Comparing TCP and QUIC through a satellite link

by Samy Bettaieb and Jonathan de Salle

1) Introduction

The objective of this project is to compare the performance of TCP [1] and one QUIC [2] implementation with real servers on the Internet through a satellite link. The measurements should be scripted and automatically analyzed to be easily performed and reproduced.

We decided to restrict ourselves to measuring the performance of both protocols in the case of a download from the satellite link, and will not measure the performances on the upload.

You can find the different files created for the project at the link https://github.com/jdesalle/linfo2142_project .

[1] https://datatracker.ietf.org/doc/html/rfc793

[2] https://datatracker.ietf.org/doc//html/rfc9000/

2) Tools

To perform the measurements, we use several tools, which will be described in this section .

We have chosen to use cloudflare's implementation of QUIC: *Quiche* [1] . We opted for that implementation, since it is compatible with a version of *cURL* [2], which we use to perform our measurements.

To have a better control over our measurements, we implemented a web server using *Nginx 1.16*. The choice of the specific 1.16 version is because a patch for this version exists to use it with Quiche, which is the QUIC implementation we are using through this project. While Nginx has its own implementation of QUIC, we chose to keep a same implementation for the whole project.

We use *Wireshark* [3] and especially its CLI interface *tshark* to capture packets and analyze their different characteristics. We will also use *QUIClog* [3] to analyze these packets.

[1] https://github.com/cloudflare/quiche

[2] https://github.com/curl/curl/blob/master/docs/HTTP3.md#quiche-version

[3] https://www.wireshark.org/

[4] https://github.com/quiclog

3) Methodology

We will use a computer connected through a satellite link (starlink) to our web server to perform our measurement. During each measurement process, Wireshark will be monitoring the traffic to allow us to observe the negotiation of the bandwidth and other interesting information about the protocol directly.

During each measurement, we will check the time needed for different things:

- time for the namelookup
- time to connect
- time to start the transfer
- total time

We will study only the downloads, since it's more relevant than uploads for a satellite connection users.

We will repeat each measurement 10 times. Furthermore, we will first try to download a simple blank web page, then we will try to download files with a size ranging from 1 MB to 500 MB, to see if the size of a file

downloaded through the satellite link will impact the performance of the said connection. For empty files, we repeated measurements 100 times, after noticing some random behavior of the download over QUIC.

4) Configuration of the server

The first step of our configuration is to add your serve in the /etc/host file, and use the domain name linfo2142.serv for our scripts to work. In our case, the IP of our server is 130.104.229.21, the command to do this is *$ sudo echo "130.104.229.21 linfo2142" >> /etc/hosts*

The next step is to proceed to the installation of our version of nginx. The basic explanation for building a nginx 1.16 server with Quiche can be found online [1]. You may have to install some dependencies using your preferred package manager.

After building nginx, you will have to add it to your path variable. There are multiple ways to do this, we chose to add it in /etc/environment. You will have to generate files for the measurement through our script [2] in /var/www/files folder, so that they will be accessible by the server. This repository must be accessible by any user.

To run nginx, you will have to add our nginx.conf [3] file in your nginx repository. You'll also need to put the SSL key [4] and certificate [5] of our server in a folder named "certs" created in the same directory. When that is done, you can run the command *$ nginx -c nginx.conf* to launch the server.

[1] https://blog.cloudflare.com/experiment-with-http-3-using-nginx-and-quiche/

[2] https://github.com/jdesalle/linfo2142_project/blob/main/files/generate.sh

[3] https://github.com/jdesalle/linfo2142_project/blob/main/nginx.conf

[4] https://github.com/jdesalle/linfo2142_project/blob/main/certs/server_cert/linfo2142_serv.key

[5] https://github.com/jdesalle/linfo2142_project/blob/main/certs/server_cert/linfo2142_serv.crt

5) Issues

5.1) QUIClog

To analyze QUIC further, we tried to look at QUIClog. Since we are using an NGINX server and cURL to download files, we thought that the most straightforward way to do it is to use pcap2qlog (https://github.com/quiclog/pcap2qlog).

