ENSC 427: COMMUNICATION NETWORKS

Comparison of TCP with "uTP" for BitTorrent transfers

Spring 2009

FINAL PROJECT

Adam Ciapponi	aciappon@sfu.ca
Robert Hueber	rhueber@sfu.ca
Robert Szolomicki	rms6@sfu.ca

http://www.sfu.ca/~aciappon/ENSC427/index.html

Abstract

BitTorrent is an application-layer protocol for peer-to-peer transfer of files, and is currently implemented over standard TCP in most of the applications supporting it [1]. Recently, a popular BitTorrent client, uTorrent, began testing a new version which would support a protocol called "uTP", a reliable transport service implemented over UDP that would transfer the BitTorrent application data between clients supporting the uTP protocol [2]. The stated goal of this change to minimize quality of service disruptions to other applications running on a user's internet connection by giving the BitTorrent client direct control of the congestion control parameters that are normally implemented by TCP. This project simulates the operation of BitTorrent on a network, using both the standard implementation and uTP, and examines how the change in protocol affects the quality of service for BitTorrent transfers and another application (VoIP) running on the same network.

Table of Contents

List of Figures

Glossary

.torrent	A file extension used by a BitTorrent client or other peer-to-peer programs that use the BitTorrent protocol
100Base-T	Ethernet over twisted pair running at 100Mb/s
ACK	a packet message used in the Transmission Control Protocol to acknowledge receipt of a packet
BitTorrent	a peer-to-peer file sharing (P2P) communications protocol
choked	A state of an uploader in a BitTorrent network. The transmitter doesn't currently want to send anything
Client	a piece of software that accesses services from another piece of software (a server)
DSL	Digital subscriber line. A family of technologies that provides digital data transmission over the wires of a local telephone network
FTP	File Transfer Protocol. A network protocol used to exchange and manipulate files over a TCP computer network
HTTP	Hypertext Transfer Protocol
IPv4	Internet Protocol version 4. The fourth revision in the development of the Internet Protocol (IP)
Leecher	A peer on a peer-to-peer network downloading content from a seeder without providing the file to the community simultaneously
MSS	Maximum segment size. The largest amount of data that a communications device can accommodate in a single, unfragmented segment
MTU	Maximum transmission unit. The size of the largest packet that a network protocol can transmit
OPNET	Software program used to simulate computer networks
p2p	a type of ad-hoc computer network
peer-to-peer	See p2p
Server	A server dedicated to running certain software applications for client's to access
T1	Digital Signal 1. A telecommunications standard, sometimes called a "T1 Line"

TCP	Transmission Control Protocol. A Transport layer protocol that is one of the core protocols of the Internet protocol suite
torrent	a file used by a BitTorrent client or other peer-to-peer programs that use the BitTorrent protocol
Tracker	A server on the Internet that acts to coordinate the action of BitTorrent clients
UDP	User Datagram Protocol. A simple transport protocol used in the Internet
unchoked	A state of an uploader in a BitTorrent network. Where the uploader is not choked.
uTorrent	µTorrent (or uTorrent) is a freeware closed source BitTorrent client by BitTorrent, Inc.
uTP	Micro Transport Protocol. A protocol based upon the User Datagram Protocol (UDP) for transmitting torrent files
VoIP	Voice over Internet Protocol. The technology used in the delivery of voice communications over IP networks such as the Internet or other packet-switched networks

1. Introduction

The BitTorrent protocol is a popular application level protocol for enabling a group of clients, assisted by a coordinating server known as the *tracker*, to create peer-to-peer network over the internet for the purpose of transferring a file or set of files. A brief description of the operation of the protocol follows [1]:

- A BitTorrent client is a computer running an application supporting the BitTorrent protocol. The client may have all, some, or none of the pieces of the file(s) for the particular BitTorrent network it will join.
- The client downloads a *.torrent* file, typically from a server over HTTP. This file contains information on the tracker and on the file(s) to be transferred over the particular BitTorrent network.
- Using the information in the .torrent file, the client connects to the tracker, informs the tracker that it (using a self-selected peer ID, IP address, and port) is participating in this particular BitTorrent network, and receives a list (of peer IDs, IPs, and ports) of other clients. This occurs not only when joining the torrent, but also periodically (generally every 15 minutes by default but can be adjusted manually in the uTorrent client) while the client is part of the BitTorrent network.
- Clients connect to other peers via TCP and exchange messages conforming to the BitTorrent protocol specification, which may include pieces of the file being sent from one of the peers to another.

It is important to note that the downloading of a .torrent file occurs only once, and connections to the tracker occur infrequently, and both involve the client downloading a very modest (tens of KBs) amount of information. While modelling the tracker scrape process may be important to understanding how the BitTorrent network topology changes on a timescale of hours, it is relatively un-important for assessing Quality of Service (QoS), as this will be determined mostly by how the transfer of data occurs between whichever peers happen to be connected at a given time. Thus, for the purposes of this project's simulations, the emphasis is placed on modelling the peer-to-peer transfer process.

The flow control for peer to peer data provided by the BitTorrent protocol is relatively rudimentary. Each connection a client maintains with a peer may be in either a "choked" or "unchoked" state. If peer A puts its connection to peer B in an unchoked state (by sending the appropriate message), and it is "interested" in receiving data, peer B should send peer A pieces.

