EVALUATING THE PERFORMANCES OF THE CASTGATE TUNNEL SERVER OVER TCP AND UDP LINKS IN MULTI-CLIENT CONFIGURATION

Melinda BARABAS Georgeta L. BOANEA Kris STEENHAUT* Virgil DOBROTA
Technical University of Cluj-Napoca, Communications Department
26-28 George Baritiu Street, 400027 Cluj-Napoca, Romania, Tel: +40-264-401226
E-mails: {Melinda.Barabas, Georgeta.Boanea, Virgil.Dobrota}@com.utcluj.ro
** Vrije Universiteit Brussel, ETRO Department, Pleinlaan 2, B-1050 Brussels, Belgium*
E-mail: Kris.Steenhaut@vub.ac.be

Abstract: The CastGate architecture defines a transitory tunneling solution, offering access to native multicast. This paper describes the implementation of a testing application (CastGate Tester - CT) for the CastGate Tunnel Server (TS), using TCP and UDP links. The goal is to evaluate the performance of the TS in a controlled environment, by emulating a variable number of clients that connect to the server. Two scenarios were defined: load test and stress test. Based on these, the limitations of the CastGate Tunnel Server were identified. The obtained results can be used in the optimization stages of the TS.

Key words: *multicast, CastGate project, UMTP*, Tunnel Server, Tunnel Client, load test, stress test.*

I. INTRODUCTION

Multimedia streaming uses many, individual unicast connections (from a single sender to a single receiver). This method can put a heavy load on both network and server because multiple copies of the same stream must be carried along the full paths of all the connections between the source and each of the destinations. A better solution is to send the information from the content server to a content distribution network, a system of computers that cooperate transparently to deliver content to end-users.

Probably the most efficient way of handling massive amounts of connections without overloading the network is using IP multicast: having the content server send the information in the network only once, and creating copies only when the links to the destinations split. Each multiplication of information item is kept as close as possible to the end-users, leading to a minimal traffic overhead on the network.

Multicast minimizes network and content server load, the server and the Internet connection of the content provider needs to support only one single stream. Multicast reduces the probability of problems regarding capacity, delay and delay variation for isochronous applications (streaming) [1]. IP multicast is implemented in most computers and networking systems, most content distribution end-user applications are "IP multicast ready". Although multicast allows receiving rich media and other content without placing a high burden on the network, in practice it is virtually unavailable on the Internet, being blocked in the access networks of the ISPs [2]. The main reason behind the lack of multicast deployment is an economical reason: a "three-party" deadlock [1]. Content providers do not use multicast, end-users do not ask for multicast access, and network providers do not offer multicast, as there is almost no demand from their customers and hence the initial cost to enable multicast is not justified. The lack of multicast has technical reasons too. Current Internet multicast is weak in: Authentication, Authorization and Accounting. Another problem is reliability: the communication is connectionless and usually unidirectional, so instead of TCP connections UDP is used. Thus packet loss becomes an issue because flow control is not automatically present.

The CastGate architecture uses unicast tunnels between the end-user machines to encapsulate the multicast packets in between the portion where we do have IP multicast (Tunnel Server) and the end-user.

This paper describes the implementation of a testing client for the CastGate Tunnel Server, in multi-client configuration. The second chapter describes the CastGate project and the CastGate tunneling system. The third chapter refers to the implementation of a testing client (CT) for the CastGate tunneling system. Chapter four and five present the defined scenarios: load test and stress test and the evaluation of testing results. Conclusion and future work are presented in the last chapter.

II. THE CASTGATE PROJECT

The CastGate project was started by the "Digital Telecommunications" (TELE) research group of the ETRO department at the Vrije Universiteit Brussel and it is an attempt to provide connectivity for hosts that cannot access multicast network.

