

Alfred Tso Chun Fai 20012065g

COMP5311 Project: Performance Analysis of TCP and UDP

Contents

Introduction	3
What is QUIC	3
TCP v UDP: A case study	4
Background: Related Works	4
Problem definition	5
Different implementation	5
Network Topology	5
The Bandwidth of the links involved	5
Hardware	5
Application by which these packets are sent	5
Objectives	5
Throughput	6
Average Delay	6
Average Jitter	6
Design/Methodology	8
Simulation or Real Lab	8
Tool: NS-3	8
ns-3: basics	8
Result	11
Throughput	11
Average Delay	11
TCP	13
Average Jitter	13
TCP jitter larger than UDP's	13
Discussion	14
Why not compare packets lost?	14
So what might be the reason by which QUIC chose UDP based on our study here ?	14

Code	14
Reference	15

Introduction

Even wonder why Chrome browser feels so much faster than Firefox when watching Youtube videos ? I do and as a curious student i fired up Wireshark and capture some traffic and what i found is this.

9586	12:08:02.449306	142.250.66.110	10.0.0.23	QUIC
9587	12:08:02.449638	142.250.66.110	10.0.0.23	QUIC
9588	12:08:02.450505	142.250.66.110	10.0.0.23	QUIC
9593	12:08:02.452969	142.250.66.110	10.0.0.23	QUIC
9594	12:08:02.452971	142.250.66.110	10.0.0.23	QUIC
9595	12:08:02.453892	142.250.66.110	10.0.0.23	QUIC
9596	12:08:02.453894	142.250.66.110	10.0.0.23	QUIC
9597	12:08:02.453896	142.250.66.110	10.0.0.23	QUIC
9598	12:08:02.453897	142.250.66.110	10.0.0.23	QUIC
9599	12:08:02.454050	142.250.66.110	10.0.0.23	QUIC
9600	12:08:02.454052	142.250.66.110	10.0.0.23	QUIC
9601	12:08:02.454053	142.250.66.110	10.0.0.23	QUIC

According to wiki ¹, QUIC is used by more than half of all connections from the Chrome web browser to Google's servers.

What is QUIC

QUIC is a protocol aiming to improve web application that are currently using TCP ² by establishing a number of multiplexed UDP and some even called it "TCP/2". The upcoming HTTP/3 (Internet-Draft as of Feb 2021) is purported to be using QUIC too.

Wait ... We are using UDP to replace TCP? We were told that UDP is unreliable, how come it can aim to "obsolesce" TCP. Comparison of the two protocol (TCP and UDP) is similar to comparing Train to Airplane rather than Boeing to Airbus (Both *transport*). We, however, can still compare them with some key metrics to catch a glimpse as to what this QUIC chose UDP to improve on.

¹QUIC

²QUIC GoogleDoc

TCP v UDP: A case study

Background: Related Works

- Yuan et al. (2019) ³ compared the performance of TCP and UDP over SDN using average packet delay and packet loss probability. The study found that the TCP outperform UDP over the two metrics by 12-50% and 25-100% respectively. Note that here, we are not conducting our study in SDN.
- He et al. (2004) ⁴ studied the effect of, among other things, packet aggregation and ingress buffering on the throughput and delay jitter of TCP and UDP.
- Gamess et al. (2008) ⁵ proposed an upper bound model for TCP and UDP throughput in IPv4 and IPv6. The paper presented the model for calculating the maximum theoretical throughput of both protocol to be the ratio of the number of TCP or UDP payload bytes to transmit over the min. number of bytes necessary including IFG, Preamble, SFD, Ethernet DIX encapsulation, IP Header, TCP (or UDP) Header and Paddings, multiple by the bandwidth of the link, assuming full-duplex and no processing time.
- Shahrudin et al. (2016) ⁶ compared various transport protocol including TCP and UDP over 4G network using four metrics: Throughput, Packet Loss, End to End Delay and Average Jitter.

