are not peers also receive this unwanted traffic. On the other hand, BitTorrent peers know the addresses of other BitTorrent peers and therefore, transmit the P2P packets to only BitTorrent peers. In this case, nodes that are not peers do not receive unwanted traffic.

4.2 Large Network

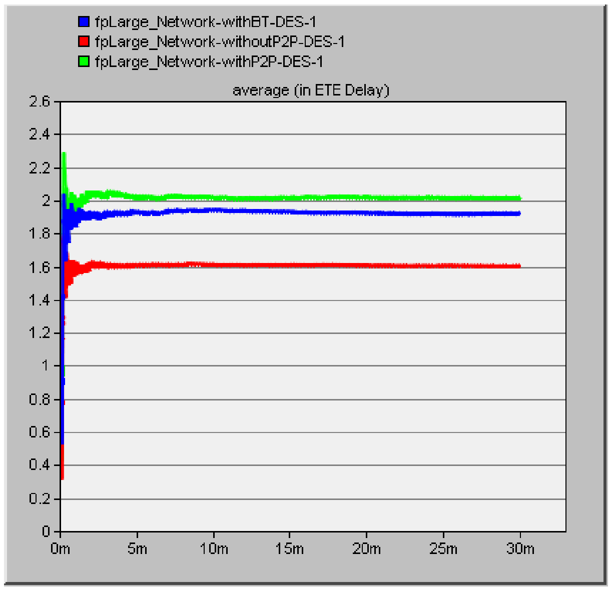


FIGURE 3.14 Average end-to-end delay between server and clients

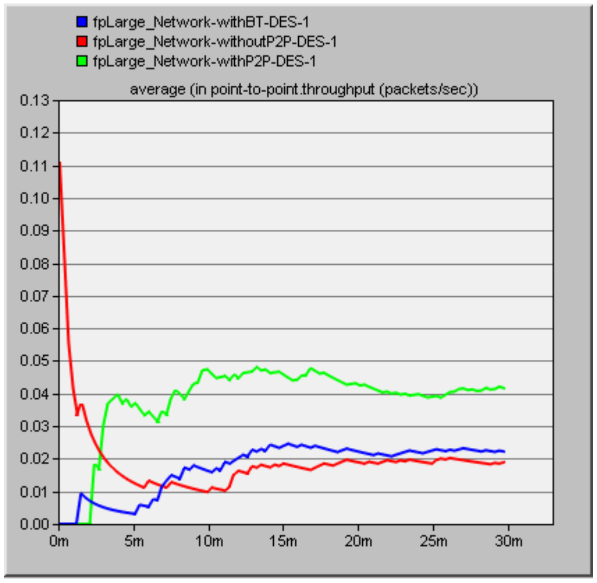


FIGURE 3.15 Throughput from the router to a client

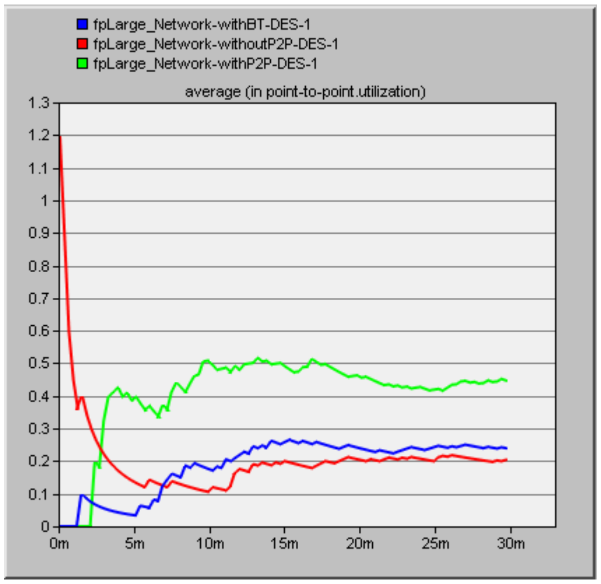


FIGURE 3.16 Utilization from the router to a client

Figure 3.14, 3.15, and 3.16 are the resulting graphs from the large network. The behaviour of the network is exactly same as the small network. Introduction of the plain P2P nodes and BitTorrent nodes slowed down the end-to-end delay between the servers and the clients. The BitTorrent nodes slow down the end-to-end delay less compare to the small network. This is because the BitTorrent nodes still reply to only one other peer with increased number of BitTorrent nodes. Throughput and utilization of a client in a subnet increased with the presence of the P2P nodes but stayed almost same with the presence of the BitTorrent nodes.

5. Discussion and Conclusion

In this project, our goal was to investigate how the introduction of the BitTorrent nodes affects different networks. As we expected, the OPNET simulation result indicates that the P2P traffics on networks increases the end-to-end delay. Also, we found out that with tracker and choke algorithm, the congestion on the network can be reduced. However, the performance on the network is still degraded significantly.

P2P data communication is very common these days. They are not only used to transfer files such as movies or applications but also used in messenger applications or online games. The problem is that non-P2P users have to suffer the congestion on the network because of the P2P users. To reduce the congestion caused by the P2P traffic, various solutions are introduced by researchers and internet service providers such as traffic throttling. However, currently available solutions like P2P traffic throttling harm P2P users. To provide better quality of service to all types of users on the network, research and development on the P2P protocols must be continued.

6. Difficulties

There were several difficulties we faced while doing the projects but the most challenging part was getting familiar with the tool, OPNET. Because it was first time doing a project on communications and network using OPNET, we needed to learn how to use it. Although we have done some tutorials, the tutorials only provided basic functionalities. We first had more complicated and realistic models for the P2P node and the BitTorrent node but we had to discard the models because they did not behave correctly. The details on the discarded BitTorrent node model are available in the appendix at the end of the report. The model was compiled without any error or warning messages but the simulation result indicated that the functionalities in the node models were not working properly.