The objective of the project is to obtain a breakthrough in the enabling of IP multicast in the public Internet. The key is to provide a simulated access towards end-users without native multicast connectivity, allowing thus

Manuscript received November 4, 2009. Revised November 22, 2009

content providers to distribute audio and video streams [3]. The CastGate project attempts to make the threshold for multicast access towards Internet users as low as possible.

This solution is mainly a transition technology to increase the number of multicast users and to boost Internet multicast traffic. CastGate by itself does not aim to reduce the network load, but it allows a gradual enabling of native IP multicast access by network operators. A detailed description of the project can be found in [1].

A. THE CASTGATE TUNNELING SYSTEM

The basic idea is to have a tunneling system, allowing end-users to connect through an automatically configured tunnel network. This is unicast communications to a server situated in a multicast enabled portion of the Internet, and the whole multicast traffic is transported inside those unicast connections.

The concept is presented in Figure 1. The following devices are included in this architecture:

a) *Tunnel Clients* (TC): host stations or Content Servers without multicast access, acting as end-points of a tunnel, and sending/receiving multimedia content;
b) *Tunnel Servers* (TS): encapsulate/decapsulate the multicast channels selected by a client into/out off a unicast communication, and receive/transmit that traffic from/to the multicast enabled network.

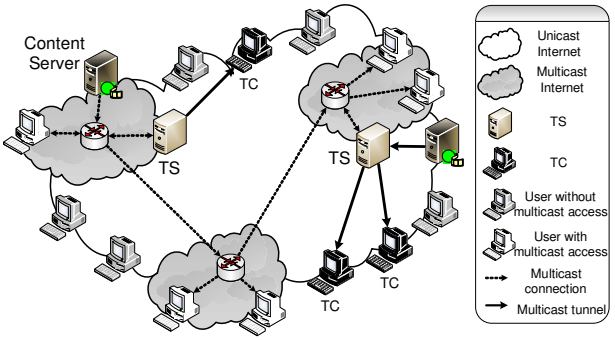

Figure 1. CastGate tunneling concept

The operation of the CastGate tunnel service is based on the availability of CastGate Tunnel Servers connected to the multicast enabled part of the Internet. These allow stations to set up unicast tunnels to them over which the multicast traffic can be forwarded. The CastGate architecture is built around the concept of *application level tunneling* augmented with an automatic tunnel end-point location mechanism [3].

Figure 2 illustrates the basic principle of CastGate multicast tunneling. The Tunnel Server (implemented at software level) bridges the multicast portion of the Internet through the individual tunnels to the clients. At the end-user side, corresponding tunnel client software needs to be installed to terminate the tunnel.

Encapsulating unicast data in multicast traffic is accomplished using the UMTP* protocol. The UMTP* sublayer is a "trimmed down" version of UMTP, the "UDP Multicast Tunnel Protocol" (presented in [4]), enhanced with CastGate specific options (described in [5]). It is used in the unicast tunnel as a data transport and signaling protocol, between the multicast traffic layers and the unicast tunnel connection layers.

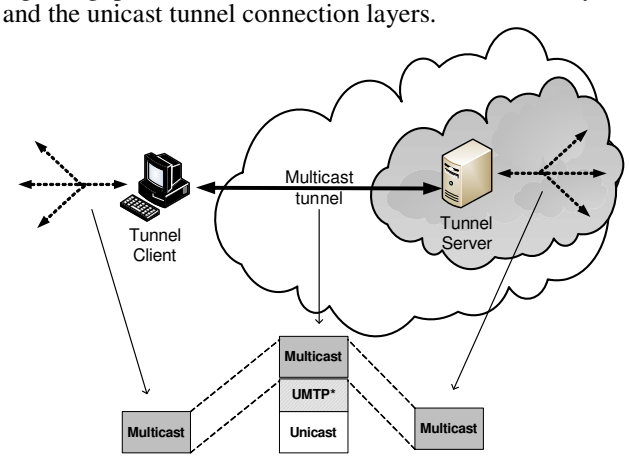

Figure 2. Basic principle of CastGate tunneling

The Tunnel Client sends UMTP* commands to the Tunnel Server, and waits for the proper response. UMTP means tunneling multicast UDP datagram packets inside unicast UDP datagrams. The UDP packets sent by the client contain the 12-octet UMTP tunneling trailer (Figure 3) and negotiation fields (which are optional).