Otherwise, if the connection is “choked”, no transfer takes place. Each peer’s choke status is changed at most every 10 seconds.

While this method is sufficient to ensure that a client’s sending of pieces to other peers does not cause so much congestion so as to reduce the download rate, it doesn’t provide enough control to allow for good co-existence with other applications on the client computer. For example, web browsing over HTTP often experiences significant slowdowns if BitTorrent is also being used on the same client computer. In order to alleviate these problems, some BitTorrent applications provide a means of throttling incoming and outgoing data transfer by limiting it to a specific numeric rate. By choosing a rate somewhat lower than capabilities of that computer’s connection to the internet, disruption to other applications on the client computer can be minimized.

Unfortunately, this method has many limitations. Firstly, it will under-utilize the connection when no other applications are transferring data, and will lead to QoS degradation if the other applications transfer too much. Additionally, the capabilities of a user’s internet connection may change over time due to the shared nature of resources on many home internet connections; however, the static rate limiting method provides no means for dynamic adjustment to these changing conditions.

As an answer to these shortcomings, the BitTorrent client “uTorrent” has implemented a protocol known as “uTP” [2]. In uTP, UDP packets (instead of TCP) are used to transfer data between peers, and connection-oriented reliable transfer, as well as congestion control, is implemented by the application layer. By moving these features into the application layer, the developers of uTorrent hope to better maximize transfer rates for BitTorrent data while at the same time minimizing QoS disruptions to other applications that may be running on the client computer [3].

The aim of this project is to analyze and compare the QoS for BitTorrent and an example application in (VoIP) in two scenarios: One using a simulated version of the standard BitTorrent protocol, and one using a simulated version of uTP. This comparison of QoS will offer an insight into the advantages and disadvantages of uTP.

2. Creation of the OPNET Model

2.1. Network Topology

The uTP protocol has several major differences from the normal implementation of BitTorrent [3]. Firstly, the transport protocol is replaced with UDP; however to provide the type of service required by the BitTorrent application, an intermediate protocol in the application layer is required. This application-level protocol is connection-oriented, and provides the reliable

transfer service required by BitTorrent traffic. The combination of UDP and this application layer protocol, referred to in this report as “pseudo-TCP”, forms a new protocol stack that transfers BitTorrent data in a novel way. This new method of transfer is the uTP protocol. A comparison of the protocol stacks can be seen below in Figure 1.

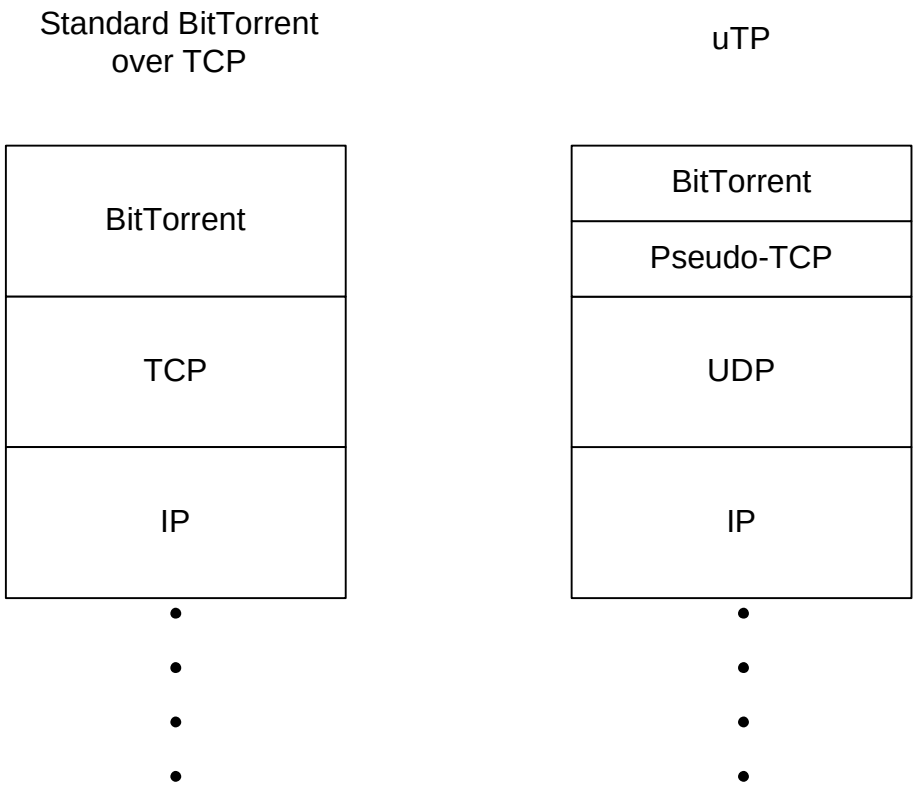

Figure 1: Protocol Stack for Standard BitTorrent and uTP

The main difference that uTP has over TCP is that the rules for congestion control are tweaked in order to attempt to emulate the design goal of improving QoS within the network. [3]

In order to appropriately model a peer-to-peer BitTorrent network, an OPNET model was created with both clients and servers to emulate seeders (peers which only upload to other peers) and leechers (peers which may upload and download to other peers). The model consists of a campus network with an IP cloud, 5 server subnets which are prefixed with “S” and 5 client subnets which are prefixed with “C”. These subnets are connected to the IP cloud through T1 lines as the capabilities of a T1 line (~1.5MBps) are, in the authors’ experience, broadly similar to the capabilities of residential DSL connections. Although BitTorrent peers are connected on a global scale, we chose to compare the protocols on a smaller, campus-sized network keeping in mind that this isn’t a campus network but a network on a campus-wide scale. The overall campus network is shown in Figure 2.