7. Acknowledgment

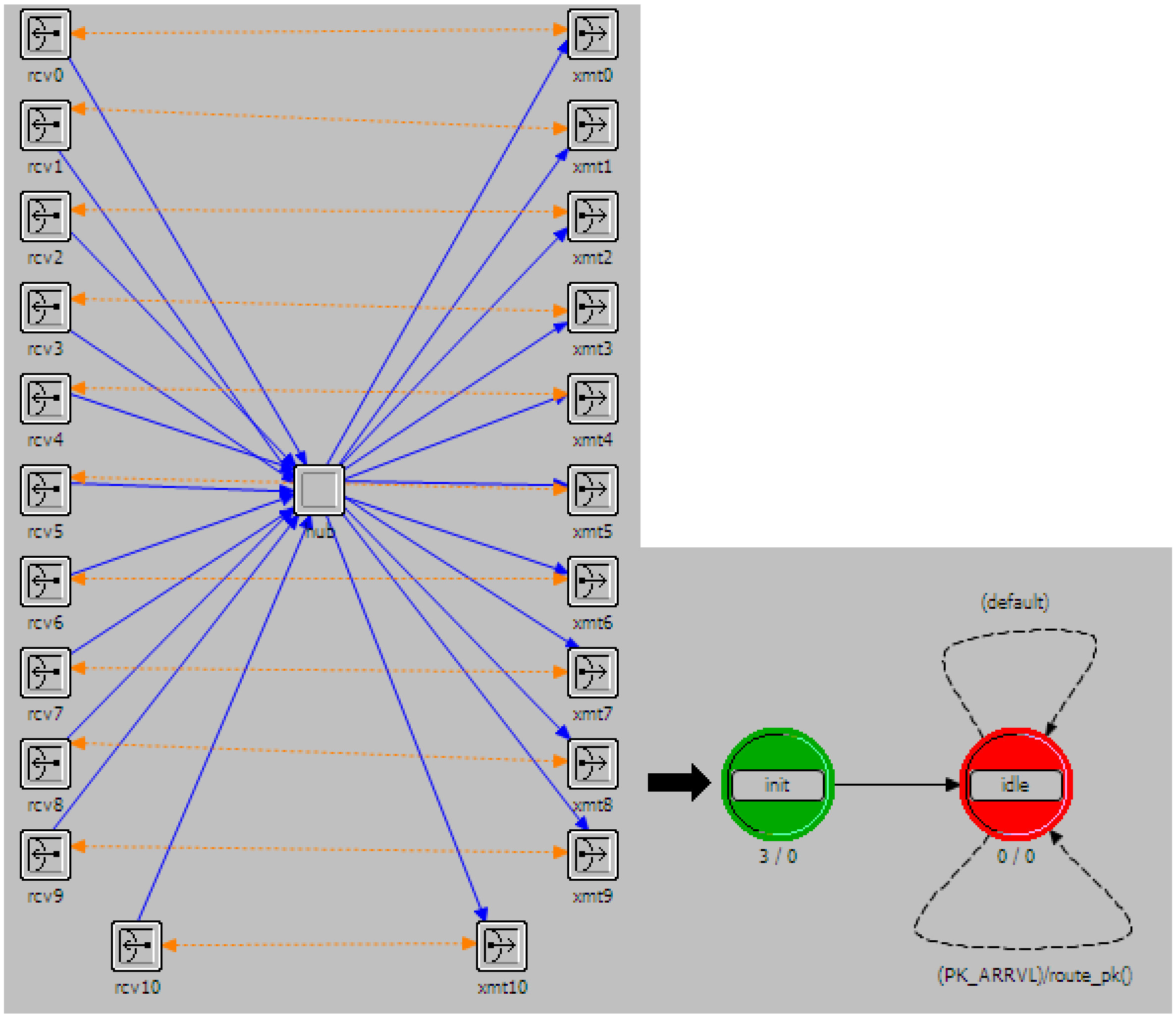
We would like to thank the professor Ljiljana Trajkovic, who gave us the valuable comments on our project.

8. References

[1] B. Cohen. The BitTorrent Protocol Specification. [Online].
Available: http://bittorrent.org/beps/bep_0003.html
[2] M. Fras, S. Klampfer and Ž. Čučej. "Impact of P2P traffic to the IP communication network performances," Systems, Signals and Image Processing, 2008. IWSSIP 2008. 15th International Conference. pp.205
[3] H. Lu and C. Wu. (2010). "Identification of P2P Traffic in Campus Network," in Computer Application and System Modeling (ICCASM), 2010 International Conference. pp.V1-21
[4] Y. Liu, H. Wang, Y. Lin and S. Cheng. (2008). "Friendly P2P: Application-level Congestion Control for Peer-to-Peer Applications," in Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE. pp.1
[5] N. Anderson. (2008). Canadian regulators allow P2P throttling. [Online]. Available: http://arstechnica.com/old/content/2008/11/canadian-regulators-allow-p2p-throttling.ars
[6] G. Urvoy-Keller and P. Michiardi. Rarest First and Choke Algorithms Are Enough. [Online]. Available: http://conferences.sigcomm.org/imc/2006/papers/p20-legout.pdf
[7] W. Ngiwlay, C. Intanagonwiwat, and Y. Teng-amnuay. BitTorrent Peer Identification based on Behaviors of a Choke Algorithm. [Online]. Available: http://delivery.acm.org/10.1145/1510000/1503392/p65-ngiwlay.pdf?key1=1503392&key2=1834303031&coll=DL&dl=ACM&ip=142.58.198.199&CFID=18345361&CFTOKEN=20623266
[8] Image. “Tracker.” [Online]. Available: http://images.morehawes.co.uk/bittorrent/tracker.png
[9] Image. “BitTorrent Protocol.” [Online]. Available: http://www.hawasly.biz/wp-content/uploads/2009/08/bitTorrentProtocol.gif
[10] Image. “Peer to peer.” [Online]. Available: http://members.tripod.com/barhoush_2/images/peer.gif
[11] Image. “BitTorrent Schema.” [Online]. Available: http://cache.lifehacker.com/assets/resources/2008/03/bittorrent-schema.png

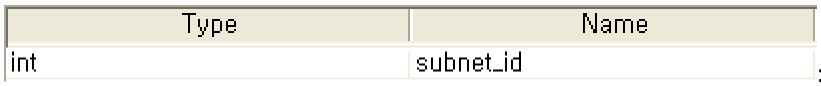
9. Appendix

Router Node #1
(11 receivers and transmitters)



Node model Process Model

State Variables:



Type	Name
int	subnet_id

:

Header Block:

```
#define PK_ARRVL (op_intrpt_type () == OPC_INTRPT_STRM)
#define OTHER_SUBNET_STRM 10
```

Init State Enter Execs:

```
Objid parent_subnet;
parent_subnet = op_topo_parent (op_topo_parent (op_id_self ()));
op_ima_obj_attr_get_int32 (parent_subnet, "user id", &subnet_id);
```