³Yuan-Cheng Lai, Ahsan Ali, Md. Shohrab Hossain, Ying-Dar Lin, Performance modeling and analysis of TCP and UDP flows over software defined networks, Journal of Network and Computer Applications, Volume 130, 2019, Pages 76-88, ISSN 1084-8045

⁴Jingyi He, S.-H. Gary Chan, TCP and UDP performance for Internet over optical packet-switched networks, Computer Networks, Volume 45, Issue 4, 2004, Pages 505-521, ISSN 1389-1286

⁵Eric Gamess, Rina Surós, An upper bound model for TCP and UDP throughput in IPv4 and IPv6, Journal of Network and Computer Applications, Volume 31, Issue 4, 2008, Pages 585-602, ISSN 1084-8045

⁶Shahrudin Awang Nor, Raaid Alubady, Wisam Abduladeem Kamil, Simulated performance of TCP, SCTP, DCCP and UDP protocols over 4G network, Procedia Computer Science, Volume 111, 2017, Pages 2-7, ISSN 1877-0509

Problem definition

Comparing transport layer protocol is no easy task, as there are many factors that could affect the result:

Different implementation

Some parameters within protocol (TCP for example) are left up to the implementation (which give rise to TCP/IP Fingerprinting, allowing malicious actor to identify, among other things, OS of the host) meaning that these difference could have subtle effect on the measurement of performance

Network Topology

The immediate route taken by the packets certainly would affect the result so the immediate routers, bottleneck links in between and other traffic could also affect the results.

The Bandwidth of the links involved

As mentioned above, we often might not be sure about the bandwidth of those immediate links or simply the link between client and the server. One of the paper mentioned made an assumption that the link would not have any collisions.

Hardware

We have seen that the buffer could have impacted the performance of these protocols and not only the software side of it is relevant, the hardware part also matters and often we could not quite control these lower level details.

Application by which these packets are sent

In most of the previous works, authors often implement a custom programme to conduct the experiment, we also need to consider the effect of how these software implementations could affect the result, say the interval by which UDP packets are sent, and are there limitation in the language being used as to, for our present purpose, the sending intervals.

Objectives

Given all these considerations, we will use a simple network topology with settings as close to the reality as possible. Where the underlying details such as buffer queue size or protocols, we will use the same for both protocol. As such, we will be comparing TCP and UDP over IPv4 in a wired one-to-one hosts using throughput, average delay and average jitter as our performance indicators.

We will send 10, 100 and `range(1e4, 1e5, 1e4)` packets, each of 1472 Bytes in UDP application and the equivalent number of Bytes in a TCP application to observe the flows of both TCP and UDP applications.

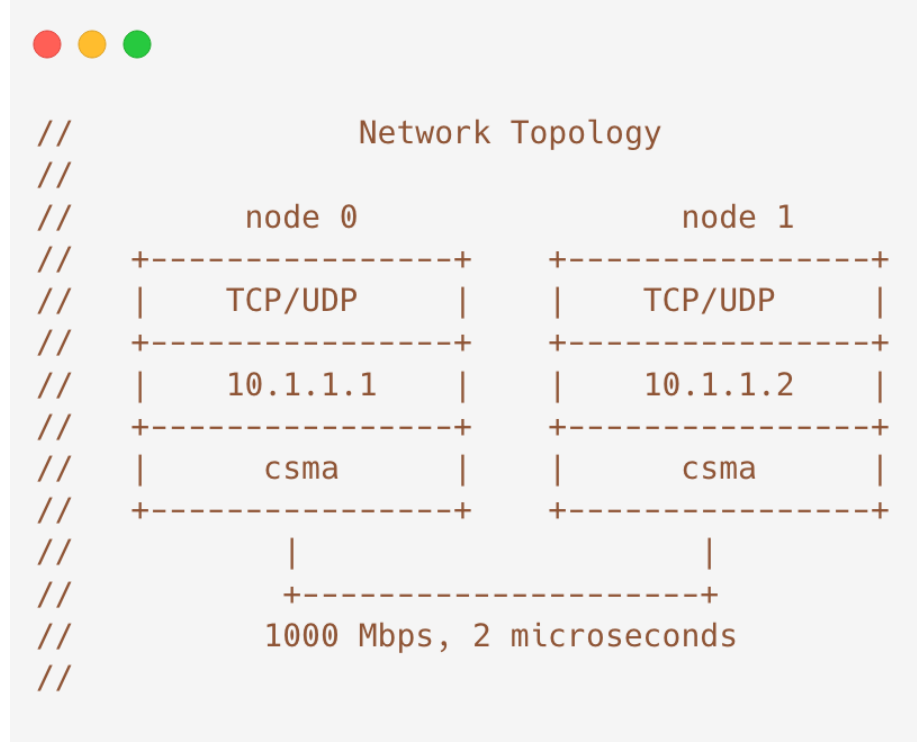


Figure 1: Network topology of our study

Throughput

$$Throughput = \frac{NumberOfReceivedPackets(RxPackets)}{TimeOfLastreceived - TimeOfFirstreceived}$$