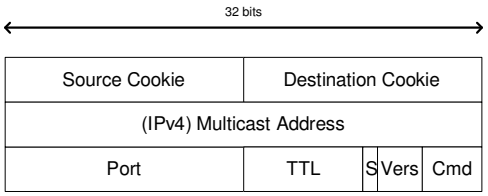

Figure 3. UMTP Trailer [4]

The *multicast address* and *port* identify the multicast channel to join or to relay data to. The command field specifies which command is sent and indicates the exact use of the trailer. *Vers* indicates the protocol version and is currently zero. The *source cookie* and *destination cookie* must be unique for each tunnel (these are used to protect against IP source address spoofing).

The Tunnel Client can send the following commands, as defined in [5]:

- PROBE: to determine the willingness of a Tunnel Server to act as a tunnel end-point. The client also indicates the source cookie that should be used in the rest of the communication. The Tunnel Server's response on a PROBE is either PROBE_ACK or PROBE_NACK.
- JOIN: to indicate to the Tunnel Server which group and port to join. The server will respond with JOIN_ACK or JOIN_NACK messages. The JOIN message should be repeated every 15 seconds. If a CastGate server does not receive a JOIN for a given session within 60 seconds, then the session is no longer tunneled.
- LEAVE: to let the server know that the client is no longer interested in receiving content from that session.

Option and parameter negotiation were introduced to the UMTP protocol in order to support the enhancements of UMTP* and to make it more extendible. These fields have variable length and they are sent before the PROBE, PROBE_ACK, PROBE_NACK, JOIN, JOIN_ACK, JOIN_NACK trailers and are used to exchange setup information between the two end-points of the tunnel [5].

The Tunnel Server sends the DATA command if it is tunneling multicast traffic towards the client. In this case the UMTP* descriptor is preceded by the data that is being tunneled. To realize a more efficient transmission, the Tunnel Server can use a 4-octet trailer (Figure 4), negotiated previously with the client.

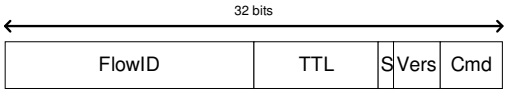

Figure 4. FLOW_DATA message

In this case, each channel (multicast address and port) is identified by a unique 2-octet identifier: FLOW_ID, obtained by the peer.

B. TUNNELING OVER UDP AND HTTP/TCP

There exist two possible choices for the unicast communication of the tunnel, as indicated in [1]: transport over UDP (Figure 5a) and transport over HTTP/TCP (Figure 5b).

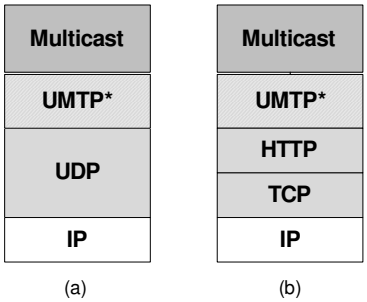

Figure 5. Tunneling over UDP (a) vs. tunneling over HTTP/TCP (b)

Transport over UDP is preferred to TCP for live streaming applications and it is the most natural choice, because native multicast traffic uses it too. However there are two practical issues: fragmentation and firewalls. To solve these two problems, an alternative unicast connection mechanism is provided in the CastGate project, using HTTP/TCP stack. By adding an additional encapsulation level in the form of HTTP, the tunnel will look like being "web" traffic, using TCP destination port 80. This approach deals with the problem of firewall restrictions on traffic.

Each UMTP* packet is either a "command" (instructing the Tunnel Server to join or leave a multicast group address and port) or "data" (an enclosed multicast UDP datagram payload), depending on the signaling used in the UMTP* sublayer. Figure 6 illustrates the UMTP* message sequence chart of a typical client-server interaction.