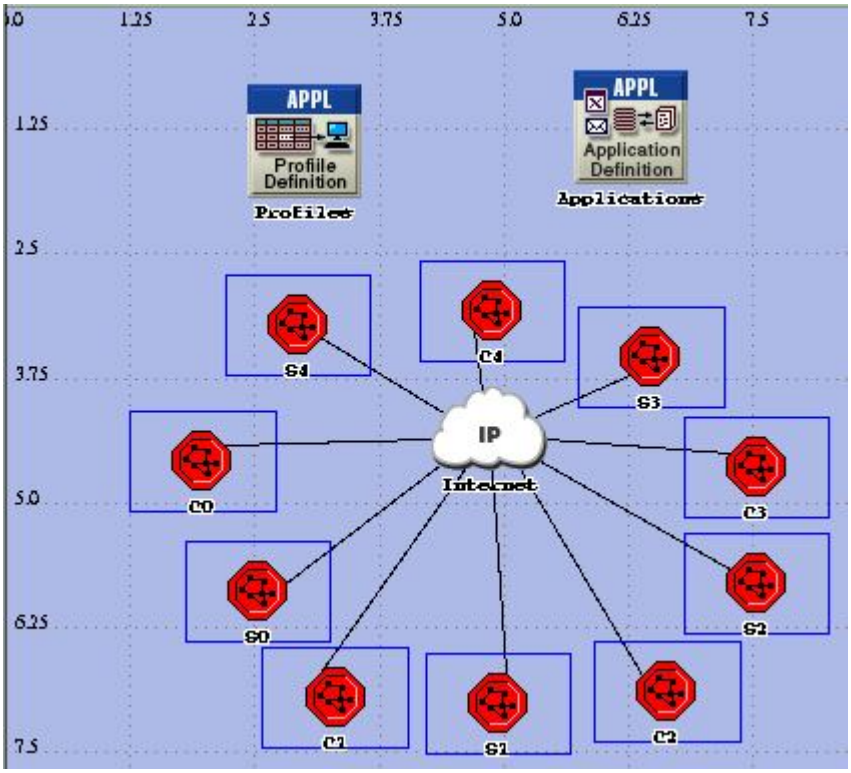

Figure 2: Campus Network

The client subnets each consist of three workstations attached to a router through 100Base-T Ethernet lines; the router is then attached to the T1 line connected to the IP cloud. These workstations act as leechers on the BitTorrent network. In an actual BitTorrent network, these workstations would be uploading and downloading to peers simultaneously, but due to limitations with custom applications in OPNET, the stations were modeled as only downloading. The reasons for this are discussed below. A client subnet is shown in Figure 3.

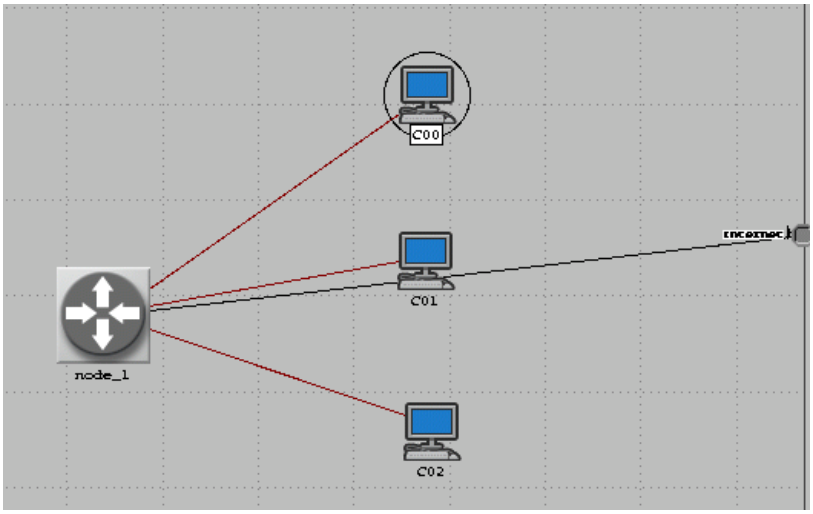

Figure 3: Client Subnet

The server subnet consists of a single server which is attached to a router through a 100Base-T Ethernet connection which is then connected to the IP cloud. This server models seeders who are only uploading data to the network and as such this closely approximates an actual BitTorrent seeder. A server subnet is shown in Figure 4.

