ENSC 427: COMMUNICATION NETWORKS

SPRING 2011

Prof. Ljiljana Trajkovic

Analysis of Video Streaming Over the BitTorrent Protocol

http://www.sfu.ca/~sha39/

Team 5

Milad Maleksabet – 301081041 – mma98@sfu.ca

Sasan Hezarkhani – 301049510 – sha39@sfu.ca

Abstract

Today BitTorrent protocol is widely popular for transfer of files. Because BitTorrent is a peer-to-peer network there is not one central server that files exist on. This means people share files with each other and each node (device connected to the internet) owns a specific chunk of the file. This requires the file data to be of a fixed size and known before transfer is started, so that the necessary parts of the file are downloaded without the need of them being in order. Because live streaming does not provide any of this information, more policies are required to allow BitTorrent to provide live streaming of video. Research shows that the change required is very minimal and not costly. In this project we will examine these changes and analyze how BitTorrent handles live video stream. Our concerns are mainly: stream speed vs. user group size, and stream buffering speed vs. video quality.

1. Introduction

1.1 Standard BitTorrent download vs. Client-Server download

It has been estimated that 27% to 55% of all Internet traffic is because of file transferring in BitTorrent. The main different between Torrent and classic download is that Torrent makes many small data request from TCP protocol from different machines, while in classic download client only makes data request from central server. The main advantage of BitTorrent protocol is that we can make connection with any machine (Low bandwidth or high bandwidth), and by increasing number of connections between nodes in the network we can increase the rate of data transfer. Figure.1 shows the overall topology for standard BitTorrent and client-server networks.

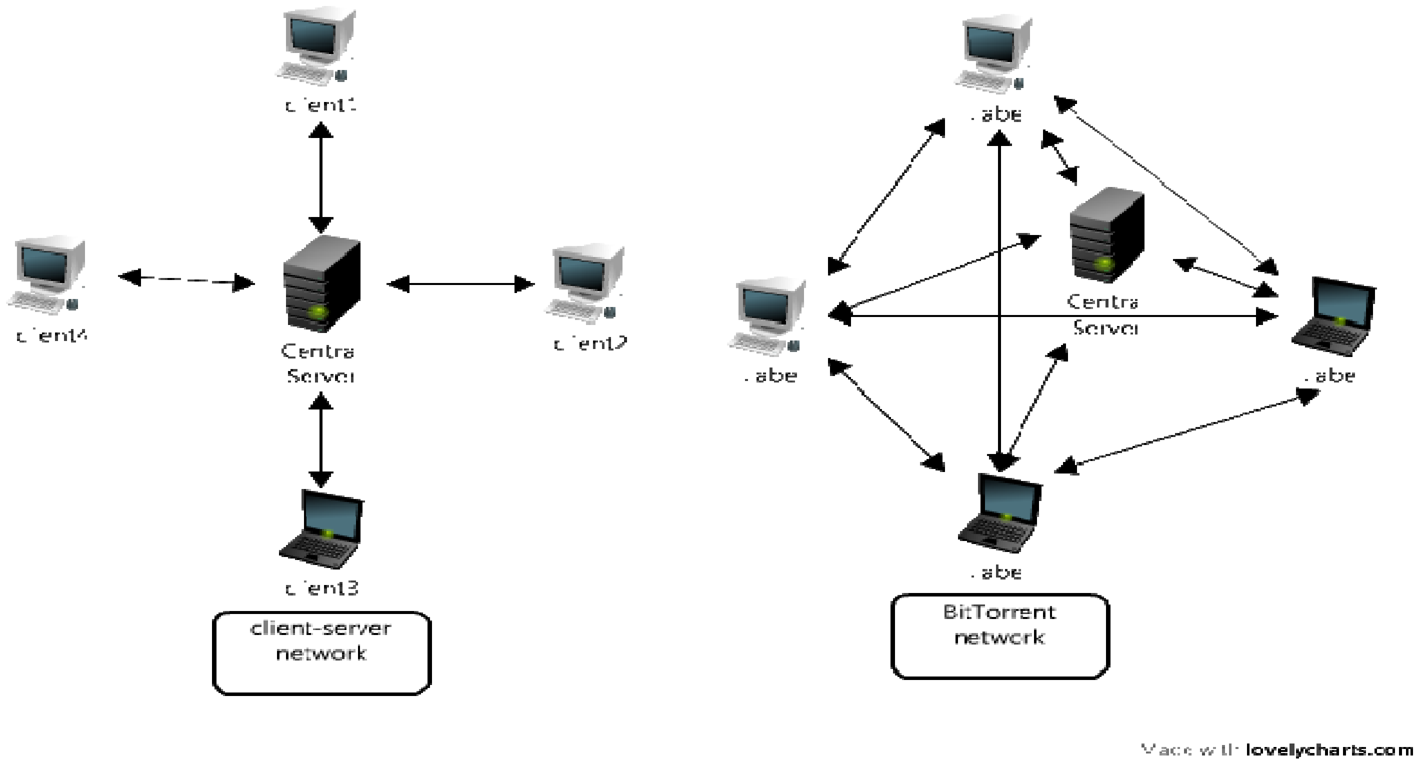

Figure 1: *structure of Normal Network & BitTorrent*

1.2 BitTorrent Structure

Terms:

- **Seeder:** A client which has entire copy of the file and can offer it for uploading. As number of seeders increases, there is more chance of getting a higher download speed.
- **Peer:** A client who does not have entire copy of the file, and it tries to download the file.
- **Tracker:** A server that keeps track of all peers and seeders which are downloading or uploading a file.
- **Swarm:** all peer and seeder connections in BitTorrent are called swarm.