Average Delay

$$AverageDelay = \frac{Totalend - to - endDelaysForRxPackets}{RxPackets}$$

Average Jitter

`JitterSum` contains the sum of all end-to-end delay jitter (delay variation) values for all received packets of the flow. Here we define *jitter* of a packet as the delay variation relatively to the last packet of the stream, i.e.

$$Jitter \{P_N\} = |Delay \{P_N\} - Delay \{P_{N-1}\}|$$

This definition is in accordance with the Type-P-One-way-ipdv as defined in IETF RFC 3393

$$AverageJitter = \frac{JitterSum}{RxPackets}$$

Design/Methodology

Simulation or Real Lab

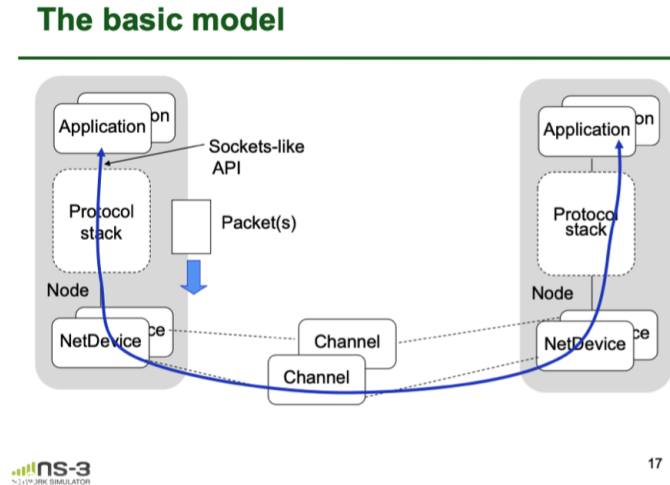
The first question of designing the said experiment is to simulate or not. Simulation might be better in our study as we can have more control over the considerations we mentioned. The biggest downside is how can we be sure that the result is “real” ? The answer to that is as long as we have a clearly defined scope for the experiment and frequent sanity check on these scopes, we can have an idea of how “real” the results can be.

Tool: NS-3

ns-3 is a open-source, discrete-event network simulator used mainly for research or educational purposes written in C++. The simulator itself has a comprehensive documentation available online.

ns-3: basics

For our present purpose, we only need to understand the main abstractions of ns-3



17

Figure 2: Main abstractions of ns-3

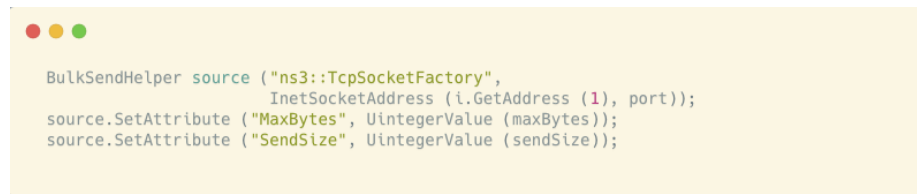
Application It is like processes in our computer, it uses “sockets” to send data to lower layer. We will install a off-the-shelf `BulkSendApplication` in ns-3 using TCP socket and an `UDPCli`ent and an `UDPServer` for sending UDP packets and a `FlowMonitor` for both case to monitor the packets.

UDP : Choice of sending interval We will be sending packets at a interval of 2 microseconds which is the delay of the underlying channel specified below

UDP: Choice of MaxPacketSize Here it refers to the payload. Since MTU is 1500, naturally we will want to use 1500 Bytes. The IP Header and the UDP Header however require $20 + 8$ bytes, so we should use 1472 instead⁷. This is confirmed in simulation; the use of 1500 bytes packet could harm throughput performance.