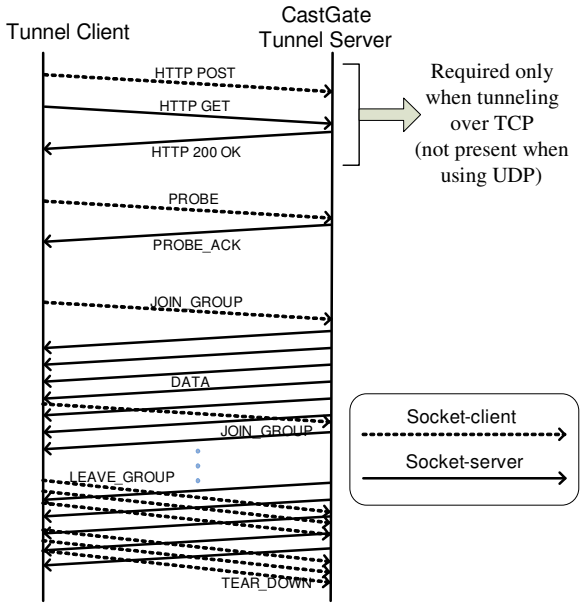

Figure 6. UMTP message sequence*

The CastGate Tester implements this exchange of UMTP* messages for each simulated Tunnel Client. If we test the server over TCP connections, the UMTP* message exchanges are preceded by HTTP POST and GET interactions.

III. TUNNEL SERVER PERFORMANCE EVALUATION

The evaluation of the Tunnel Server is performed by emulating a large number of clients that will be served by a single Tunnel Server in a controlled environment. The CastGate server is public TS software provided by *BELNET*, the academic and research network in Belgium. The CastGate Tester runs on a computer that is in the same local area network (1 Gbps LAN) as the Tunnel Server.

An evaluation of the performance and capacity of the CastGate TS is needed because of the unpredictable behavior of these servers in real usage conditions. When the CastGate system was tested the first time, because of the large number of viewers, the system became unstable and restarted every 5-6 minutes. Not knowing how many clients the Tunnel Server can handle and how it reacts to the clients' behavior, you cannot guarantee the proper quality of the multimedia stream.

The Tunnel Servers will be used in public and commercial audio and video streaming (e.g. radio, television quality video), and it is vital to know how many tunnels can be managed by a single server, in which conditions and for what type of media, in order to be able to satisfy the clients' expectations. Based on these observations, we have to identify techniques to improve the behavior of the CastGate Tunnel Server. Before deploying large scale content distribution architectures based on CastGate, all bugs and imperfections of the software implementation have to be identified and eliminated.

The CastGate Tester is a software application written in C# using the Microsoft.NET technology, Framework 3.5. The CastGate Tester emulates a large number of

clients simultaneously in order to evaluate the performance and capacity of the CastGate multicast system, i.e. detecting missing packets by monitoring the sequence numbers in the RTP header, respectively detecting the failure to create new tunnels.

For each client an application level tunnel must be built. Each emulated client uses two sockets to establish a connection with the Tunnel Server (see Figure 6). The first socket is responsible for sending the following commands: PROBE, JOIN_GROUP, LEAVE_GROUP, TEAR_DOWN, and HTTP POST (only in the case of TCP connections). Using the second socket, the Tunnel Server sends the PROBE_ACK command and the DATA messages. This socket is also responsible for the exchange of HTTP GET and HTTP 200 OK messages.

Each tunnel is identified by the port numbers in the outer UDP header (when tunneling over UDP) or in the TCP header (when tunneling over TCP) of the involved sockets. In this way, the different clients emulated with the CastGate Tester (having the same IP address) must be bound to different ports on the local machine in order to create distinct tunnels.