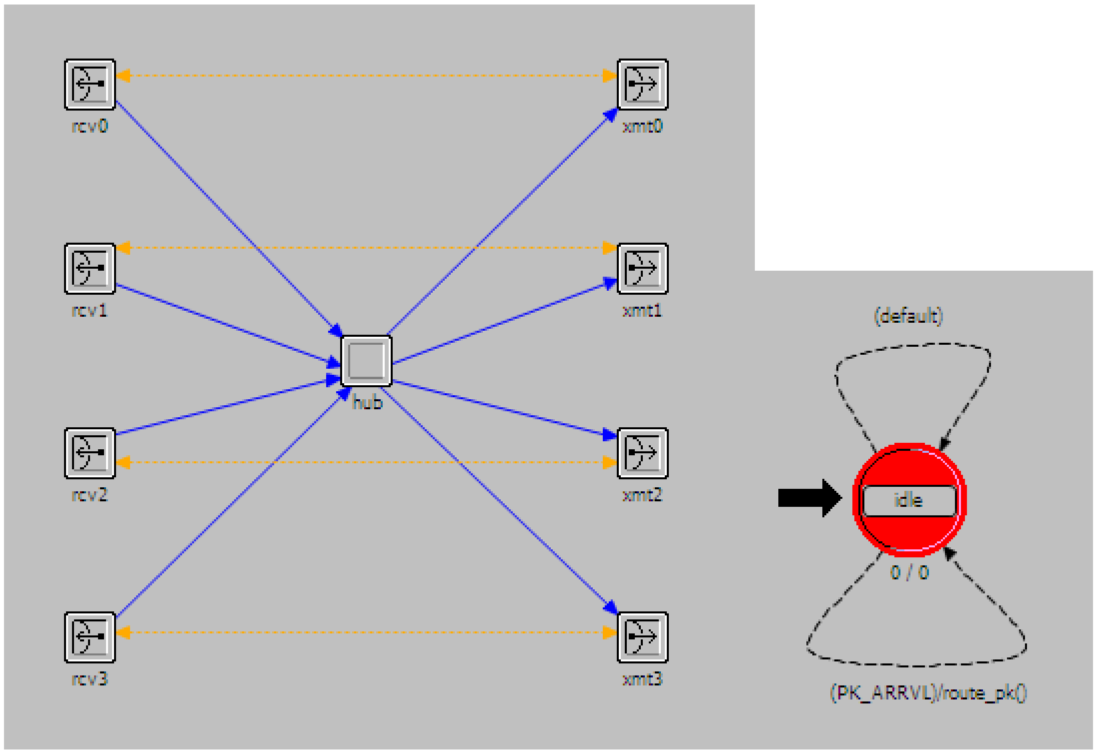
Function Block used for the small network:

```
static void route_pk (void)
{
        int dest_address;
        Packet * pkptr;
        FIN (route_pk ());
        pkptr = op_pk_get (op_intrpt_strm ());
        op_pk_nfd_get_int32 (pkptr, "dest_address", &dest_address);
        op_pk_send (pkptr, dest_address);
        FOUT;
}
```

Function Block used for the large network:

```
static void route_pk (void)
{
        int dest_address;
        int dest_subnet;
        Packet * pkptr;
        FIN (route_pk ());
        pkptr = op_pk_get (op_intrpt_strm ());

        // Read the destination subnet value
        op_pk_nfd_get_int32 (pkptr, "dest_subnet", &dest_subnet);

        // Check if the subnet id matches
        if (dest_subnet == subnet_id)
        {
                int dest_address;
                op_pk_nfd_get_int32 (pkptr, "dest_address", &dest_address);
                op_pk_send (pkptr, dest_address);
        }
        else
                op_pk_send (pkptr, OTHER_SUBNET_STRM);
        FOUT;
}
```

Router Node #2
(4 receivers and transmitters)



Node Model Process Model

Header Block:

```
#define PK_ARRVL (op_intrpt_type () == OPC_INTRPT_STRM)
#define OTHER_HUB 3
```

Function Block for router0:

```
static void route_pk (void)
{
        int dest_subnet;
        Packet * pkptr;
        FIN (route_pk ());
        pkptr = op_pk_get (op_intrpt_strm ());
        op_pk_nfd_get_int32 (pkptr, "dest_subnet", &dest_subnet);

        // If the destination subnet is larger than 2, send the packet to other router
        if(dest_subnet > 2)
        {
                op_pk_send(pkptr, OTHER_HUB);
        }
        else
        {
                op_pk_send (pkptr, dest_subnet);
        }
        FOUT;
    }
```

Function Block for router1:

```
    static void route_pk (void)
    {
        int dest_subnet;
        Packet * pkptr;
        FIN (route_pk ());
        pkptr = op_pk_get (op_intrpt_strm ());
        op_pk_nfd_get_int32 (pkptr, "dest_subnet", &dest_subnet);

        // Route the packet correctly
        if(dest_subnet == 4)
        {
            op_pk_send(pkptr, 1);
        }
        else if(dest_subnet == 3)
        {
            op_pk_send(pkptr, 0);
        }
        else if(dest_subnet < 3)
        {
            op_pk_send(pkptr, 3);
        }
        else
        {
            op_pk_send (pkptr, 2);
        }
        FOUT;
    }
```

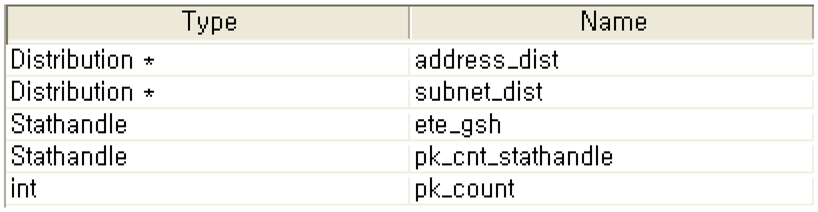
Function Block for router2:

```
    static void route_pk (void)
    {
        int dest_subnet;
        Packet * pkptr;
        FIN (route_pk ());
        pkptr = op_pk_get (op_intrpt_strm ());
        op_pk_nfd_get_int32 (pkptr, "dest_subnet", &dest_subnet);

        // If the destination subnet is smaller than 5, send the packet to the other router
        if(dest_subnet < 5)
        {
            op_pk_send(pkptr, 3);
        }
```

```
		else
		{
			op_pk_send (pkptr, dest_subnet - 5);
		}
		FOUT;
	}
```

Normal Server Node

State Variables:



Type	Name
Distribution *	address_dist
Distribution *	subnet_dist
Stathandle	ete_gsh
Stathandle	pk_cnt_stathandle
int	pk_count

Header Block:

```
	/* packet stream definitions */
	#define RCV_IN_STRM 0
	#define SRC_IN_STRM 1
	#define XMT_OUT_STRM 0

	/* transition macros */
	#define SRC_ARRVL (op_intrpt_type () == \
		OPC_INTRPT_STRM && op_intrpt_strm () == SRC_IN_STRM)

	#define RCV_ARRVL (op_intrpt_type () == \
		OPC_INTRPT_STRM && op_intrpt_strm () == RCV_IN_STRM)
```

Init State Enter Execs:

```
	address_dist = op_dist_load ("uniform_int", 0, 9);
	subnet_dist = op_dist_load ("uniform_int", 0, 7);
	ete_gsh = op_stat_reg ("ETE Delay", OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
	pk_count = 0;
	pk_cnt_stathandle = op_stat_reg ("packet count", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
```

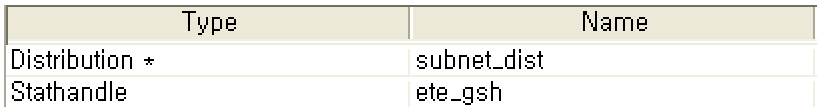
Function Block:

```
	// Transmit a packet to a client
	static void xmt (void)
	{
		Packet * pkptr;
		FIN (xmt ());
		pkptr = op_pk_get (SRC_IN_STRM);
		op_pk_nfd_set_int32 (pkptr, "protocol", 3);
		op_pk_nfd_set_int32 (pkptr, "dest_subnet", (int)op_dist_outcome (subnet_dist));
		op_pk_nfd_set_int32 (pkptr, "dest_address", (int)op_dist_outcome (address_dist));
```

```
        op_pk_send (pkptr, XMT_OUT_STRM);
        FOUT;
}

// Packet received; check the packet type and calculate the end-to-end delay
static void rcv (void)
{
        Packet * pkptr;
        double ete_delay;
        int protocol;

        FIN (rcv ());
        pkptr = op_pk_get (RCV_IN_STRM);
        op_pk_nfd_get_int32(pkptr,"protocol",&protocol);

        // If the packet type is non-P2P
        if(protocol == 3)
        {
                ete_delay = op_sim_time () - op_pk_creation_time_get (pkptr);
                op_stat_write (ete_gsh, ete_delay);
                op_pk_destroy (pkptr);
                ++pk_count;
                op_stat_write (pk_cnt_stathandle, pk_count);
        }
        FOUT;
}
```

Normal Client Node

State Variables:



Type	Name
Distribution *	subnet_dist
Stathandle	ete_gsh

Header Block:
(Same as the normal server node)

Init State Enter Execs:

```
subnet_dist = op_dist_load("uniform_int", 1, 2);
ete_gsh = op_stat_reg ("ETE Delay", OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
```

Function Block for the small network:

```
// Transmit a packet to the server
static void xmt (void)
{
        Packet * pkptr;
        FIN (xmt ());
        pkptr = op_pk_get (SRC_IN_STRM);
        op_pk_nfd_set_int32 (pkptr, "protocol", 3);
        op_pk_nfd_set_int32 (pkptr, "dest_address", 10);
        op_pk_send (pkptr, XMT_OUT_STRM);
        FOUT;
}

// Packet received; check the packet type and calculate the end-to-end delay
static void rcv (void)
{
        Packet * pkptr;
        double ete_delay;
        int protocol;

        FIN (rcv ());
        pkptr = op_pk_get (RCV_IN_STRM);
        op_pk_nfd_get_int32(pkptr,"protocol",&protocol);

        if(protocol == 3)
        {
                ete_delay = op_sim_time () - op_pk_creation_time_get (pkptr);
                op_stat_write (ete_gsh, ete_delay);
                op_pk_destroy (pkptr);
        }
        FOUT;
}
```

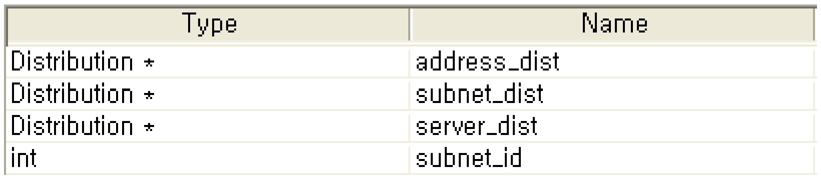
Function Block for the large network:

```
// Transmit a packet to the server
static void xmt (void)
{
        Packet * pkptr;
        int dest_subnet;
        FIN (xmt ());
        pkptr = op_pk_get (SRC_IN_STRM);
        op_pk_nfd_set_int32 (pkptr, "protocol", 3);
        // Server subnet addresses are two and four
        dest_subnet = (int)op_dist_outcome (subnet_dist) * 2;
        op_pk_nfd_set_int32 (pkptr, "dest_subnet", dest_subnet);
        op_pk_send (pkptr, XMT_OUT_STRM);
        FOUT;
}

// Packet received; check the packet type and calculate the end-to-end delay
static void rcv (void)
{
        Packet * pkptr;
```

```
		double ete_delay;
		int protocol;

		FIN (rcv ());
		pkptr = op_pk_get (RCV_IN_STRM);
		op_pk_nfd_get_int32(pkptr,"protocol",&protocol);

		// If the packet type is non-P2P
		if(protocol == 3)
		{
			ete_delay = op_sim_time () - op_pk_creation_time_get (pkptr);
			op_stat_write (ete_gsh, ete_delay);
			op_pk_destroy (pkptr);
		}
		FOUT;
	}
```

P2P Node

State Variables:



Type	Name
Distribution *	address_dist
Distribution *	subnet_dist
Distribution *	server_dist
int	subnet_id

Header Block:

```
	#define RCV_IN_STRM 0
	#define REQ_IN_STRM 1
	#define REP_IN_STRM 2
	#define SRC_IN_STRM 3
	#define XMT_OUT_STRM 0

	#define RCV_ARRVL (op_intrpt_type()==\
		OPC_INTRPT_STRM && op_intrpt_strm() == RCV_IN_STRM)
	#define REQ_ARRVL (op_intrpt_type()== \
		OPC_INTRPT_STRM && op_intrpt_strm() == REQ_IN_STRM)
	#define REP_ARRVL (op_intrpt_type()== \
		OPC_INTRPT_STRM && op_intrpt_strm() == REP_IN_STRM)
	#define SRC_ARRVL (op_intrpt_type()==\
		OPC_INTRPT_STRM && op_intrpt_strm() == SRC_IN_STRM)
```