The main idea of BitTorrent protocol is to divide the main file to smaller parts to make the process of transferring easier through the network. BitTorrent is peer to peer file sharing protocol where each peer in network can behave as server for other peers without the need for a central server.

In BitTorrent network, the client connects to the tracker to get list of swarm that are available to transfer the data. The client requests a part of main file from the list of swarm. If any of peers or seeders has the requested part, the client can connect directly to it and begins downloading. However, the effectiveness of data transferring depends on the method of determining to whom to send the data or which parts need to send first. Normally, the client with higher bandwidth has higher chance of getting the file first to improve overall uploading performance for other clients.

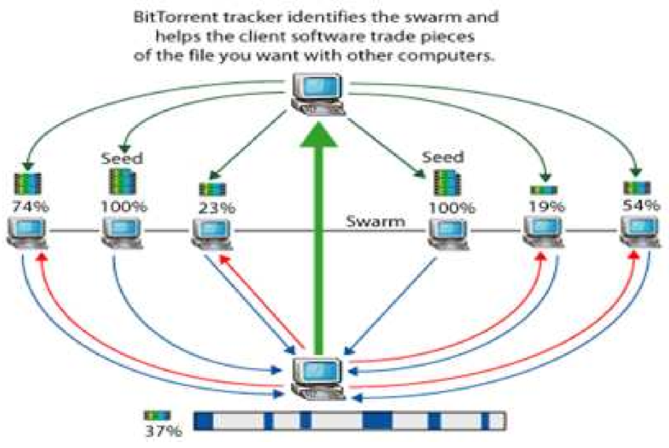
BitTorrent tracker identifies the swarm and helps the client software trade pieces of the file you want with other computers.
Seed
Seed
74%
100%
23%
Swarm
100%
19%
54%
37%

Figure 2: *BitTorrent overall topology* [8]

2. Video Streaming

2.1 Client-Server Live Video Streaming

Client-server model is a structure which a central server provides the data for other clients in the network. In recent years, client-server model has become an important aspect of live video streaming over the internet, and number of people who use video streaming over the client-server model is growing every year. Providing video streaming is a complex process and it can be very difficult and costly for telecommunication companies. These difficulties are:

- As number of client requests for video streaming increases, the main server can become overloaded, because of shortage in bandwidth.
- Providing high bandwidth connection for the main server can be very costly.
- There always is a limitation for number of clients that can use video streaming over the client-server network, because of the maximum number of connections that can be handled through the line.

An excellent solution that can be used to solve the problems which we have stated for client server model is using a peer to peer protocol. This protocol can solve the overload problem on the main server.

2.2 BitTorrent Live Video Streaming

The business of video streaming is growing rapidly. According to Bridge Ratings; there is more than a million video streaming on YouTube every day and number of viewers increase every day. Today, providing high bandwidth connection between client and server for live streaming video can be very costly and inefficient, thus by using a modified version of the BitTorrent protocol we can provide live video streaming without the heavy load on the source computer and maximize the bandwidth and redundancy for clients. In modified BitTorrent Network, the initial pieces transfer from the main machine to several clients, and then these pieces are individually transferred from the client to client thorough BitTorrent protocol. The main machine only needs to send one copy of file for several clients not the entire network. In modified BitTorrent network, the initial pieces transfer from the main machine to several clients, and then these pieces are individually transferred from the client to client thorough BitTorrent protocol.

2.2.1 BitTorrent modifications

Because BitTorrent is not designed to provide live streaming some modifications to the protocol is needed. We can categories these changes into four sections: file length, data validation, playback, and file providers [1]. We will examine each issue in the following sections and provide the solution that was chosen in this project.

2.2.1.1 File Length

BitTorrent protocol is designed so that each peer creates its data arrays based on the size of the desired file. This raises an issue in live streaming since a finite size is not defined for the video that is being downloaded. A very simple solution is to assume a very large file length that would never be reached. For example, each peer can assume that the file contains 2^{32} pieces. However, this solution is not practical because of memory limitations for each peer and seeder. A better approach is for each to have a sliding window of data that has been downloaded. This way data that fall out of this window are deleted and will never be requested by other peers on the network [1]. Figure 3 illustrates this solution.

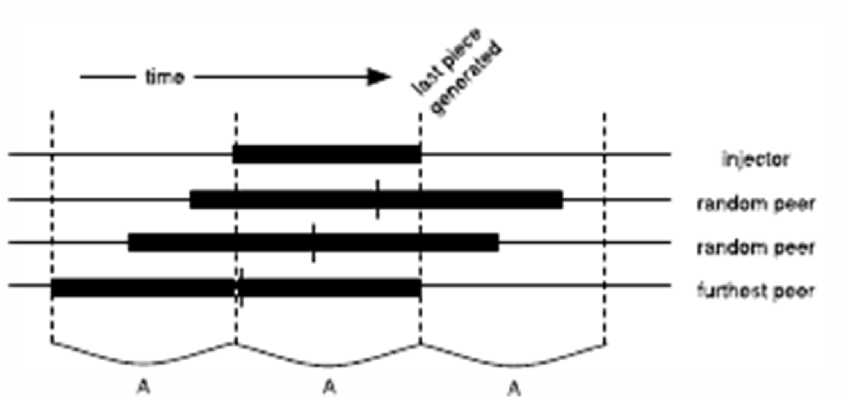

Figure3: *Sliding windows of each peer and the downloaded data. A is the length of the sliding window.*

2.2.1.2 Data Validation

To make sure of validity of the data that has been downloaded each peer does a validation at the end of the download. Data validation is a simple process in the normal BitTorrent client; a hash is generated based on the completely downloaded file which is then checked against the provided hash in the .torrent file. In the case of live streaming because the client never finishes downloading the file, data validation becomes a complex issue. Because we only wanted to focus on simulating and implementing the live stream feature we left data validation as a future work.