Two major testing approaches were identified: load testing and stress testing. In this paper we describe both testing scenarios, in order to determine how the CastGate server acts in heavy loading scenarios. In these scenarios we assume that the Tunnel Server is behaving incorrectly if a client is unable to connect to the server or if packets with multicast content are missing.

Figure 7 presents the testing scenario for both testing approaches.

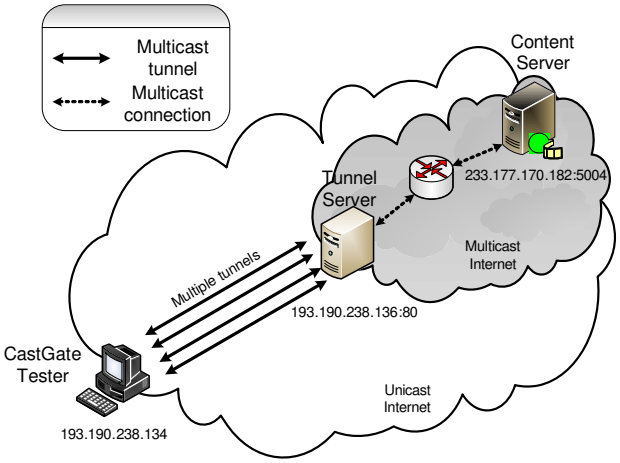

Figure 7. Testing scenario

IV. LOAD TESTING

The goal of the load test is to estimate the number of Tunnel Clients that can connect consecutively to a Tunnel Server, i.e. to determine how many simultaneous connections can be handled by a single Tunnel Server without having a significant loss of multicast data packets. With this information a service provider can set up a functional, stable CastGate tunneling system by limiting the number of clients connected to a certain server, and redirecting or not accepting other clients.

Emulating the operation of multiple Tunnel Clients relies on implementing a special client. This asks for the creation of a large number of tunnels, setting up one tunnel at a time and incrementing the number until the server does not respond anymore, or a given number of errors start to appear in the tunnels. The final result of a load test represents the number of tunnels for which the CastGate server functioned properly.

The multicast stream is described by RTP (Real-Time Transport Protocol) headers, which also contain a sequence number indicating the frame being transmitted. Comparing the sequence numbers of the incoming packets, we can detect missing packets. The absence of more consecutive packets in a certain stream indicates that the Tunnel Server is overloaded and cannot manage to send all the frames to each client. Another indicator of the incorrect behavior of the Tunnel Server is the failure to send PROBE_ACK or PROBE_NACK messages to the newly connecting clients.

In the load testing scenario, tunnels are created one after the other, using multi-threading, and the data packets sent by the server are monitored. Setting up tunnels emulates the behavior of real clients because it involves all the messages a real Tunnel Client would exchange with the tunneling server. But this implementation of multiple clients, accomplished by the CastGate Tester, neglects the actual multimedia information. The multimedia content does not get any attention; it is not displayed, but discarded after verifying the RTP sequence number.

Two important parameters of load testing were identified:

- time delay between setting up two consecutive tunnels, in seconds *(d)*;
- type of multimedia traffic being tunneled (audio or video).

We carried out different load tests, changing the delay interval *d* and the type of traffic being tunneled, for both TCP and UDP links. A detailed presentation of the results of these tests can be found in [6] and [7].

If we define a delay of *d=1s* between setting of two consecutive tunnels over TCP connection and if we select video traffic, we can observe the evolution of the TCP segments received/sent by the CastGate Tester, illustrated in Figure 8.