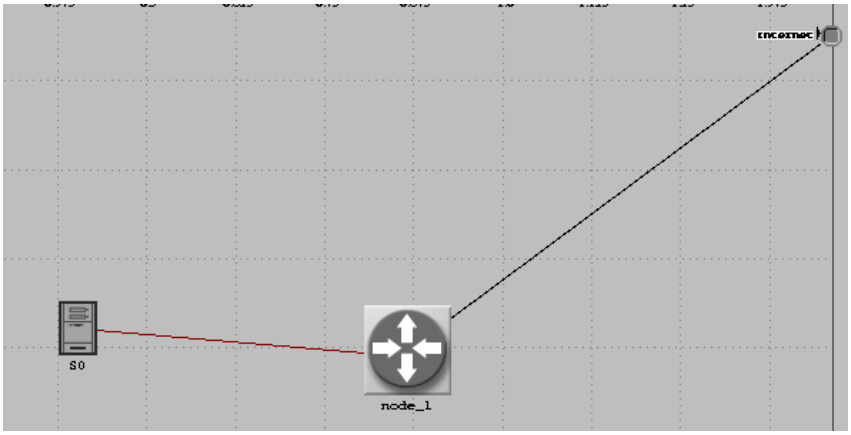

Figure 4: Server Subnet

The number of clients and servers was chosen to appropriately represent the number of seeders and leechers on a typical torrent. As a torrent starts, there is only one seeder present. Leechers will then join and when they complete their download they will act as seeders. Some peers will also leave the BitTorrent network once finished in order to avoid uploading too much data. As the torrent becomes more mature the ratio of seeders to leechers can stabilize between 0.25 to 0.5, or in other words 2 to 4 leechers for every seeder. The OPNET model consists of 3 leechers to every seeder for a seeder to leecher ratio of 0.33. Other work related to p2p networks in OPNET showed a similar architecture but at a more complicated level that compared multiple different applications with p2p traffic. [9]

2.2. Application Profiles and Definitions

There were two applications used in the uTP analysis; FTP and VoIP. FTP was chosen to model the behaviour of file transfers from a seeder to a leecher. One limitation is that the FTP traffic is almost completely one-way from the servers to the client, and thus does not represent the complete behaviour of BitTorrent where two peers may be exchanging data bi-directionally. While this behaviour could have been modeled with a custom application, there were significant implementation problems which ultimately made use of custom applications unfeasible. VoIP was used as a "co-existence" application to model the effects on QoS other applications that the transfer of BitTorrent data with regular TCP and with uTP create. The description of the FTP application is shown below in Figure 5.

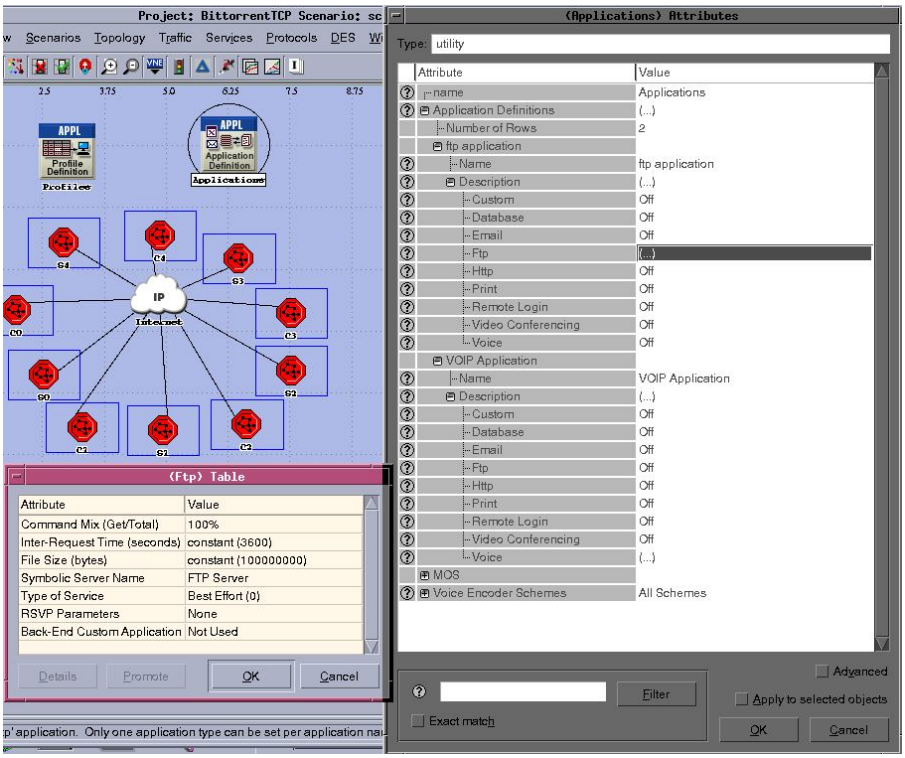

Figure 5: Application Definition (FTP details)

This FTP file transfer sends a 100Mb file one time during the entire simulation in order to bring the network utilization to a saturation point. Each seeder and leecher uses this FTP definition. The inter-request time was set to one hour to ensure the simulation is over before another file is sent. This allows for the observation of one file transferred per seeder/leecher group. I.e., s2 and c2 subnets interact with each other as the server in s2 provides the 100Mb file to the three clients in the c2 subnet. The same thing happens with the s3, c3 subnet pair and so on.