2.2.1.3 Playback

When a new peer starts to watch the live video it has to decide on which piece number to start downloading from. BitTorrent's default chunk selection is rarest piece first. This means each peer first scans through its chunks to find one that has least number of pieces downloaded. Next it sends a request to the available peers for this piece. While this can speed up the downloading process, because of its random nature it can't serve the live streaming purposes. In this project we choose to download each chunk sequentially for the least amount of lost packets and smooth play back. One of the draw backs of this solution is that the pre-buffering time of each peer can increase. This situation only happens when the latest available chunk to a peer is far from the actual latest available chunk on the media server. Thus, the peer is lagged behind and needs time to catch up to the stream's bit rate.

2.2.1.4 Video Providers

In the classic model of the BitTorrent protocol when a peer finishes downloading the file, it becomes a seeder to the other peers. In the case of the live streaming none of the peers that are watching the stream will ever finish downloading; hence no seeders are ever available. Therefore, we let seeders be clients which are never choked by the peers and are directly connected to the media server. Only these "trusted" peers are aware of the media server's address and act as the seeders in our model.

3. Simulations in ns2

3.1 Client-Server simulation using Goddard Streaming

Goddard streaming is an application based on the client server model. Goddard application is designed based on behaviors of real Networks Streaming media and Windows stream media [2].In Goddard application, the communication between clients and server is over both TCP and UDP. The overall design of Goddard application is shown in Figure 4.

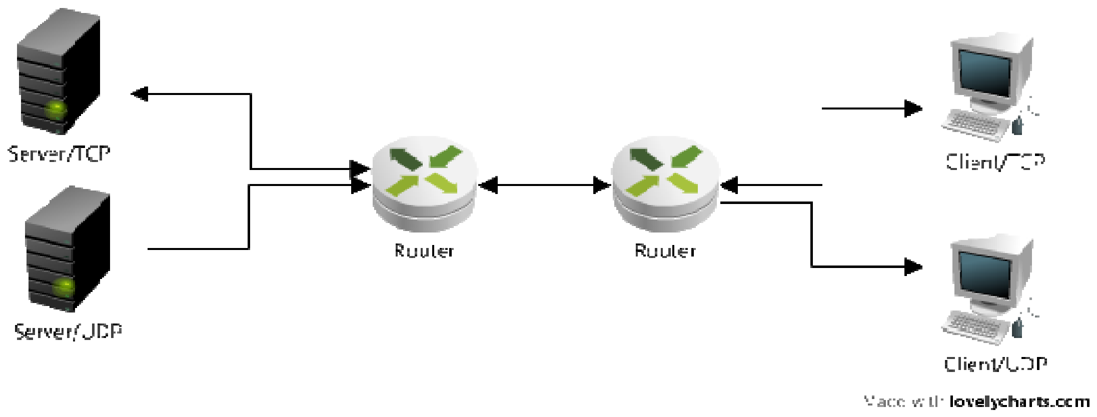

Figure 4: *Standard Network Topology of Goddard application*

In this project, the main structure of Goddard application was modified to a star topology and also communication links between nodes were changed to only use TCP, this allows us to compare our results with BitTorrent where TCP is also used. Figure 5 shows the modified version of Goddard application.

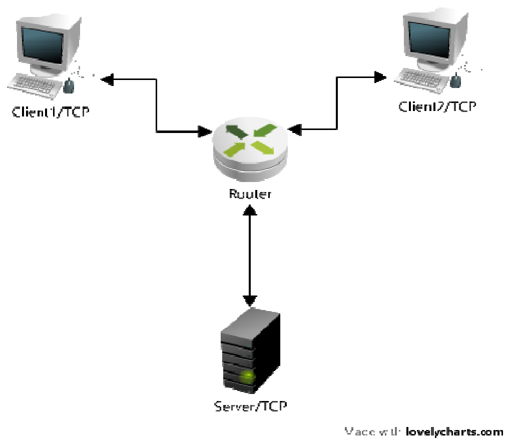

Figure 5: *Modified version of Goddard application.*

In the modified Goddard application, each client sends a request message to main server to establish the TCP connection. The main server replies back to each client with a connection ID. After establishing the connection, server sends the most recent piece of live video to each client. When a client receives a piece of the video, it sends a reply back massage to the main server asking for the next recent piece of the video. This process will continue until the live video streaming is done.

The simulation time for Goddard application was 200 seconds and each packet size was 10**Kb.** Number of clients which used was 20.