Init State Enter Execs:

```
	// Get subnet id
	Objid parent_subnet;
	parent_subnet = op_topo_parent (op_topo_parent (op_id_self ()));
	op_ima_obj_attr_get_int32 (parent_subnet, "user id", &subnet_id);
```

```
address_dist = op_dist_load("uniform_int", 0, 9);
subnet_dist = op_dist_load("uniform_int", 0, 7);
server_dist = op_dist_load("uniform_int", 1, 2);
```

Function Block for the small network:

```
// SRC arrvied from packet generator; send the packet to random address
static void xmt (void)
{
	Packet * pkptr;
	FIN (xmt ());
	pkptr = op_pk_get (SRC_IN_STRM);
	op_pk_nfd_set_int32 (pkptr, "protocol", 3);
	op_pk_nfd_set_int32 (pkptr, "dest_address", 10);
	op_pk_send (pkptr, XMT_OUT_STRM);
	FOUT;
}

// REQUEST arrived from packet generator; send the packet to random address
static void req_xmt(void)
{
	Packet * pkptr;
	FIN(req_xmt ());
	pkptr = op_pk_get (REQ_IN_STRM);
	op_pk_nfd_set_int32(pkptr,"src_address",op_id_self());
	op_pk_nfd_set_int32(pkptr,"protocol",0);
	op_pk_nfd_set_int32(pkptr,"dest_address",(int)op_dist_outcome(address_dist));
	op_pk_send (pkptr, XMT_OUT_STRM);
	FOUT;
}

// REPLY arrived from packet generator; send the packet to random address
static void rep_xmt(void)
{
	Packet * pkptr;
	FIN(rep_xmt ());
	pkptr = op_pk_get (REP_IN_STRM);
	op_pk_nfd_set_int32(pkptr,"src_address",op_id_self());
	op_pk_nfd_set_int32(pkptr,"protocol",1);
	op_pk_nfd_set_int32(pkptr,"dest_address",(int)op_dist_outcome(address_dist));
	op_pk_send (pkptr, XMT_OUT_STRM);
	FOUT;
}

// Packet received; destroy the packet
static void rcv(void)
{
	Packet * pkptr;
	FIN(rcv ());
```

```
                pkptr = op_pk_get (RCV_IN_STRM);
                op_pk_destroy (pkptr);
                FOUT;
        }
```

Function Block for the large network:

```
        // SRC arrvied from packet generator; send the packet to random address
        static void xmt (void)
        {
                Packet * pkptr;
                int dest_subnet;
                FIN (xmt ());
                pkptr = op_pk_get (SRC_IN_STRM);
                op_pk_nfd_set_int32 (pkptr, "protocol", 3);
                // Server subnet addresses are two and four
                dest_subnet = (int)op_dist_outcome (server_dist) * 2;
                op_pk_nfd_set_int32 (pkptr, "dest_subnet", dest_subnet);
                op_pk_send (pkptr, XMT_OUT_STRM);
                FOUT;
        }

        // REQUEST arrived from packet generator; send the packet to random address
        static void req_xmt(void)
        {
                Packet * pkptr;
                FIN(req_xmt ());
                pkptr = op_pk_get (REQ_IN_STRM);
                op_pk_nfd_set_int32(pkptr,"src_subnet",subnet_id);
                op_pk_nfd_set_int32(pkptr,"src_address",op_id_self());
                op_pk_nfd_set_int32(pkptr,"protocol",0);
                op_pk_nfd_set_int32(pkptr,"dest_subnet",(int)op_dist_outcome (subnet_dist));
                op_pk_nfd_set_int32(pkptr,"dest_address",(int)op_dist_outcome(address_dist));
                op_pk_send (pkptr, XMT_OUT_STRM);
                FOUT;
        }

        // REPLY arrived from packet generator; send the packet to random address
        static void rep_xmt(void)
        {
                Packet * pkptr;
                FIN(rep_xmt ());
                pkptr = op_pk_get (REP_IN_STRM);
                op_pk_nfd_set_int32(pkptr,"src_subnet",subnet_id);
                op_pk_nfd_set_int32(pkptr,"src_address",op_id_self());
                op_pk_nfd_set_int32(pkptr,"protocol",1);
                op_pk_nfd_set_int32(pkptr,"dest_subnet",(int)op_dist_outcome (subnet_dist));
                op_pk_nfd_set_int32(pkptr,"dest_address",(int)op_dist_outcome(address_dist));
                FOUT;
```

```
}

// Packet received; destroy the packet
static void rcv(void)
{
        Packet * pkptr;
        FIN(rcv ());
        pkptr = op_pk_get (RCV_IN_STRM);
        op_pk_destroy (pkptr);
        FOUT;
}
```

BitTorrent Node

State Variable:
(Same as the P2P node)

Header Block:

```
#define RCV_IN_STRM 0
#define REQ_IN_STRM 1
#define REP_IN_STRM 2
#define SRC_IN_STRM 3
#define XMT_OUT_STRM 0
#define SNK_OUT_STRM 1

#define REQ_ARRVL (op_intrpt_type()== \
        OPC_INTRPT_STRM && op_intrpt_strm() == REQ_IN_STRM)
#define RCV_ARRVL (op_intrpt_type()==\
        OPC_INTRPT_STRM && op_intrpt_strm() == RCV_IN_STRM)
#define REP_ARRVL (op_intrpt_type()==\
        OPC_INTRPT_STRM && op_intrpt_strm() == REP_IN_STRM)
#define SRC_ARRVL (op_intrpt_type()==\
        OPC_INTRPT_STRM && op_intrpt_strm() == SRC_IN_STRM)

int conn_subnet;
int conn_address;
```

Init State Enter Execs:

```
// Get subnet id
Objid parent_subnet;
parent_subnet = op_topo_parent (op_topo_parent (op_id_self ()));
op_ima_obj_attr_get_int32 (parent_subnet, "user id", &subnet_id);
address_dist = op_dist_load("uniform_int", 0, 2);
subnet_dist = op_dist_load("uniform_int", 0, 7);
server_dist = op_dist_load("uniform_int", 1, 2);
conn_subnet = (int)op_dist_outcome(subnet_dist);
conn_address = (int)op_dist_outcome(address_dist);
```