TCP: BulkSendApplication “This traffic generator simply sends data as fast as possible up to MaxBytes or until the application is stopped (if MaxBytes is zero). Once the lower layer send buffer is filled, it waits until space is free to send more data, essentially keeping a constant flow of data.”.

We set the MaxBytes to the Total Bytes (PacketCount * PacketSize) of UDP to compare performance and the SendSize doesn't matter upon some experiments

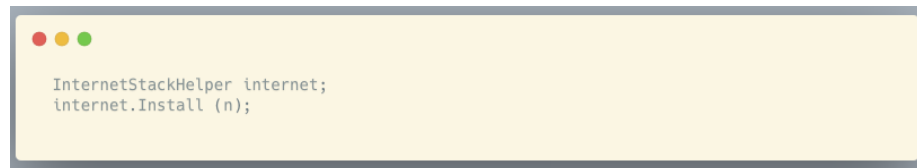
A code snippet for BulkSendApplication. It shows the initialization of a BulkSendHelper with a ns3::TcpSocketFactory, an InetAddress, and port. It also sets attributes for MaxBytes and SendSize.

```
BulkSendHelper source ("ns3::TcpSocketFactory",  
    InetAddress (i.GetAddress (1), port));  
source.SetAttribute ("MaxBytes", UintegerValue (maxBytes));  
source.SetAttribute ("SendSize", UintegerValue (sendSize));
```

Figure 3: Code for BulkSendApplication

Protocol stack Also called the **InternetStackHelper** which usually includes UDP,TCP Sockets, IPv4, and for our present purpose a **TrafficControlLayer**, which includes a queue and when it is full it would notify the upper layer

NetDevice This corresponds to both the hardware (Network Interface Cards, NICs) and the software drivers of them and also included a queue for packets, so there are two level of queueing in **ns-3**

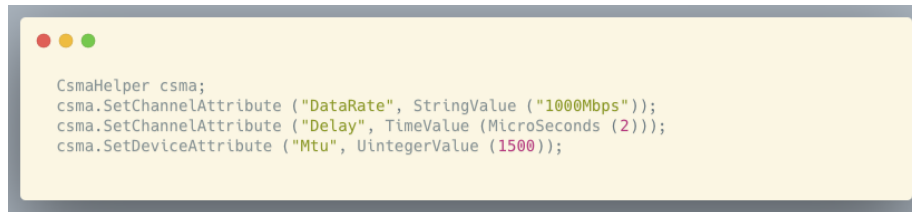
A code snippet for InternetStack. It shows the creation of an InternetStackHelper object and the installation of the stack.

```
InternetStackHelper internet;  
internet.Install (n);
```

Figure 4: InternetStack

⁷Eric Gamess, Rina Surós, An upper bound model for TCP and UDP throughput in IPv4 and IPv6, Journal of Network and Computer Applications,Volume 31, Issue 4, 2008, Pages 585-602, ISSN 1084-8045

Node It is the end hosts by which we can install **Application**, **InternetStack** and **NetDevice** to, similar to our computer. We will be creating two **Node** and connect them with something similar to ethernet cable.



```
CsmaHelper csma;  
csma.SetChannelAttribute ("DataRate", StringValue ("1000Mbps"));  
csma.SetChannelAttribute ("Delay", TimeValue (MicroSeconds (2)));  
csma.SetDeviceAttribute ("Mtu", UIntegerValue (1500));
```

Figure 5: Code for CSMA Channel

Channel Provides communication channel to **Node** and we will be using a CSMA channel (like Ethernet). The important details here is the choice of these attributes available for tweaking: MTU, DataRate and Delay. We will be using standard MTU (no Jumbo frame, sometimes available to specific network), gigabit and a delay of 2 microsecond. The reason for 2 microsecond is for light to travel 600m, as compared to other studies use of 0 delay.

Result

Results are as follows:

Throughput



Figure 6: TCP and UDP Throughput (Mbps)

We observed that the throughput for UDP for 14720, 147200 Bytes sent exceeded 1000Mbps (Our bandwidth is 1Gbps). We then looked at the loss of packets

We can see that the in 147200 Bytes 97 out of 100 packets are lost. It is safe to say that we should drop these 2 and further investigate

We can also see that TCP throughput gradual increased and flatted out at around 400Mbps.