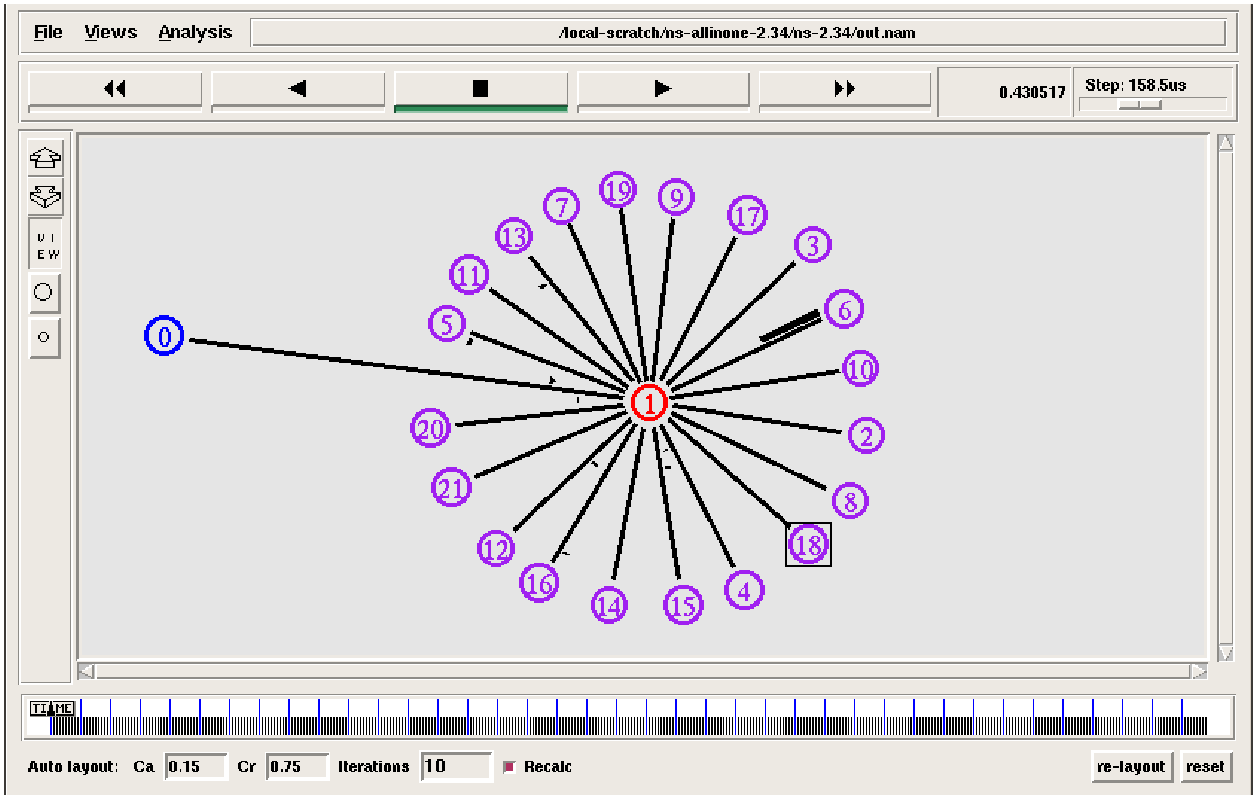

Figure 6: *Simulation of modified Goddard Application with 20 clients.*

Figure 7 and 8 show the results of our simulation using the above configuration.