When using pcap2qlog, the pcap is parsed and a json file is created. That json file si then parsed and a .qlog file is created. However, we had a bug in the second phase of the process : the .json is created correctly and the packets are decrypted (using the TLS session keys) but when creating the qlog file we had and error "Error: convertPacketHeader: unknown QUIC packet type found! : [object Object]".

After a discussion with François Michel and Maxime Piraux, it looks as if it's a bug in pcap2qlog itself.

This issue has been fixed 30 mins before this deadline (thanks to a patch from François Michel). So we will use pcap2qlog for the final deadline.

5.2) Download files over http3

We tried to check the influence of the size of a file on the speed and packet loss for each protocol. For this we tried to create files of different sizes (empty, 1 MB, 100 MB, 250 MB, 500 MB) to download. Unfortunately, curl seems to have a problem with big files in http3 [1], which didn't allow making the measurement in QUIC for this iteration of the report.

When trying to download large files (or even non-empty files), the current speed drops to 0 and it looks like the download does not terminate even after 3 hours (see image below). It is a random bug, sometimes it works, but most of the time it does not ¯_(■)_/¯.

```
sambt@von:~/Desktop
$ ~/curl/src/curl --http3 -X GET -k https://130.104.229.21/files/file_1MB -o test
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
 51 1024k   51  529k    0     0     44      0  6:37:11  3:24:02  3:13:09     0
```

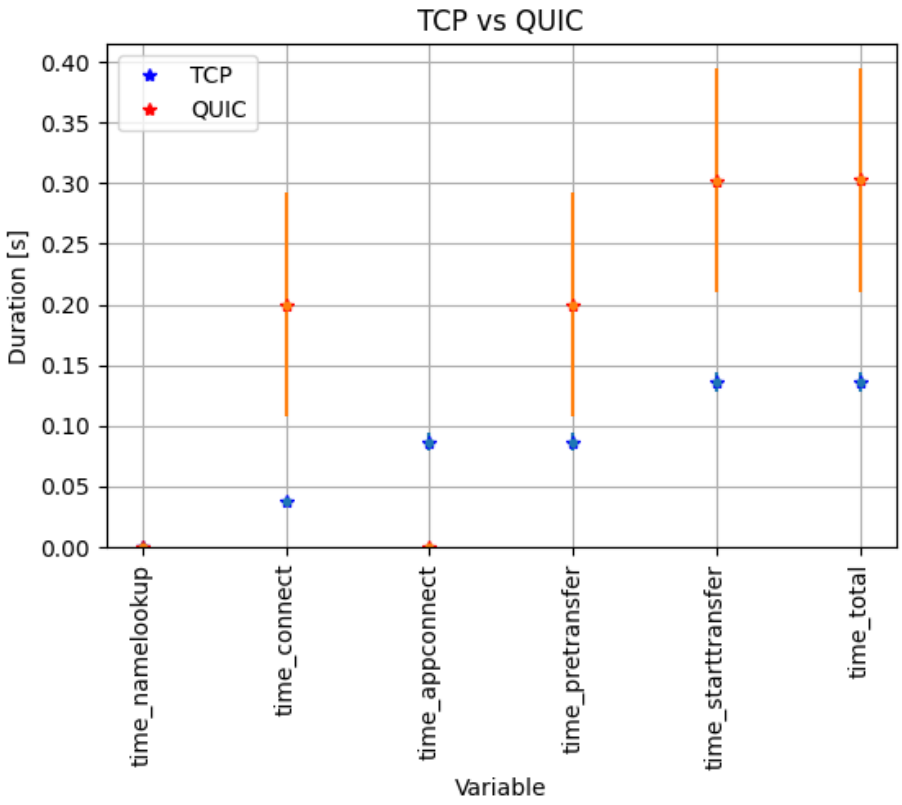
It seems to be a known bug, and since the majority of issues on this topic on github are closed, we did not notice the problem early enough to change our methodology. We will try to find another way to download files in http3 for the final deadline.