Function Block for the small network:

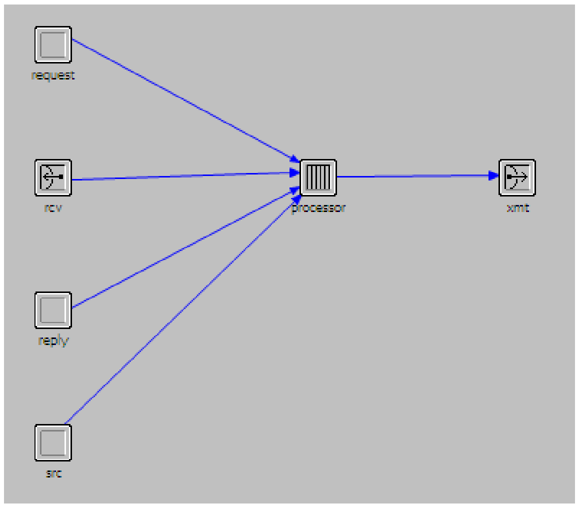
```
// SRC arrived from packet generator; send the packet to random address
static void xmt (void)
{
	Packet * pkptr;
	FIN (xmt ());
	pkptr = op_pk_get (SRC_IN_STRM);
	op_pk_nfd_set_int32 (pkptr, "protocol", 3);
	op_pk_nfd_set_int32 (pkptr, "dest_address", 10);
	op_pk_send (pkptr, XMT_OUT_STRM);
	FOUT;
}

// REQUEST arrived from packet geneartor; send the packet to random address
static void req_xmt(void)
{
	Packet * pkptr;
	FIN(req_xmt ());
	pkptr = op_pk_get (REQ_IN_STRM);
	op_pk_nfd_set_int32(pkptr,"src_address",op_id_self());
	op_pk_nfd_set_int32(pkptr,"protocol",0);

	// Send the request to only BitTorrent nodes
	op_pk_nfd_set_int32(pkptr,"dest_address",(int)op_dist_outcome(address_dist));
	op_pk_send (pkptr, XMT_OUT_STRM);
	FOUT;
}

// REPLY arrived from packet geneartor; send the packet to random address
static void rep_xmt(void)
{
	Packet * pkptr;
	int temp_address;
	FIN(rep_xmt ());
	pkptr = op_pk_get (REP_IN_STRM);
	op_pk_nfd_set_int32(pkptr,"src_address",op_id_self());
	op_pk_nfd_set_int32(pkptr,"protocol",1);

	// Check the source of received packet
	// send reply back only if the packet is from allowed node
	temp_address = (int)op_dist_outcome(address_dist);
	if(temp_address == conn_address)
	{
		op_pk_nfd_set_int32(pkptr,"dest_address",temp_address);
		op_pk_send (pkptr, XMT_OUT_STRM);
	}
	else
	{
```

```
                        op_pk_destroy (pkptr);
                }
                FOUT;
        }

        // Packet received; destroy the packet
        static void rcv(void)
        {
                Packet * pkptr;
                FIN(rcv ());
                pkptr = op_pk_get (RCV_IN_STRM);
                op_pk_destroy (pkptr);
                FOUT;
        }
```

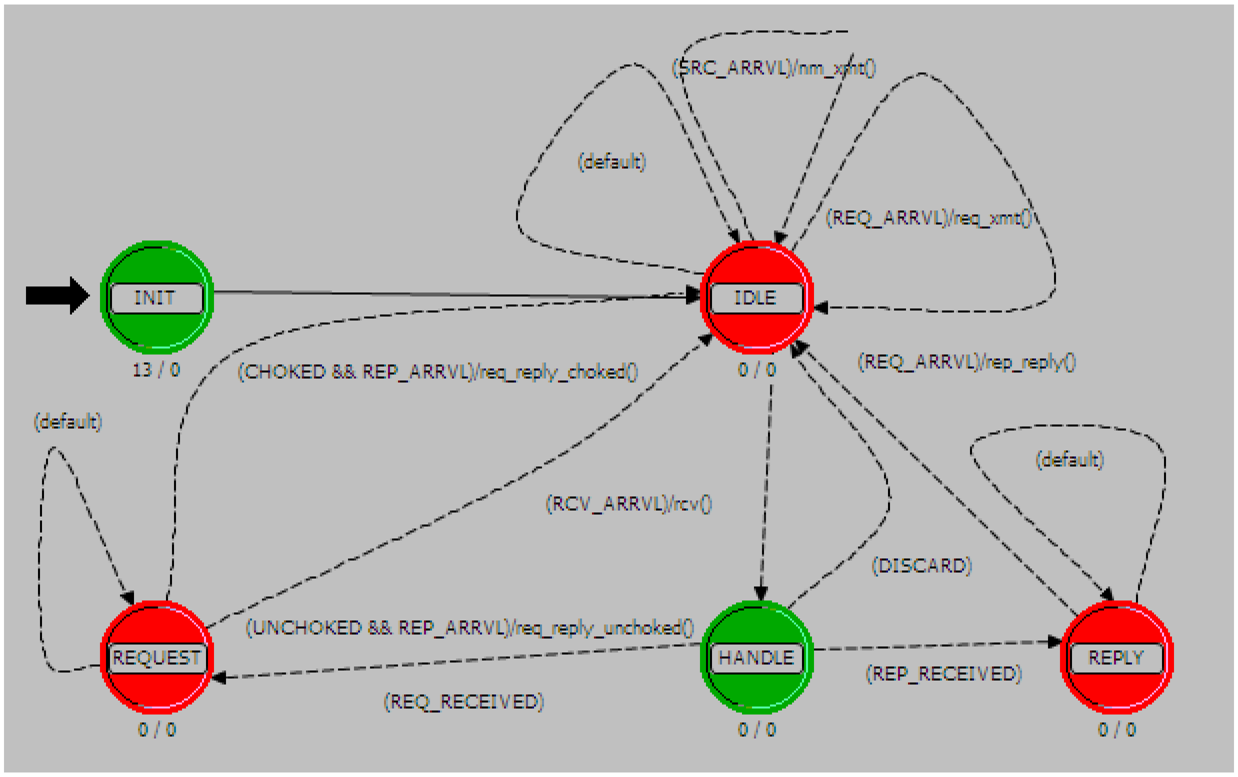
Function Block for the large network:

```
        // SRC arrvied from packet generator; send the packet to random address
        static void xmt (void)
        {
                Packet * pkptr;
                int dest_subnet;
                FIN (xmt ());
                pkptr = op_pk_get (SRC_IN_STRM);
                op_pk_nfd_set_int32 (pkptr, "protocol", 3);

                // Send the packet to servers only
                dest_subnet = (int)op_dist_outcome (server_dist) * 2;
                op_pk_nfd_set_int32 (pkptr, "dest_subnet", dest_subnet);
                op_pk_send (pkptr, XMT_OUT_STRM);
                FOUT;
        }

        // REQUEST arrived from packet geneartor; send the packet to random address
        static void req_xmt(void)
        {
                Packet * pkptr;
                int temp_subnet;
                FIN(req_xmt ());
                pkptr = op_pk_get (REQ_IN_STRM);
                op_pk_nfd_set_int32(pkptr,"src_subnet",subnet_id);
                op_pk_nfd_set_int32(pkptr,"src_address",op_id_self());
                op_pk_nfd_set_int32(pkptr,"protocol",0);

                // Send the request to only BitTorrent nodes
                temp_subnet = (int)op_dist_outcome(subnet_dist);
                if(temp_subnet == 2 || temp_subnet ==4)
                {
                        temp_subnet = temp_subnet + 1;
                }
                op_pk_nfd_set_int32 (pkptr,"dest_subnet",temp_subnet);
                op_pk_nfd_set_int32(pkptr,"dest_address",(int)op_dist_outcome(address_dist));
                op_pk_send (pkptr, XMT_OUT_STRM);
                FOUT;
        }

        // REPLY arrived from packet geneartor; send the packet to random address
        static void rep_xmt(void)
        {
                Packet * pkptr;
                int temp_subnet;
                int temp_address;
                FIN(rep_xmt ());
                pkptr = op_pk_get (REP_IN_STRM);
                op_pk_nfd_set_int32(pkptr,"src_subnet",subnet_id);
                op_pk_nfd_set_int32(pkptr,"src_address",op_id_self());
                op_pk_nfd_set_int32(pkptr,"protocol",1);

                // Check the source of the packet
                // Send reply back only if the packet received is from allowed source
                temp_subnet = (int)op_dist_outcome (subnet_dist);
                temp_address = (int)op_dist_outcome (address_dist);
                if(temp_subnet == conn_subnet && temp_address == conn_address)
                {
                        op_pk_nfd_set_int32 (pkptr,"dest_subnet",conn_subnet);
                        op_pk_nfd_set_int32 (pkptr,"dest_address",conn_address);
                        op_pk_send (pkptr, XMT_OUT_STRM);
                }
                else
                {
                        op_pk_destroy (pkptr);
                }
                FOUT;
        }

        // Packet received; destroy the packet
        static void rcv(void)
        {
                Packet * pkptr;
                FIN(rcv ());
                pkptr = op_pk_get (RCV_IN_STRM);
                op_pk_destroy (pkptr);
                FOUT;
        }
```