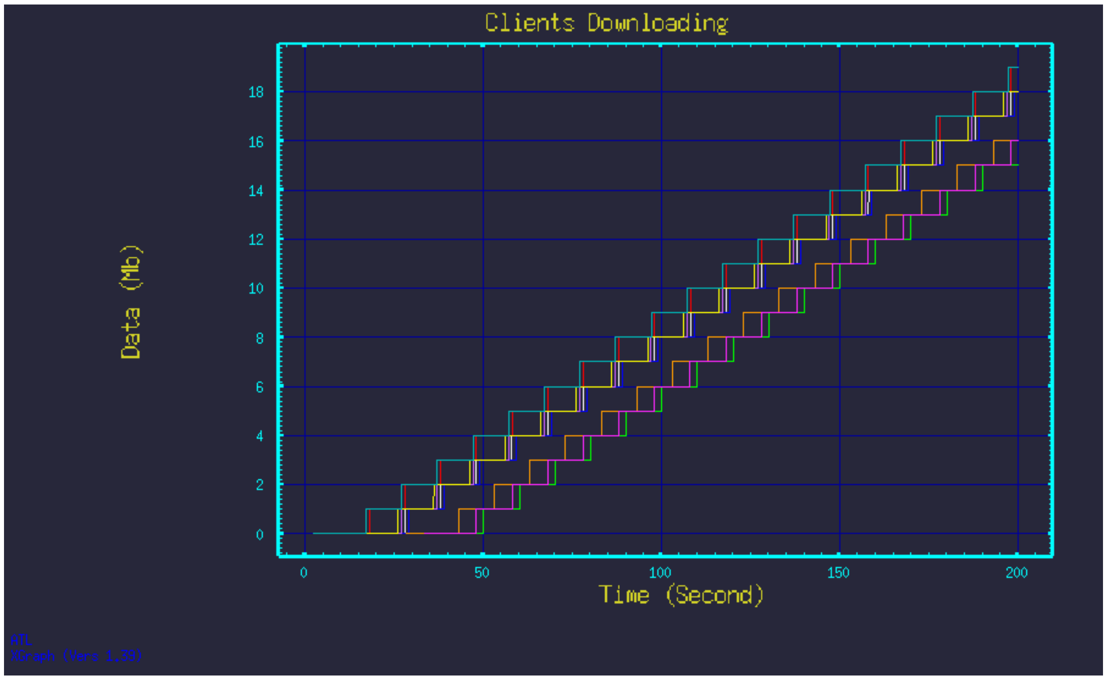

Figure 7: *Clients downloading the stream from a single server vs. time*

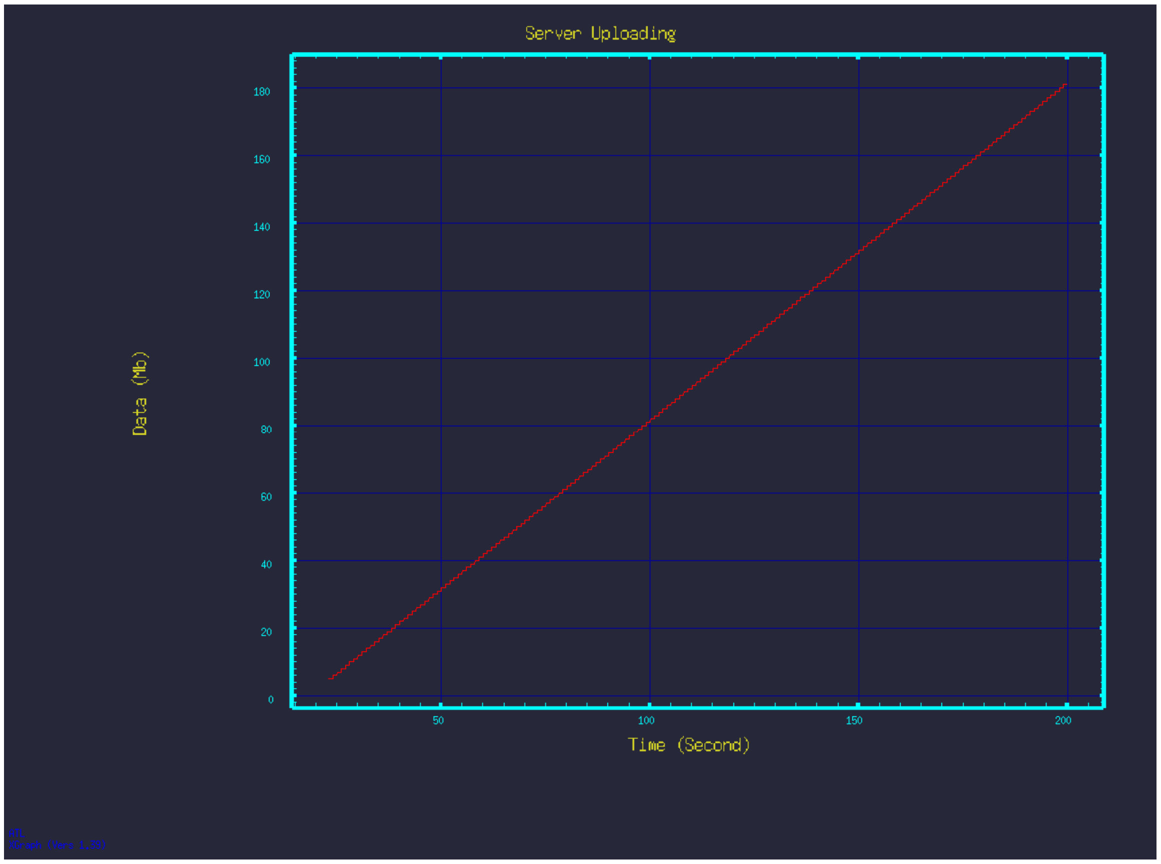

Figure 8: *Single server providing the data vs. time*

3.2 BitTorrent simulation

We use Kolja Eger's [3] implementation of BitTorrent in ns2 as a base to our project. We applied the changes that has been discussed in the previous section to this code base and simulated the network. Instead of using duplex connections we used two single connections for each node for downlink and uplink. All the peers and seeders are connected through a central router. Provided in Table 1 is some information about the network.

Table 1: Network properties used to simulate the BitTorrent video streaming.

Network Property	**Value**
Uplink Speed (Seeders)	100Mb
Downlink Speed (Clients)	Range 2Mb-100Mb
Uplink speed (Clients)	Range 512Kb-5Mb
Number of chunks	Fixed 5
Piece size	16384 Bytes (16KB)

For this simulation we used 10 peers and 1 seeder. Also there is a delay between each peer entering the swarm; this is to simulate what happens when a peer starts to watch the video at some random point throughout the stream. Figure 9 illustrates peer's downloaded data vs. time.

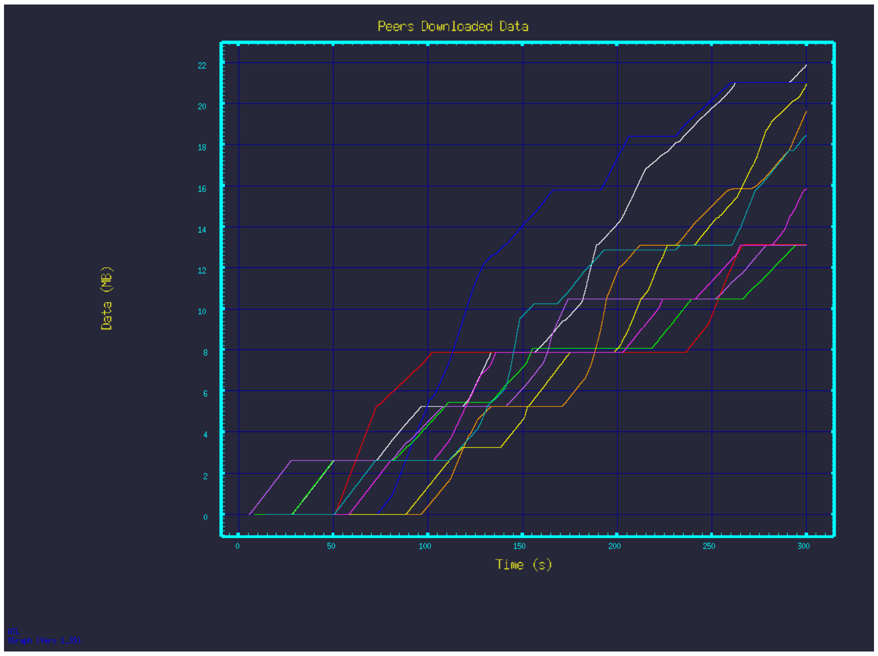

Figure 9: *Peer downloaded data (MB) vs. time (s)*

To see how the seeder is working in this network we also gathered the uploaded data by the seeder. Figure 10 provides this information.