Average Delay

Due to the lack of traffic control, UDP packets can be “stuck” in lower layer’s queues. The result might also due to our rapid always-on sending packets causing the congestion. Further investigation will be need to confirm, such as lowering buffer to see if delay actually increases.

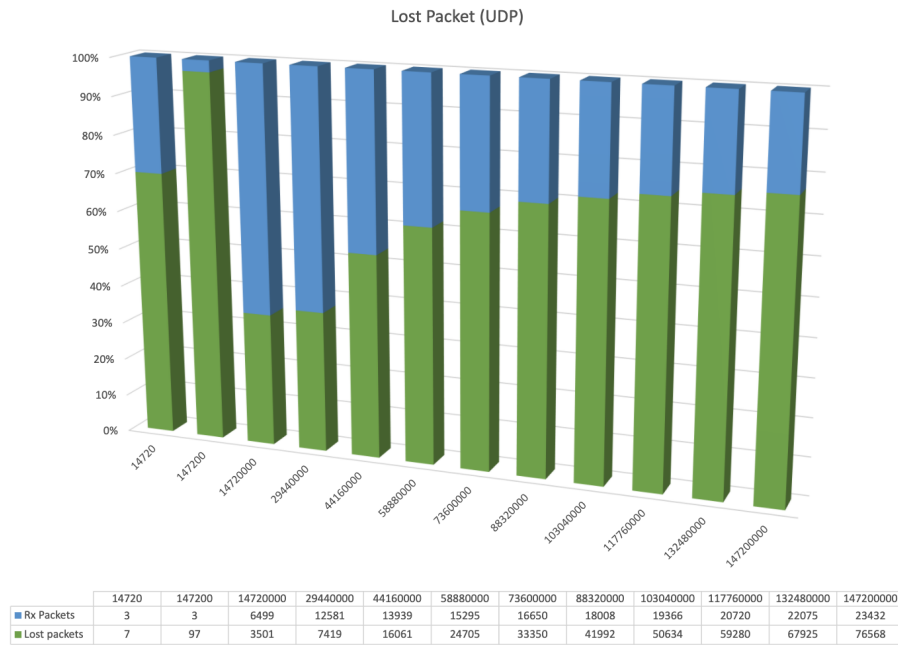


Figure 7: UDP Lost packets

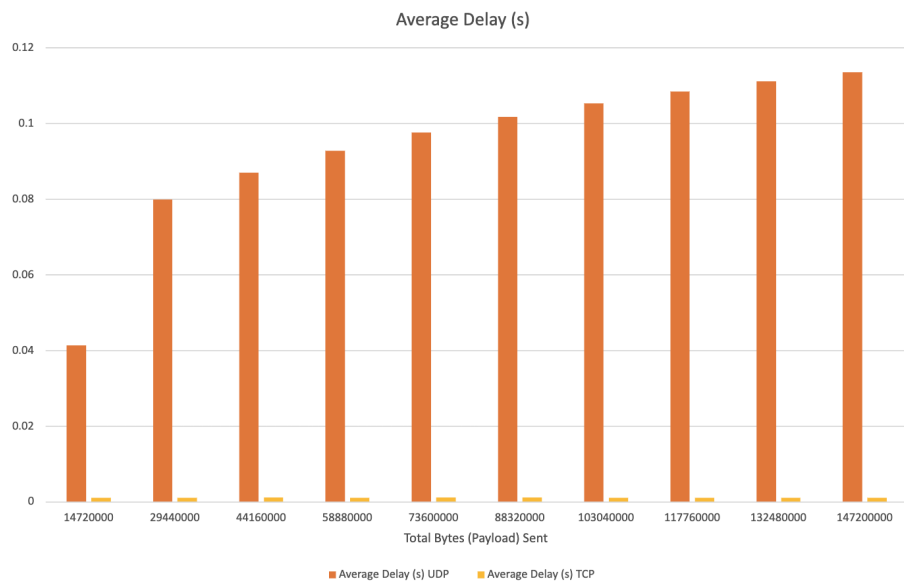


Figure 8: Average Delay for TCP and UDP

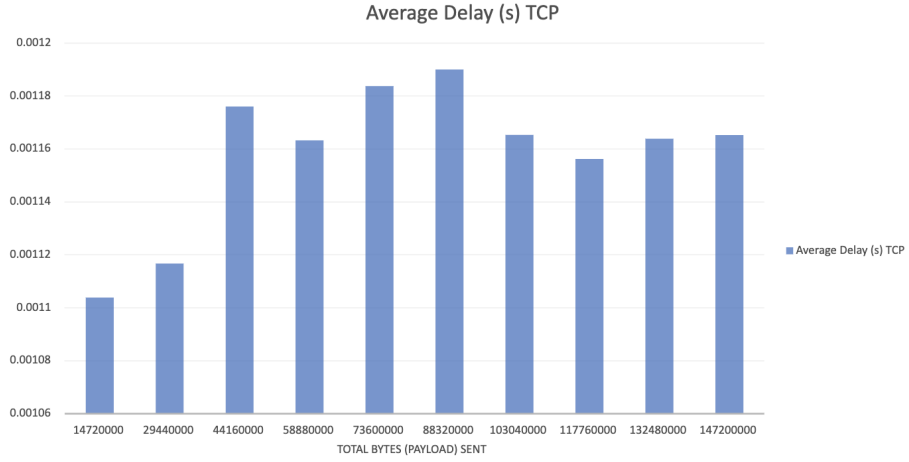


Figure 9: Average Delay: TCP

TCP

The average delay exhibited dips just after “peaking” and then gradually increase again. This might be due to the `cwnd` adjusting in congestion avoidance. Further investigation is needed as we need to confirm the algorithm for congestion control actually produces such patterns because there are many choice for it in `ns-3` (Veno, Cubic... etc)

Average Jitter

The average jitter is expected to decrease as the system “stables” out and the variation in delay would be less.

TCP jitter larger than UDP’s