[1] https://curl.se/docs/knownbugs.html#HTTP3

6) Results

6.1) Time measurements

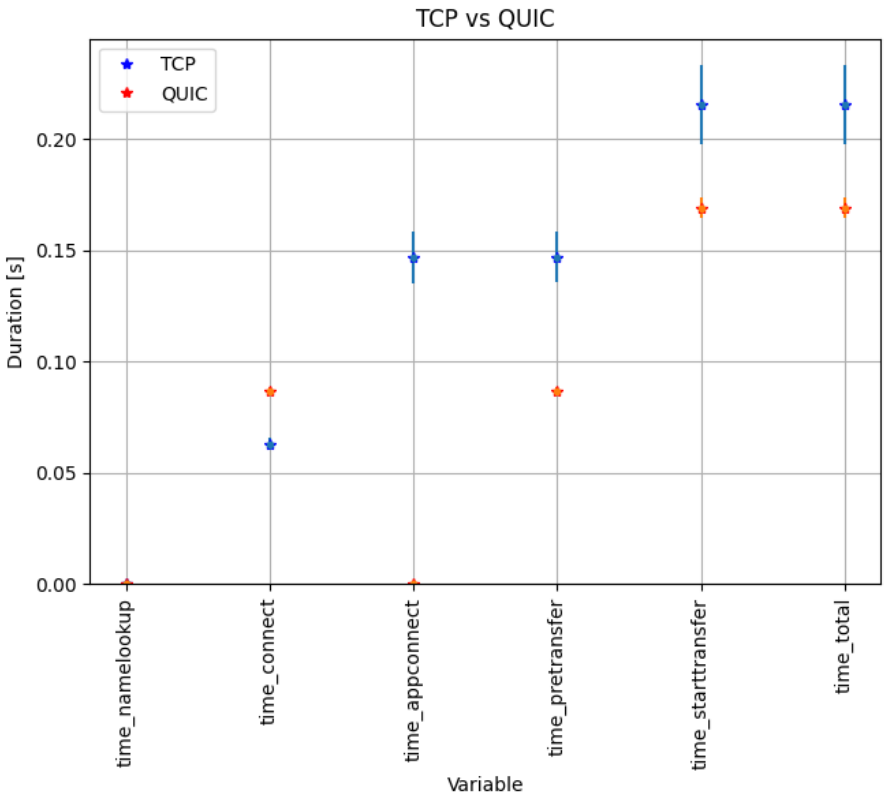
First, we compared some basic metrics while downloading a blank page



We can see that the average time for downloading the empty file is 0.13652440000000002 s, which corresponds to the time to get an empty packet from the satellite, which will be the minimum delay of our operations.

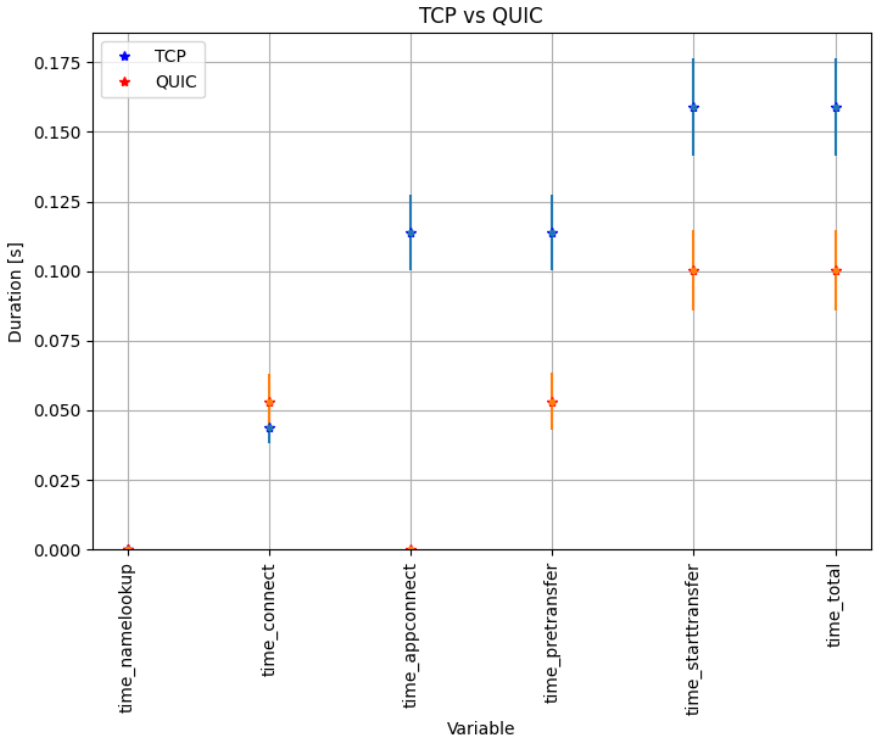
The first thing we notice is the high standard deviation in QUIC compared to TCP, the performance of TCP is more stable/less random than QUIC's.