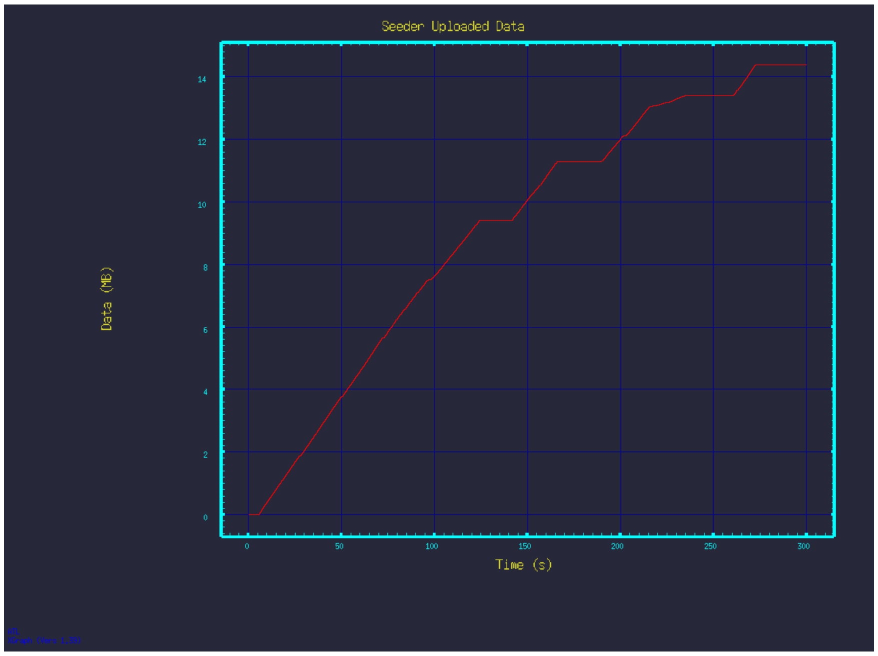

Figure 10: *Seeder uploaded data (MB/100) vs. time (s)*

These results are further discussed in section 3.3 where they are compared with the standard streaming option.

3.3 Comparison and Results

By comparing the results from both simulations we can see that BitTorrent video streaming can have advantages over the client-server model. We will go over the main differences that are observed in both clients and the server load.

3.3.1 Client downloading

First we will consider the peers who are downloading the video. The main difference in the simulations is the way that users download and buffer the data. In the standard model each user remains in buffering mode in between receiving times of the data, this rapid buffering could punish some users if the number of connections to the server is large. In the BitTorrent model we can see that users spend more time downloading and instantly playing the video, although some of these peers are seen to have some buffering time. Also, looking at start time of the stream for each peer we can see that BitTorrent clients are spending more time in pre-buffering when they first enter the pool to download the stream.

3.3.2 Server uploading

In addition to clients, servers have an importance in the streaming model chosen, because the companies who provide the streaming are mainly concerned in their cost for running such service. We can notice that in the client server model, the server is constantly providing the content to the entire network. However, in the BitTorrent model the graphs show that the server has less load on it and gets some resting moments because the peers are helping each other to provide the data.

4. Conclusions

In this project, we modified standard BitTorrent protocol to be able to do live video streaming. We used ns-2 version 2.29 and 2.34 to simulate both the modified version of BitTorrent and the standard client-server model. The results show that BitTorrent is a better option for the networks where number of client is large; however, client-server model is the best option when there are only a few clients connected to the server.

Furthermore, we concluded that by using BitTorrent, we can decrease the data load on the server. This advantage allows the server to have less bandwidth dedicated to video streaming. By having more bandwidth, servers are able to do multitasking; it means that providers can stream different types of data at the same time. Because this method is using less bandwidth, it is also more cost efficient.

5. Appendix

Note: Due to our long code, we have included only the TCL part in this report, you can find the complete code attached and on the web.