The absence of traffic control in UDP means that the packets will be continuously push down the stack and be sent without interference, the variation in delay should only be minimal while packets’ delay of consecutive packets could be large because of the said traffic control.

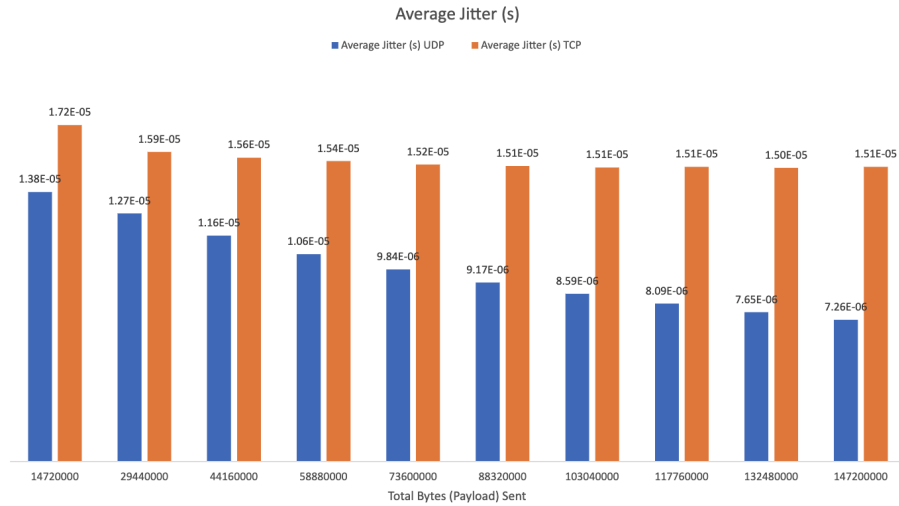


Figure 10: Average jitter of TCP and UDP

Discussion

Why not compare packets lost?

The `BulkSendApplication` made sure the once the lower layer send buffer is filled, it would wait. So the only place we could drop packets is in the channel which we will need to introduce error into.

`UDPClient` has no such traffic control so we could observe the loss of packets.

So what might be the reason by which QUIC chose UDP based on our study here ?

Might simply be the throughput (greater throughput) and the jitter (less variation of delay). The larger average delay is mainly due to the lack of traffic control which the QUIC aims to move the “congestional control algorithm into the user space ... rather than the kernel space”⁸.

Code

<https://github.com/alfredtso/ns-3-project>

⁸QUIC

Reference

ns-3 training resources

HTTP/3 draft