Also, in another test, QUIC seemed to faster than TCP.



This random behavior of QUIC is probably due to the download issue over http3.

Since downloads for empty files seemed to terminate anyway, we decided to make 100 measurements. With 100 iterations, we seemed to have more stable results :



We see that over all QUIC is faster and spend less time to connect : This is probably to QUIC's handshake which uses the mode 0-RTT [1], where the transport and cryptographic handshake can be sent in a single operation along with the http3 requests in the first connection and makes a 0 Round-Trip Time possible.

[1] https://blog.cloudflare.com/even-faster-connection-establishment-with-quic-0-rtt-resumption/

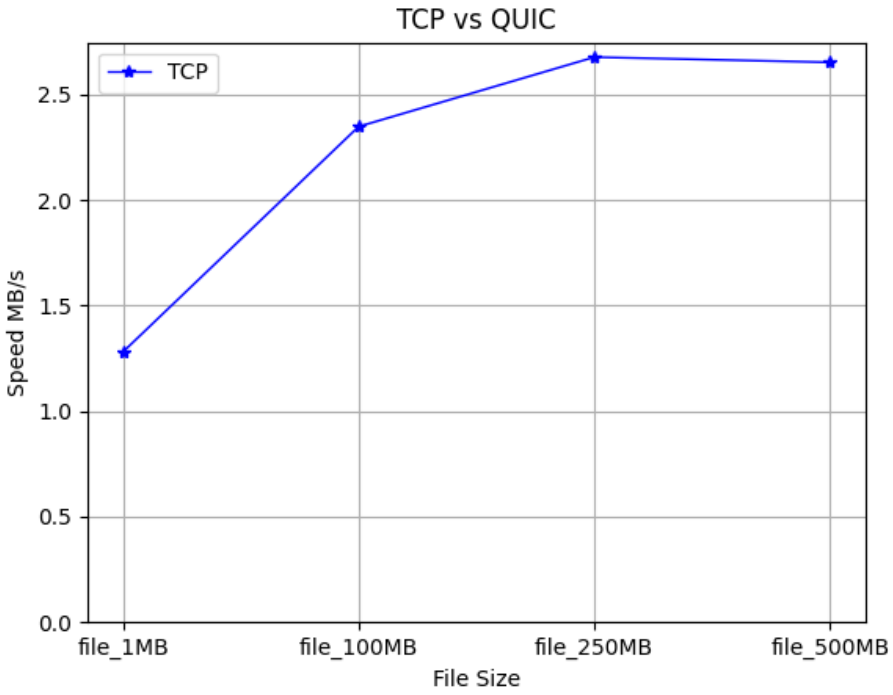
6.2) Congestion control algorithms

TCP and QUIC can use the same congestion control algorithms. In our case, both the server and our client machine use cubic by default for TCP, it should be the method used. The Quiche implementation of QUIC can use both cubic or Hystart++ [1]. In our cases, both TCP and QUIC should use the same congestion control algorithm: cubic.

[1] https://blog.cloudflare.com/cubic-and-hystart-support-in-quiche/

6.3) Influence of file size (in TCP, see issues)

For TCP we could measure the influence of the file size (unlike QUIC), the speed is dropping when the file size is increasing. The drop in speed seem to follow the concave growth of a cubic function, which is consistent with the use of the cubic congestion control algorithm



7) Conclusions

For this study, we configured a file server compatible with both QUIC and TCP, on port 443, using NGINX 1.16. We installed a development branch of curl, allowing http3 to be able to get our measurements on a client computer, connected to a Starlink connection. Those configurations allowed us to gather some data's in both protocols, which allowed us a basic comparison between them, through a satellite connection.