5.1 Goddard Streaming

GoddardStreaming.tcl

```
#NS simulator object
set ns [new Simulator]

set nf [open out.nam w]
$ns namtrace-all $nf


#Set random seed
global defaultRNG
$defaultRNG seed 100

#setting number of flows
set num_node 10

#setting distribution number for flow speed
set dist 2

#Making the two network nodes
set node_server [$ns node]
$node_server color blue
set node_router [$ns node]
$node_router color red

#making random generator

set RND [new RNG]
set speed [expr round([$RND uniform 1 100])]

#Making edge nodes for num_node
for {set i 0} {$i < $num_node} {incr i} {
    set node_($i) [$ns node]
    $node_($i) color pink

}

#Creating edge links for num_node
for {set i 0} {$i < $num_node/2} {incr i} {
      set speed [expr round([$RND uniform 1 50])]
      $ns duplex-link $node_router  $node_($i) [expr $speed]Mb 5ms DropTail
}

for {set i [expr ($num_node/2)]} {$i < $num_node} {incr i} {
      set speed [expr round([$RND uniform 50 100])]
      puts $speed
      $ns duplex-link $node_router  $node_($i) [expr $speed]Mb 5ms DropTail
}



#Creating the network link
$ns duplex-link $node_server $node_router 10Mb 5ms DropTail
set fq [[$ns link $node_server $node_router] queue]
$fq set limit_ 20
$fq set queue_in_bytes_ true
$fq set mean_pktsize_ 1000

#Open the trace files
set tfile_ [open out.tr w]
set clink [$ns link $node_server $node_router]
$clink trace $ns $tfile_

#Setup Goddard Streaming[expr $speed]


for {set i 0} {$i < $num_node} {incr i} {

      set gs($i) [new GoddardStreaming $ns $node_server $node_($i) TCP 1000
$i]
      set goddard($i) [$gs($i) getobject goddard]
      set gplayer($i) [$gs($i) getobject gplayer]
      $gplayer($i) set upscale_interval_ 1.0
      set sfile1_ [open stream-tcp.tr w]
      $gplayer($i) attach $sfile1_

}

#Scehdule Simulation
for {set i 0} {$i < $num_node} {incr i} {
    set t [expr round([$RND uniform 1 50])]
    ##$ns at [expr 12.5 * $i] "$goddard($i) start"
    $ns at $t "$goddard($i) start"
    $ns at 200.0 "$goddard($i) stop"

}
$ns at 200.0 "finish"

#Define a 'finish' procedure
proc finish {} {
        global f0 f1 f2 nf ns
        #Close the output files
      $ns flush-trace
      close $nf
        #Call xgraph to display the results
        exec xgraph graph/out0.tr graph/out1.tr graph/out2.tr graph/out3.tr
graph/out4.tr graph/out5.tr graph/out6.tr graph/out7.tr graph/out8.tr
graph/out9.tr &
        exit 0
}


$ns run
```