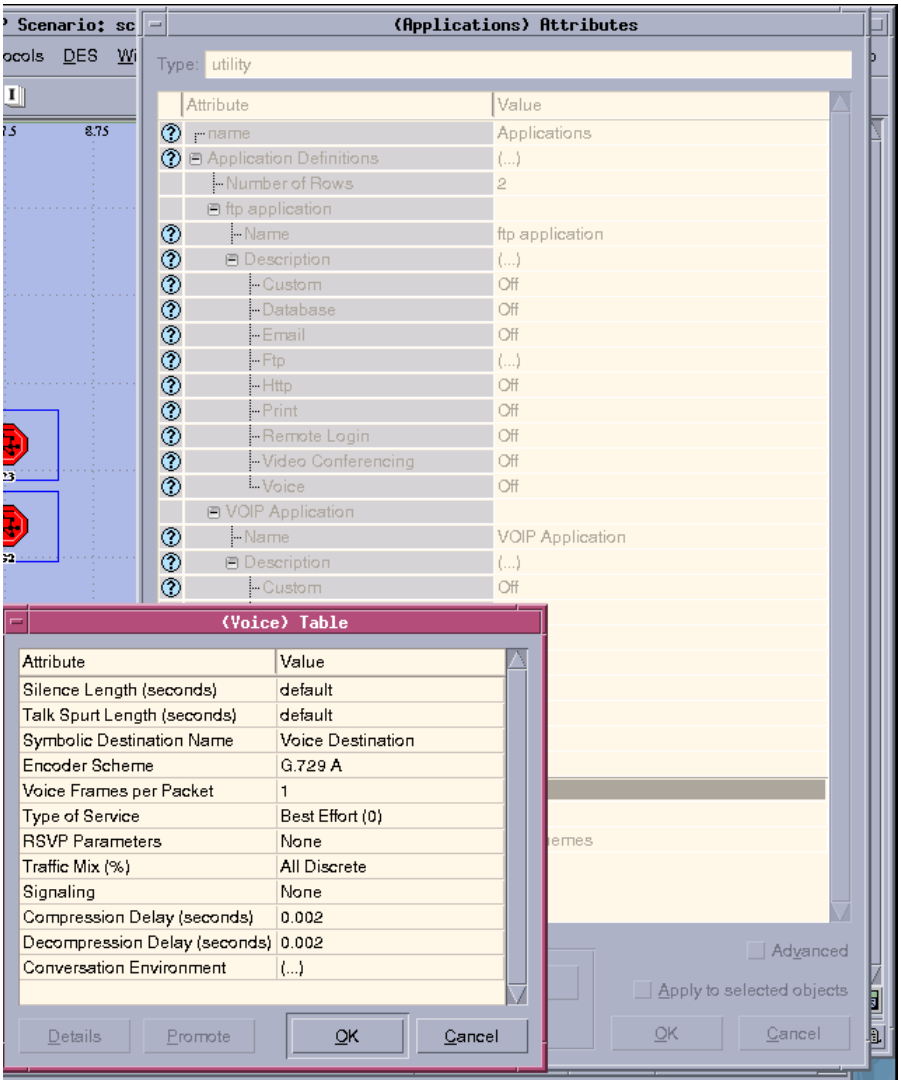

Figure 6: Application Definition (VoIP details)

The VoIP application details were all kept to their default parameters as no extra complexity was required to be introduced. Only one VoIP call, between S0 and C00, is active in this scenario, and statistics for both the FTP traffic and the VoIP traffic were gathered specifically for this client-server pair.

This network setup formed the baseline scenario, where VoIP traffic travels between two peers which are also part of a BitTorrent network.

This scenario was then cloned and modified to create a second scenario which represented uTP. In order to represent two peers with the uTP changes, S0 and C00 had their TCP parameters modified. Below in Figure 7 is the description of the new uTP changes that were to be made to the two peers.