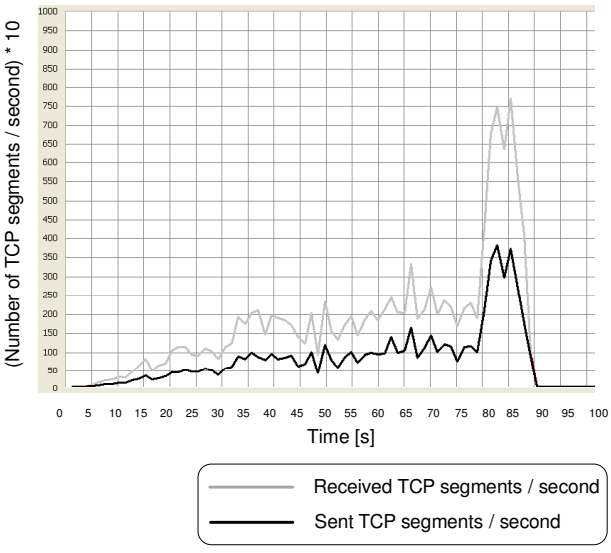

Figure 8. Load testing over TCP connections

Under these conditions up to *70* clients emulated by the CastGate Tester could connect to a single Tunnel Server. The test was stopped because some Tunnel Clients did not receive data over the tunnel, as seen in Figure 9.

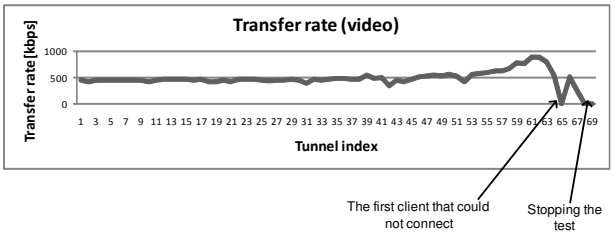

Figure 9. Load testing over TCP – transfer rate

If we keep the previous settings (*d=1s* and video traffic) and create UDP tunnels, we can observe that up to *155* clients emulated by the CastGate Tester could connect successfully to a CastGate Server (See Figure 10).

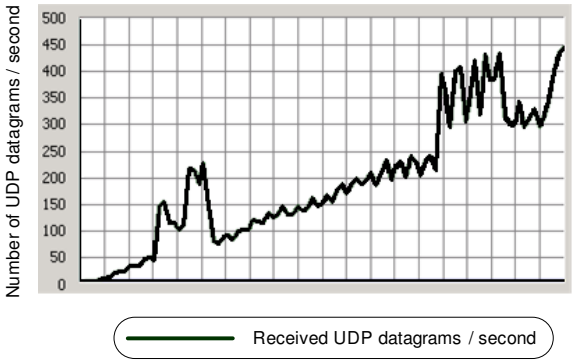

Figure 10. Load testing over UDP links

Repeating the tests several times, we observed that we cannot have more than *110* TCP tunnels connected simultaneously to a single Tunnel Server, while the number of UDP tunnels can reach even *400*; i.e. if we create only UDP tunnels, the Tunnel Server is able to handle a greater number of clients.

Examining the effect of the delay between creating two consecutive tunnels, we observed that the server is behaving correctly for a longer period of time if the value of *d* increases. We concluded that it is more difficult to manage clients if they connect to the CastGate Tunnel Server shortly one after another or almost simultaneously. For example, in the case of a TCP connection and transmitting audio traffic, if *d=3s*: the server behaves correctly for 115 seconds. If *d=0.1s*: the first denied tunnel appears after 17 seconds.

Performing different tests, we came to the conclusion that the number of clients which can be served by a single server depends highly on the type of traffic: audio or video. Table 1 presents the number of connected tunnels, if modifying the delay between starting two consecutive tunnels *(d)*. An audio stream requiring less bandwidth, therefore more simultaneously transmitting tunnels can be created. This observation is valid for both UDP and TCP links.

	Delay between setting up two consecutive tunnels			
	d = 0.1 s	***d = 0.5 s***	***d = 1 s***	***d = 3 s***
Audio stream	109 tunnels	90 tunnels	50 tunnels	30 tunnels
Video stream	85 tunnels	73 tunnels	69 tunnnels	35 tunnels

Table 1. Tunnels connected to the TS over TCP

Another observation is that more RTP packets are missing if the received traffic is a video stream than in the case of audio streams.