5.2 BitTorrent Protocol

BitTorrentStream_star.tcl

```
# BitTorrent P2P Simulation
# Flashcrowd
# PARAMETERS: N_P RNG_SEED C_up

global argv

# topology: STAR

if { $argc > 0 } {
       set i 1
        foreach arg $argv {

               if {$i==1} {
                      set no_of_peers $arg
               }
               if {$i==2} {
                      set s $arg
               }
               if {$i==3} {
                      set C_up [expr $arg * 1000]
               }
                 incr i
        }
}

if {$argc != 3} {
       puts "Error: wrong parameters ->  peers  run upload_cap\[kBits/s\]"
       exit 0
}

#Create a simulator object
set ns [new Simulator]
set nf [open star.nam w]
set f0 [open out0.tr w]
set f1 [open out1.tr w]
set f2 [open out2.tr w]

$ns namtrace-all $nf

remove-all-packet-headers
add-packet-header IP TCP Flags

$ns use-scheduler Heap

#set the routing protocol
$ns rtproto Manual
```

```
# Simulation Parameters:
       source bittorrent/bittorrent_default.tcl

       BitTorrentApp set leave_option -1

       # number of peers
       set N_P $no_of_peers

       # number of seeds
       set N_S 1

       # upload capacity in bytes
       set C_up_bytes [expr $C_up / 8.0 ]

       # factor that download capacity is higher than upload capacity
       set C_down_fac 8

       # queue size at access links (default 50)
       set Q_access 25

       # delay
       set DelayMin 1
       set DelayMax 50

       # file size
       set S_F_MB 100

       set S_F [expr $S_F_MB * 1024.0 *1024]
       set S_C [expr 256.0 *1024]
       #set N_C [format %.0f [expr ceil($S_F / $S_C)]]
       set N_c 5

       # set the seed for the RNG (0: non-deterministic, 1 - MAXINT (2147483647))
       set rng_seed $s

# End of SimulationParameters

set peerCount 0
set FinishedPeers 0


# NAME OF TRACE FILE
set p2ptrace bittorrent/results_flash_packet_star_
append p2ptrace $S_F_MB
append p2ptrace MB_N_P_
append p2ptrace $N_P
append p2ptrace _C_
append p2ptrace $C_up_bytes
append p2ptrace Bps
append p2ptrace _seed_
append p2ptrace $s
append p2ptrace _
append p2ptrace [clock seconds]

exec mkdir $p2ptrace
puts $p2ptrace

set p2ptrace2 $p2ptrace
append p2ptrace /log

exec cp bittorrent/scripts/bt_flashcrowd_star.tcl $p2ptrace2
exec cp bittorrent/bittorrent_default.tcl $p2ptrace2

set fh [open $p2ptrace w]


# set MSS for all FullTCP connections
Agent/TCP/FullTcp set segsize_ 1460
Queue set limit_ $Q_access

# Seed the default RNG
global defaultRNG
$defaultRNG seed $rng_seed
#puts [$defaultRNG seed]



proc attach-expoo-traffic { node sink size burst idle rate } {
	#Get an instance of the simulator
	set ns [Simulator instance]

	#Create a UDP agent and attach it to the node
	set source [new Agent/UDP]
	$ns attach-agent $node $source

	#Create an Expoo traffic agent and set its configuration parameters
	set traffic [new Application/Traffic/Exponential]
	$traffic set packetSize_ $size
	$traffic set burst_time_ $burst
	$traffic set idle_time_ $idle
	$traffic set rate_ $rate

	# Attach traffic source to the traffic generator
	$traffic attach-agent $source
	#Connect the source and the sink
	$ns connect $source $sink
	return $traffic
}

# Create Connections
proc fully_meshed2 {no_of_peers} {
	global ns peer router C_up C_down_fac DelayMin DelayMax sink


	set e2eDelayRng [new RNG]
	set e2eDelay [expr round([$e2eDelayRng uniform $DelayMin $DelayMax])]

	# upstream
	$ns simplex-link $peer($no_of_peers) $router $C_up [expr $e2eDelay]ms DropTail
	# downstream
	$ns simplex-link $router $peer($no_of_peers) [expr $C_down_fac * $C_up] [expr
$e2eDelay]ms DropTail

	# do the routing manually between peer and router
	[$peer($no_of_peers) get-module "Manual"] add-route-to-adj-node -default
[$router id]
	[$router get-module "Manual"] add-route-to-adj-node -default
[$peer($no_of_peers) id]

	[$router get-module "Manual"] add-route [$peer($no_of_peers) id] [[$ns link
$router $peer($no_of_peers)] head]

	[$peer($no_of_peers) get-module "Manual"] add-route [$router id] [[$ns link
$peer($no_of_peers) $router] head]
```

```
        return 0
}



proc done {} {
        global app FinishedPeers N_P fh ns nf f0 f1 f2 outFile

        incr FinishedPeers

        for {set i 0} {$i < $N_P} {incr i} {
                $app($i) stop
        }

        $ns flush-trace
        close $nf
        close $fh


        exec xgraph graph/out1.tr graph/out2.tr graph/out3.tr graph/out4.tr
graph/out5.tr graph/out6.tr graph/out7.tr graph/out8.tr graph/out9.tr &
        puts [$ns now]
        exit 0

}

proc record {} {
        #global sink0 sink1 sink2 f0 f1 f2
        #set ns [Simulator instance]
        #set time 0.5
        #set bw0 [$sink0 set bytes_]
        #set bw1 [$sink1 set bytes_]
        #set bw2 [$sink2 set bytes_]

        #set now [$ns now]

        #puts $f0 "$now [expr $bw0/$time*8/1000000]"
        #puts $f1 "$now [expr $bw1/$time*8/1000000]"
        #puts $f2 "$now [expr $bw2/$time*8/1000000]"


        #$sink0 set bytes_ 0
        #$sink1 set bytes_ 0
        #$sink2 set bytes_ 0

        #$ns at [expr $now+$time] "record"
}


# create tracker
# Parameters: File Size [B], Chunk Size [B]
set go [new BitTorrentTracker $S_F $S_C]
$go tracefile $p2ptrace


# uniform start offset for peers
set t_offset_rng [new RNG]
set t_offset [new RandomVariable/Uniform]
$t_offset set min_ 0
#[BitTorrentApp set choking_interval]
$t_offset set max_ 50
```

```
$t_offset use-rng $t_offset_rng


set router [$ns node]
$router shape box
$router color blue
$ns at 0.0 "$router label \"Router\""

$ns color 1 Red
$ns color 2 Blue
$ns color 3 Green
$ns color 4 Yellow
$ns color 5 Black
$ns color 6 White
$ns color 7 Purple

# Create Seeds
for {set i 0} {$i < $N_P} {incr i} {

       # make nodes
       set peer($i) [$ns node]
       ##set sink($i) [new Agent/LossMonitor]

       # make links
       fully_meshed2 $i

       if {$i < $N_S} {
              set app($peerCount) [new BitTorrentApp 1 $C_up $go $peer($i)]

              $app($peerCount) set super_seeding 1
              $app($peerCount) tracefile $p2ptrace


              # start apps
              $ns at 0.0 "$app($peerCount) start"

              $peer($i) color green
              $ns at 0.0 "$peer($i) label \"Media Server\""
              incr FinishedPeers
       } else {
              set app($peerCount) [new BitTorrentApp 0 $C_up $go $peer($i)]

              $app($peerCount) tracefile $p2ptrace

              # start apps
              $ns at [$t_offset value] "$app($peerCount) start"
              $ns at [$t_offset value] "$peer($i) label \"Peer Entered\""
       }


       incr peerCount
}

$ns at 300.0 "done"
# Run the simulation
$ns run
```

6. References

[1] J.J.D. Mol, A. Bakker, J.A. Pouwelse, D.H.J. Epema, and H.J. Sips “The Design and Deployment of a BitTorrent Live Video Streaming Solution”, 16 Sep. 2009. <http://pds.twi.tudelft.nl/pubs/papers/ism2009.pdf>.

[2] Jae Chung, Mark Claypool and Robert Kinicki “MTP: A Streaming-Friendly Transport Protocol”, 6 May 2007.

[3] Eger, Kolja. "BitTorrent in Ns-2." BitTorrent in Ns-2. Kolja Eger, 18 Mar. 2010. Web. 12 Feb. 2011. <http://sites.google.com/site/koljaeger/bittorrent-simulation-in-ns-2>.

[4] Rahul R. Pandey and KetanKumar Patil, “Study of Bit Torrent based Video on Demand Systems”, 22 Aug 2010. <http://www.ijcaonline.org/journal/number11/pxc387403.pdf>

[5] Saurabh Tewari, Leonard Kleinrock “Analytical Model for BitTorrent-based Live Video Streaming”, 11 Oct. 2006.

[6] Chung, Jae. "Goddard Streaming Media." Network Simulator, Ns-2 Blog. May 2008. Web. 10 Apr. 2011. <http://ns-2.blogspot.com/2007/05/streaming-media-system-for-ns-2.html

[7] Chung Jae, Mark Claypool, and Robert Kinicki. MTP: A Streaming-Friendly Transport Protocol, Technical Report WPI-CS-TR-05-10, Computer Science Department, Worcester Polytechnic Institute, May 2005.

[8] Image from <http://computer.howstuffworks.com/bittorrent2.htm>.