BitTorrent Node
(Version that we failed to build)

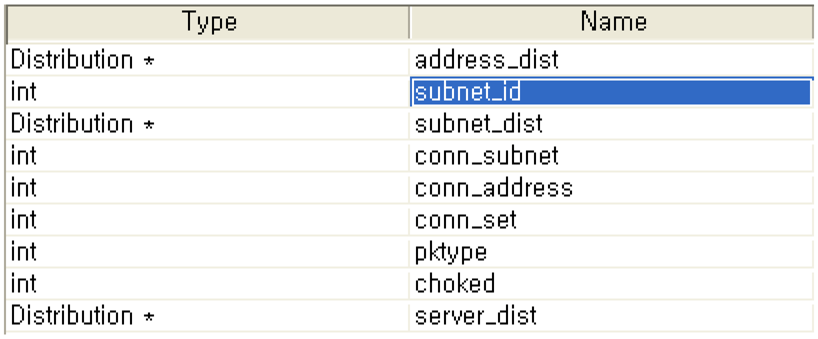


Node Model



Process Model

State Variables:



Type	Name
Distribution *	address_dist
int	subnet_id
Distribution *	subnet_dist
int	conn_subnet
int	conn_address
int	conn_set
int	pktype
int	choked
Distribution *	server_dist

Header Block:

```
/* packet stream definition */
#define RCV_IN_STRM 0
#define REQ_IN_STRM 1
#define REP_IN_STRM 2
#define SRC_IN_STRM 3
#define XMT_OUT_STRM 0

/* transition macros */
#define REQ_ARRVL (op_intrpt_type()== \
        OPC_INTRPT_STRM && op_intrpt_strm() == REQ_IN_STRM)

#define REP_ARRVL (op_intrpt_type()== \
        OPC_INTRPT_STRM && op_intrpt_strm() == REP_IN_STRM)

#define RCV_ARRVL (op_intrpt_type()==\
        OPC_INTRPT_STRM && op_intrpt_strm() == RCV_IN_STRM)

#define SRC_ARRVL (op_intrpt_type()==\
        OPC_INTRPT_STRM && op_intrpt_strm() == SRC_IN_STRM)

#define CHOKED (choked == 1)
#define UNCHOKED (choked == 0)
#define REQ_RECEIVED (pktype == 0)
#define REP_RECEIVED (pktype == 1)
#define DISCARD (pktype != 0 && pktype != 1)
```

Init State Enter Execs:

```
// Get the subnet id of the node
Objid parent_subnet;
parent_subnet = op_topo_parent (op_topo_parent (op_id_self ()));
op_ima_obj_attr_get_int32 (parent_subnet, "user id", &subnet_id);

// Set distribution
address_dist = op_dist_load("uniform_int", 0, 9);
subnet_dist = op_dist_load("uniform_int", 0, 7);
```

```
server_dist = op_dist_load("uniform_int", 1, 2);

// Initialize variables
pktype = 0;
choked = 0;
conn_subnet = 0;
conn_address = 0;
conn_set = 0;
```