V. STRESS TESTING

In the stress testing scenario, the CastGate Tester is randomly setting up and dropping tunnels, the connection-disconnection rate (dynamic load) depending on the specified connection time. By connection time we mean the period of time for which the client stays connected to the server.

Using this kind of scenario we can test the dynamic behavior of the Tunnel Server, pinpointing situations which lead to an unpredictable behavior of the server. This knowledge could help improve the performance of the CastGate system by avoiding critical situations or managing them in a different way.

Two important parameters of stress testing were identified:

- the average number of clients connected to the Tunnel Server at a time *(M)*;
- the average connection time in seconds *(D)*.

If we set *M=75* and *D=1000s*, we obtain the following transfer rates at the CastGate Tester side (Figure 11):

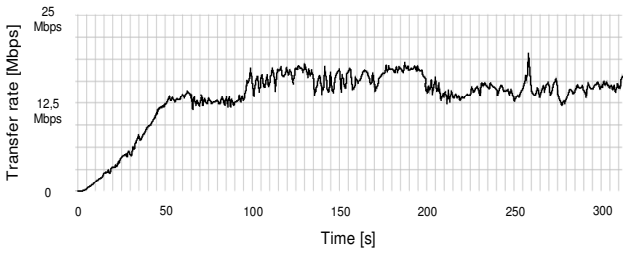

Figure 11. Stress testing over TCP – stable TS

But if we set a much lower connection time for the individual TCP tunnels *(D = 10s)*, even when the number of served clients is reduced *(M = 25)*, the Tunnel Server becomes unstable, as you can see in Figure 12.

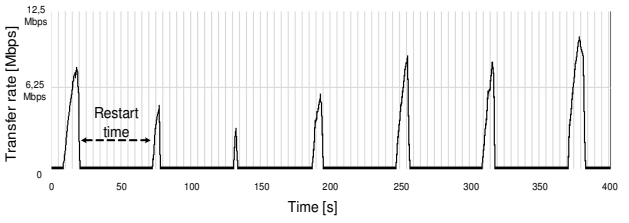

Figure 12. Stress testing over TCP – unstable TS

Performing several stress tests with different parameters, we can observe that the Tunnel Server can manage approximately 70-80 Tunnel Clients when streaming audio content, and 50-55 Tunnel Clients when streaming video content. The server becomes unstable if more than 10-15 clients disconnect simultaneously, i.e. they induce instability.

This event manifests in periodic restarts, which lasts 50 seconds and occurs once every 2-3 minutes. Without any warning, a Tunnel Server can shut down and does not respond for several minutes. Whether using TCP or UDP links to establish tunnels, the server presents the same unstable behavior.

To evaluate the combined effect of TCP and UDP tunnels, we performed stress testing under concurrent traffic conditions. The defined scenario for this type of stress testing is illustrated in Figure 13. If we run two separate instances of the CastGate Tester, one can fulfill the role of a TCP Tester and the other the role of a UDP Tester.

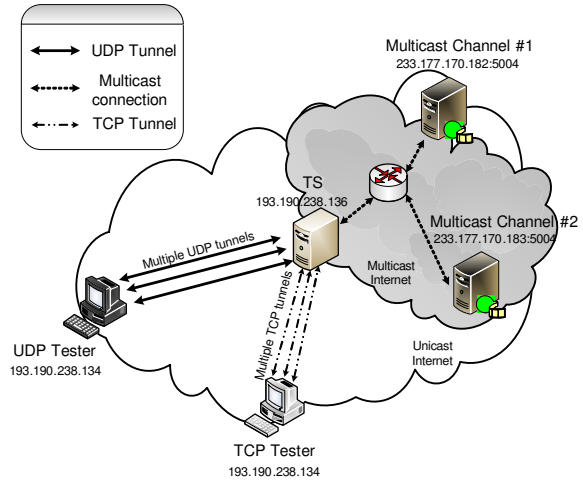

Figure 13. Stress testing under concurrent traffic conditions

If we define *M=10* UDP tunnels and *M=10* TCP tunnels and *D* is set to *50* seconds for a video streaming, the Tunnel Server will restart periodically after *60-70* seconds, as presented in Figure 14.