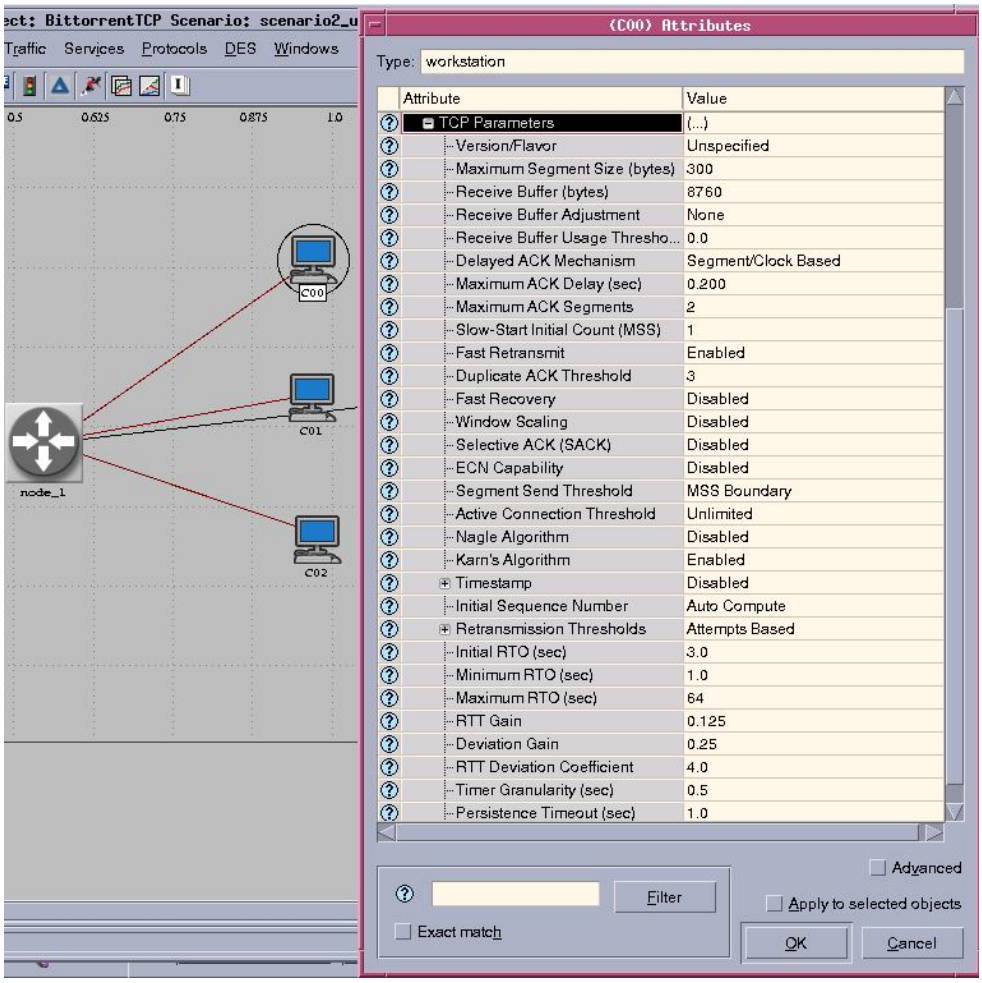

Figure 7: Changes made to TCP to model uTP

While simulating a protocol that travels over UDP using modified TCP may at first seem strange, it is important to remember that that uTP has an application level protocol that operates in almost the same manner as TCP. Since UDP offers no congestion control at all, and congestion control is the focus of this project, simulating uTP using general UDP traffic is not an option. Instead, the pseudo-TCP/UDP stack is simulated using TCP with modified parameters.

2.2.1. Segment Size

The maximum segment size (MSS) is the largest amount of data (in bytes) that a communications device can handle in a single unfragmented piece. In TCP and IPv4, the MSS is calculated by using the maximum transmission unit (MTU) of Ethernet and the header size of both TCP and IPv4. TCP and IPv4 headers are both 20 bytes each and the MSS is equal to the MTU minus the header sizes of both IPv4 and TCP added together which is 1460 bytes [5]. If a data segment is larger than the MTU, the datagram must be fragmented as per RFC 791 [4]. In the version of uTP that was in development at the time of the project, the packet size was set to a fixed 300 bytes, so in this OPNET simulation, the MSS was set to 300 bytes [3].

2.2.2. Fast Recovery

The fast recovery parameter refers to the algorithm executed during congestion avoidance conditions when packets are dropped. When a packet is dropped, a duplicate ACK is received as shown below in Figure 8 [7]. Congestion is detected when three duplicate ACKS are sent back to the sender. This threshold is indicated by the 'duplicate ACK threshold' parameter seen in Figure 7. TCP maintains a congestion window limiting the total number of unacknowledged packets between communication devices to avoid congestion collapse. Through a mechanism known as 'slow start,' the congestion window is increased after a connection has been made and after a timeout has occurred. For every packet acknowledged, the window increases by 1xMSS bytes. When this window size has increased above a specified threshold or when congestion is detected via duplicate ACKs, a congestion avoidance algorithm, such as Reno or New Reno, begin to affect the window size. By default, the fast recovery algorithm used for congestion is Reno [6].

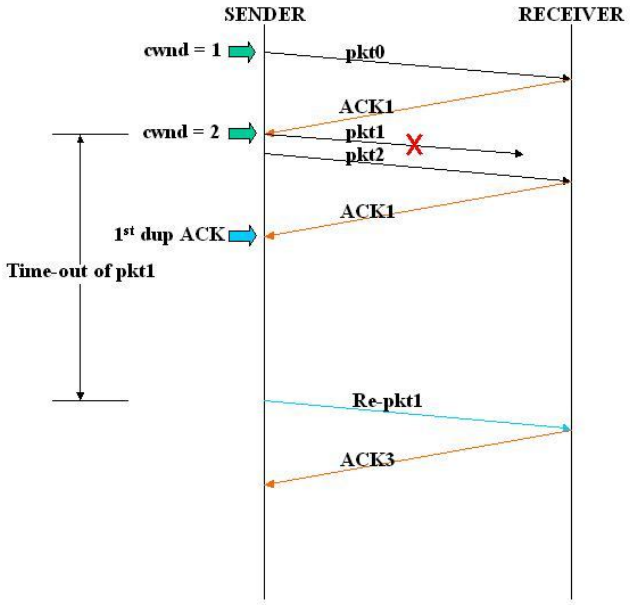

Figure 8: Dropped packets producing duplicate ACKS

When the Reno algorithm is executed, the congestion window is halved after three duplicate ACKS are received and the system performs a 'fast re-transmit' and enters the 'fast recovery' phase in which the dropped packet is re-transmitted. If the ACK for this re-transmitted packet isn't received, a timeout occurs.