Function Block:

```
// SRC(normal packet) arrived from packet geneartor; send the SRC to server
static void nm_xmt(void)
{
        Packet * pkptr;
        int dest_subnet;
        FIN (nm_xmt ());
        pkptr = op_pk_get (SRC_IN_STRM);
        op_pk_nfd_set_int32 (pkptr, "protocol", 3);
        dest_subnet = (int)op_dist_outcome (server_dist) * 2;
        op_pk_nfd_set_int32 (pkptr, "dest_subnet", dest_subnet);
        op_pk_send (pkptr, XMT_OUT_STRM);
        FOUT;
}

// REQUEST arrived from packet geneartor; send the REQUEST to random address
static void req_xmt(void)
{
        Packet * pkptr;
        FIN(req_xmt());
        pkptr = op_pk_get (REQ_IN_STRM);
        op_pk_nfd_set_int32(pkptr,"src_subnet",subnet_id);
        op_pk_nfd_set_int32(pkptr,"src_address",op_id_self());
        op_pk_nfd_set_int32(pkptr,"protocol",0);
        op_pk_nfd_set_int32(pkptr,"dest_subnet",(int)op_dist_outcome (subnet_dist));
        op_pk_nfd_set_int32(pkptr,"dest_address",(int)op_dist_outcome(address_dist));
        op_pk_send (pkptr, XMT_OUT_STRM);
        FOUT;
}

// REQUEST received; send REPLY(choked) back
static void req_reply_choked(void)
{
        Packet * pkptr;
        int src_subnet;
        int src_address;
        FIN(req_reply_choked());
        if (!op_subq_empty (0)) // If the subqueue is not empty
        {
```

```
                    // Dequeue the received packet, read the source, and reply back
                    pkptr = op_subq_pk_remove (0, OPC_QPOS_HEAD);
                    op_pk_nfd_get_int32(pkptr,"src_subnet",&src_subnet);
                    op_pk_nfd_get_int32(pkptr,"src_address",&src_address);
                    pkptr = op_pk_get (REP_IN_STRM);
                    op_pk_nfd_set_int32(pkptr,"src_subnet",subnet_id);
                    op_pk_nfd_set_int32(pkptr,"src_address",op_id_self());
                    op_pk_nfd_set_int32(pkptr,"protocol",2);
                    op_pk_nfd_set_int32(pkptr,"dest_subnet",src_subnet);
                    op_pk_nfd_set_int32(pkptr,"dest_address", src_address);
                    op_pk_send (pkptr, XMT_OUT_STRM);
          }
          FOUT;
}

// REQUEST received; send REPLY(unchoked) back
static void req_reply_unchoked(void)
{
          Packet * pkptr;
          int src_subnet;
          int src_address;
          FIN(req_reply_unchoked());
          if (!op_subq_empty (0)) // If the subqueue is not empty
          {
                    // Dequeue the received packet, read the source, and reply back
                    pkptr = op_subq_pk_remove (0, OPC_QPOS_HEAD);
                    op_pk_nfd_get_int32(pkptr,"src_address", &src_address);
                    op_pk_nfd_get_int32(pkptr,"src_subnet", &src_subnet);
                    pkptr = op_pk_get (REP_IN_STRM);
                    op_pk_nfd_set_int32(pkptr,"src_subnet",subnet_id);
                    op_pk_nfd_set_int32(pkptr,"src_address", op_id_self());
                    op_pk_nfd_set_int32(pkptr,"protocol",1);
                    op_pk_nfd_set_int32(pkptr,"dest_subnet",src_subnet);
                    op_pk_nfd_set_int32(pkptr,"dest_address", src_address);
                    op_pk_send (pkptr, XMT_OUT_STRM);
          }
          FOUT;
}

// REPLY received; send REQUEST again
static void rep_reply(void)
{
          Packet * pkptr;
          int src_subnet;
          int src_address;
          FIN(rep_reply());
          if (!op_subq_empty (0)) // If the subqueue is not empty
          {
            // Dequeue the received packet, read the source, and reply back
            pkptr = op_subq_pk_remove (0, OPC_QPOS_HEAD);
            op_pk_nfd_get_int32(pkptr,"src_address", &src_address);
            op_pk_nfd_get_int32(pkptr,"src_subnet", &src_subnet);
            pkptr = op_pk_get (REP_IN_STRM);
            op_pk_nfd_set_int32(pkptr,"src_subnet",subnet_id);
            op_pk_nfd_set_int32(pkptr,"src_address", op_id_self());
            op_pk_nfd_set_int32(pkptr,"protocol", 0);
            op_pk_nfd_set_int32(pkptr,"dest_subnet",src_subnet);
            op_pk_nfd_set_int32(pkptr,"dest_address", src_address);
            op_pk_send (pkptr, XMT_OUT_STRM);
        }
        FOUT;
}

static void rcv(void)
{
        Packet * pkptr;
        int protocol;
        int temp_subnet;
        int temp_address;
        FIN(rcv());
        pkptr = op_pk_get (RCV_IN_STRM);
        op_pk_nfd_get_int32(pkptr,"protocol",&protocol);

        // REQUEST RECEIVED; store the packet in the subqueue
        if(protocol == 0)
        {
            // Insert the new packet according to priority in subqueue
            if (op_subq_pk_insert (0, pkptr, OPC_QPOS_PRIO) != OPC_QINS_OK)
            {
                // If the insertion failed, discard the packet
                op_pk_destroy (pkptr);
                pktype = 3;
            }
            else
            {
                pktype = 0;
                choked = 1;
                op_pk_nfd_get_int32(pkptr, "src_subnet", &temp_subnet);
                op_pk_nfd_get_int32(pkptr, "src_address", &temp_address);

                // Check if the received packet is from the allowed source
                // If the packet is from the allowed source, send REPLY(unchoked)
                // If the packet is not from the allowed source, send REPLY(choked)
                if(temp_address == conn_address    && temp_subnet == conn_subnet)
                {
                    choked = 0;
```

```
                        }
                        else if(conn_set == 0)
                        {
                                    conn_set = 1;
                                    conn_subnet = temp_subnet;
                                    conn_address = temp_address;
                                    choked = 0;
                        }
            }

    }

    // REPLY RECEIVED; store the packet in the subqueue
    else if(protocol == 1)
    {
            if (op_subq_pk_insert (0, pkptr, OPC_QPOS_PRIO) != OPC_QINS_OK)
            {
                        op_pk_destroy (pkptr);
                        pktype = 3;
            }
            else
            {
                        pktype = 1;
            }
    }

    // CHOKED RECEIVED; discard the packet
    else if(protocol == 2)
    {
            op_pk_destroy (pkptr);
            pktype = 2;
    }

    // NORMAL PACKET RECEIVED; discard the packet
    else if(protocol == 3)
    {
            op_pk_destroy (pkptr);
            pktype = 3;
    }
    FOUT;
}
```