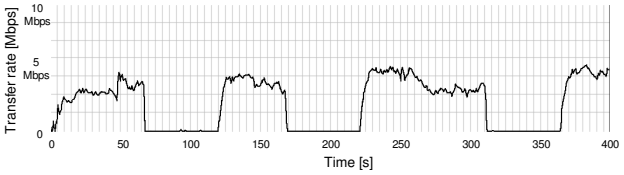

Figure 14. Stress testing under concurrent traffic conditions

In a similar situation, but creating *M=20* tunnels of the same type (either TCP or UDP links), we can observe that the server remains stable for a longer time. Varying the number of UDP and TCP connections and analyzing the results, we can conclude that TCP tunnels have a greater impact over the Tunnel Server's behavior. Thus they are the major cause of instability.

VI. CONCLUSIONS

This paper describes the implementation of a testing-client for the CastGate Tunnel Server. The CastGate Tester offers the possibility of evaluating the performance of the TS in a controlled environment, thus the connection between the behavior of the TC and the operating mode of the TS can be determined. Multithreading mechanisms were used to simultaneously emulate the behavior of a large number of Tunnel Clients, communicating with the TS through HTTP/TCP or UDP tunnels. Multicast information is sent through the tunnel in form of RTP packets encapsulated in UMTP* packets.

Comparing the results obtained for tunneling over TCP and over UDP, important differences appear in the behavior of the CastGate Tunnel Server when we change the type of the connection. We observed that the number of TCP tunnels which can be served simultaneously without errors is much lower than in the case of UDP tunnels.

Additionally, the Tunnel Server is able to handle a greater number of clients when streaming audio content.

The stability of the tunneling system proved to be dependent on the number of clients that end the connection to the server almost at once. The frequency of restarts increases with the number of simultaneously disconnecting Tunnel Clients.

Possible improvements applicable to the CastGate system, identified with the help of the CastGate Tester, are: *a)* separating the UDP and TCP functions of the Tunnel Servers by providing distinct servers for UDP and for TCP tunnel, because this approach could ensure a greater stability of the whole system; *b)* limiting the number of clients which can connect to a single tunneling server using TCP connections (~50); *c)* limiting the number of clients which can connect to a single Tunnel Server using UDP links (~100); and *d)* preventing the simultaneous disconnection of more the 10-15 clients.

ACKNOWLEDGMENTS

We acknowledge the support obtained from the TELE research group of the ETRO department, during our stage at the Vrije Universiteit Brussel within the Erasmus Program.

REFERENCES

[1] P. Liefooghe, M. Goossens, A. Swinnen, B. Haagdorens, "*The VUB Internet multicast "CastGate" project*", Technical Report 10/2004 v1.8, 2004.
[2] M. Goossens, P. Liefooghe, A. Swinnen, "*The CastGate project - Enabling Internet multicast for content distribution*", 2006.
[3] P. Liefooghe, "*An Architecture for Seamless Access to IP Multicast Content*", 2002.
[4] R. Finlayson, "*The UDP Multicast Tunneling Protocol <draft-finlayson-umtp-09.txt>*", Internet-Draft, 2003.
[5] P. Liefooghe, "*CastGate: an auto-tunneling architecture for IP Multicast <draft-liefooghe-castgate-02.txt>*", Internet-Draft, 2004.
[6] M. Barabas, "*Testing the CastGate Tunnel Server over TCP Connections in Multi-client Configuration – Diploma Thesis*", Technical University of Cluj-Napoca, Romania, 18 June 2008.
[7] G. Boanea, "*UDP Multi-client Tester for CastGate Tunnel Server – Diploma Thesis*", Technical University of Cluj-Napoca, Romania, 18 June 2008.