Once congestion occurs, the congestion window is reduced by an amount dependant on the algorithm in use. If the Reno fast recovery method is used (as in the baseline scenario), the congestion window is halved. In the simulation of uTP, the TCP parameters were changed to disable fast recovery, so the system initiates a 'fast retransmit' after three duplicate ACKS are detected and the congestion window is reduced to 1 MSS. This is done immediately after three duplicate ACKS are received before waiting for its timeout, hence the name, 'fast re-transmit.'

[8] This change means that the congestion window will spend more time at a smaller value in the uTP scenario, and should make TCP (in the uTP scenario) less aggressive in terms of the amount of data it sends. This should thus model the enhanced co-existence (enhanced QoS for other applications being run on the workstation) that uTP intends to achieve.

2.2.3. Slow-Start Initial Count (MSS)

Originally the TCP parameter, ‘slow start initial count,’ was set to 2 by default. In the uTP scenario, this value was reduced to 1, such that the congestion window would start at an even smaller size. Like the previous change, this results in the congestion window spending more time at a smaller size.

These three parameters are the changes to TCP that we made in order to simulate uTP. Modelling the behaviour of uTP using UDP would have, as explained above, resulted in an inaccurate representation of how the uTP protocol operates. By keeping TCP as the primary reliable service transport protocol in the simulation, the simulation results reflect what is observed in reality more accurately. The changes made to TCP are the modeled effects that the new transport protocol implements and our results do show reasonable changes in performance of other applications when using the simulated uTP protocol.

3. Results

The packet latency over the network is shown in Figure 9. After the network reaches a steady state level of operation it can be seen that the regular TCP (shown in blue) and uTP (shown in red) result in a similar average level of latency. The difference between them is that the uTP is much more prone to bursts and lulls in latency time. It is possible that the smaller TCP packets used the uTP scenario result in this variation; however, It is not readily clear why this is the case and further investigation would be required to determine the full behaviour.

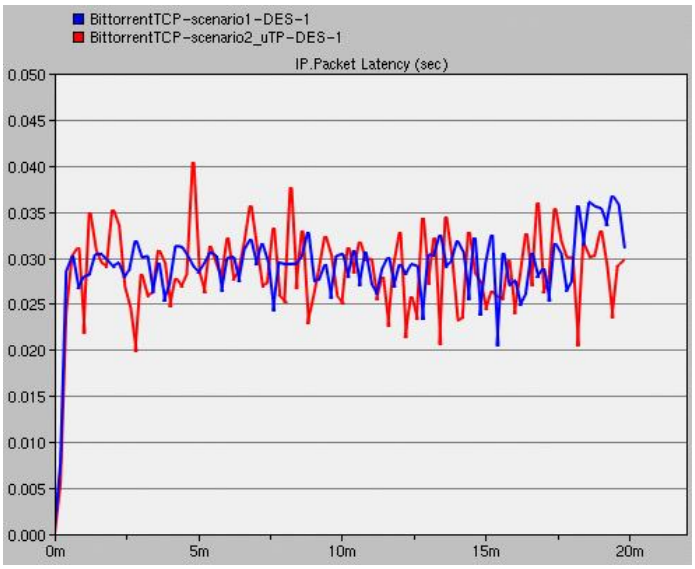

Figure 9: Packet Latency over the IP Cloud.

The overall traffic received from a client is shown in Figure 10. As shown on the graph, less data is transferred through uTP overall, due to several factors. One of the reasons is the changes to the congestion behaviour, which mean that congestion window spends more time at a smaller value in the uTP scenario. However, the packet size will also play a role. The much smaller packets used in the uTP scenario will reduce the throughput of data, as since the headers of fixed size, proportionally more of the space inside the packet is taken up by the IP and TCP headers and thus proportionally more bandwidth is "wasted" on transferring headers.

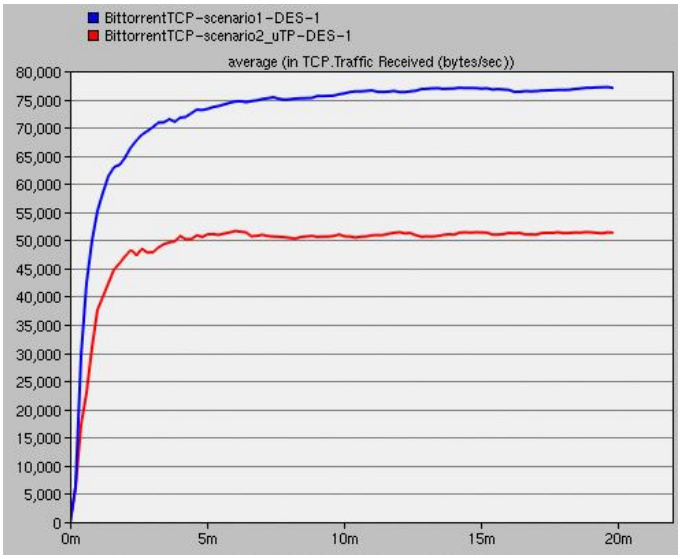

Figure 10: TCP Traffic Throughput

The VoIP application jitter is shown in Figure 11. As can be seen, the more aggressive congestion avoidance present in the uTP scenario means that the connection from the client

and server to the network is not as heavily loaded with FTP data and thus the VOIP packets can be carried through the network at a more even pace.

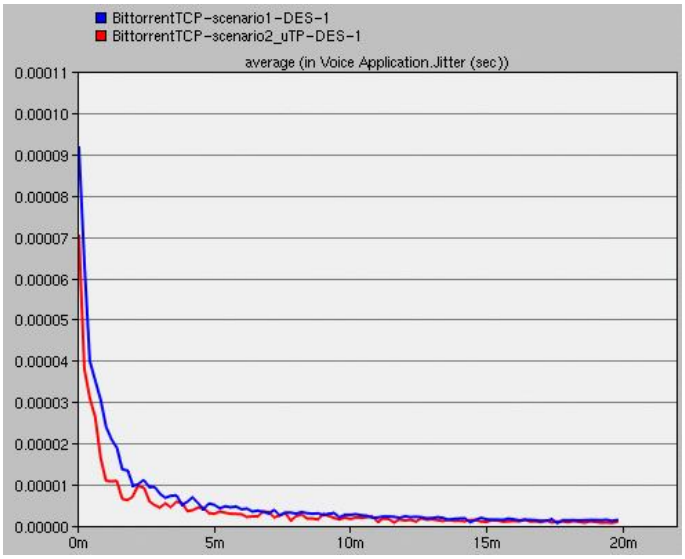

Figure 11: VoIP Application Jitter

The overall end-to-end delay for the VoIP application is shown in Figure 12. As can be seen, the congestion control again reduced the load on the client and server's connection to the network and thus allowed the VoIP packets to be transmitted with much less delay.

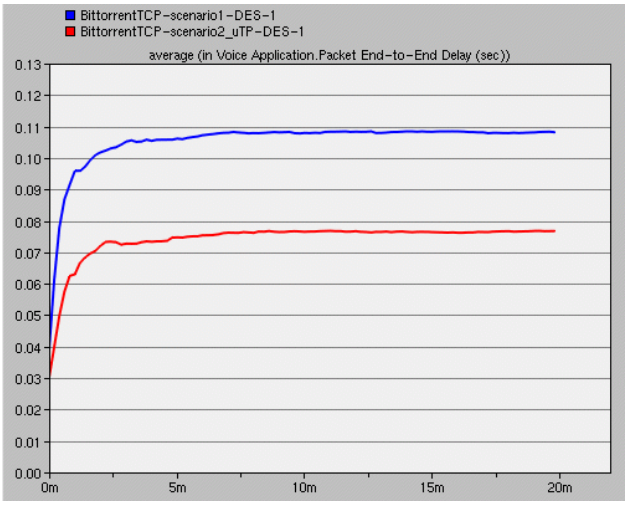

Figure 12: VoIP Application End-to-End Delay

4. Conclusion

This project implements a very limited and simplified version of the uTP protocol, and the accuracy of the simulation of uTP is difficult to evaluate. The network topology is also somewhat simplified as compared to a real-world BitTorrent network. Nevertheless, our simulations show that uTP successfully improved coexistence with other applications. The delay experienced by VoIP packets, as well as the delay jitter, was substantially reduced in the uTP scenario. As expected, this improved QoS did come at a price, as the transfer rate of the BitTorrent data decreased substantially. It is clear that from this simulation, the aggressive traffic control provided by uTP comes at a price but for many network users an improvement in QoS for the other applications running along-side BitTorrent is likely to be worthwhile.

References

[1] B. Cohen, “The BitTorrent Protocol Specification,” Jan. 2008. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html [Accessed: March 7, 2009].

[2] R. Chirgwin, "Torrents of Disruption on the Way?," Dec. 4, 2008. [Online] Available: http://searchnetworking.techtarget.com.au/articles/27957-Torrents-of-Disruption-on-the-Way- [Accessed: Feb. 13, 2009].

[3] "µTorrent 1.9 alpha 14589," Nov. 25, 2008. [Online] Available: http://forum.utorrent.com/viewtopic.php?id=49813&p=1 [Accessed: Feb. 13, 2009].

[4] “INTERNET PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION,” Sept. 1981. [Online] Available: http://tools.ietf.org/html/rfc791 [Accessed: Mar. 30, 2009]

[5] “TCP Maximum Segment Size tuning,” 2008. [Online] Available: http://publib.boulder.ibm.com/infocenter/systems/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/tcp_max_seg_size_tuning.htm [Accessed: Mar.28, 2009].

[6] K. Fall and S. Floyd, “Simulation-based Comparisons of Tahoe, Reno, and SACK TCP,” [Online] Available: www.icir.org/floyd/papers/sacks.pdf [Accessed: Apr.2, 2009].

[7] W. Putthividhya, “About Fast-Retransmit Algorithm,” Mar. 13, 2001. [Online] Available: http://www.isi.edu/nsnam/DIRECTED_RESEARCH/DR_WANIDA/DR/JavisInActionFastRetransmitFrame.html [Accessed: Apr.4, 2009].

[8] W. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms,” Jan. 2009. [Online] Available: http://www.ietf.org/rfc/rfc2001.txt [Accessed: Apr.4, 2009].

[9] M. Fras, S. Klampfer and Ž. Čučej, “Impact of P2P traffic to the IP communication network performances,” [Online] Available: www.sparc.uni-mb.si/OPNET/PDF/ImpactOfP2P.pdf [Accessed: Mar. 20, 2009].