The effects of an AQM enabled home gateway on First Person Shooter game traffic

Russell Collom*
Centre for Advanced Internet Architectures, Technical Report 160826A
Swinburne University of Technology
Melbourne, Australia
collom_r@hotmail.com

***Abstract*—At the home, more and more devices are connecting to the internet via their home gateway to utilise a variety of different services. Online gaming, in particular first person shooter games (FPS), require a low latency connection with only a small percentage of tolerable packet loss in order to keep the game state feeling responsive. Today, this is becoming difficult to maintain with the internet usage patterns and available upload/download speeds for a typical household. With all traffic being funnelled to one bottleneck in the home gateway, when congestion does occur, there is a spike in the delay time, causing game traffic to lose its responsiveness. Emerging technologies such as Active Queue management (AQM), are being tested and used in order to mitigate this problem by getting TCP to react sooner through the implementation of early packet drops. Via using the CAIA developed tool TEACUP, for running experimental research on TCP, we can explore what impacts, both positive and negative, different types of AQM's have on FPS games.**

***Index Terms*—CAIA, TEACUP, AQM, FPS**

I. Introduction

In the world of today, the need to have everything at our fingertips, and going simultaneously, is causing a significant impact on networking infrastructure. As more applications are being introduced and utilised, users at the home are starting to notice that these services do not work as optimally as they used to. Different applications have different requirements, but online video games, and first person shooter (FPS) games in particular, are being greatly effected by having multiple services running at the same time over a fixed internet connection. Increasing the download and upload speed would resolve these problems, however FPS game traffic only sends out its User Datagram Protocol (UDP) packets in small, semi-regular updates to inform all players of the current game state.

This inherently makes it so that round trip time (RTT), packet-loss and jitter, are the only key requirements for FPS traffic, and that as long as every packet can get through in a timely matter, the speed of the connection is unimportant. One technique that can be used to facilitate game traffic meeting these key requirements is Active Queue Management (AQM), used at the home gateway bottleneck to shape all traffic that passes through it.

In this report we will explore what different AQM methods can be implemented in order to remove this problem, and the effect they have on game traffic. Section II will cover what prior research has been done in the field, as well as some information on the technologies that will be implemented. Section III will describe the testbed being used to conduct the experimental research, as well as the various components and elements used. From this, Section IV will illustrate and comment on all the results obtained by the various experiments run, with finally Section V summarising and concluding the findings of this report.

II. Background and Prior Research

Online multiplayer games are something that have constantly been on the rise in terms of its impact on network infrastructure, however it is not something that requires heaps of bandwidth available for it, simply just a stable connection without long delays. The key goal of a video game is to offer the player an immersive and interactive experience that makes the user feels in control of their character at all times. When it's simply a local connection that isn't a problem as there is only very small RTT being experienced, somewhere in the range of 1 to 10 ms. For however online games, in which the user is geographically separated from the other players, suddenly a more significant RTT is being introduced

*The author performed this work while a Bachelor of Engineering undergraduate winter intern in 2016, supervised by Professor Grenville Armitage.

based on this physical distance, ranging typically from 10 ms up. This will be represented as RTT_{base} in this report.

To check some of the RTT_{base} introduced from physical distances in Melbourne, Australia, we found a website for public Counter Strike Source Servers [10] (which is a type of online FPS game), which where pinged from a residential address in Melbourne. Table I features the RTT_{base} found by conducting this search.

Table I
RTT_{BASE} FOR COUNTER STRIKE SOURCE SERVERS FROM MELBOURNE

Location	RTT
Sydney	20 ms
California	167 ms
Washington	239 ms
London	330 ms

From Table I, it illustrates the general idea that from Melbourne to Sydney, there is an RTT_{base} introduced of 20 ms, so regardless of any additional RTT experienced, when playing on that server from Melbourne, a user will see at least a delay of 20 ms. The same applies for the other servers in Table I as well, when playing on the London server there will always be a delay of 330ms or more. This will inherently change based on where a user is physical located with respect to their nearest game server, but their total RTT experienced will always be factored in by this distance.

When a user is playing an online game, there are three ways in which traffic is typically communicated. First being the Peer-to-Peer model, in which every client is connected to every other client in the game, transmitting their game state to each other, creating potentially a lot of overhead for each client as more clients are added [3]. The more common or typical model is the client-to-server model, in which each client only sends it's personal updates to the server, which then gathers all that information and only sends relevant information of the current game state back to each client. Finally there is a variant on the model called distributed client-to-server, which can have multiple servers communicating information between each other, otherwise exhibiting the exact same properties as a client-to-server model.

Since game server status updates only care about the now, and any information lost in the past is too late to matter, UDP is often used for its transportation. This is because UDP doesn't re-transmit lost packets, unlike Transport Control Protocol (TCP) [3], allowing it to maintain a better RTT for FPS games. This works well with UDP additionally, as online FPS games do not need flow control, and only require that the game state updates as frequently as required by the server in order to keep the experience immersive.

Most of the main functional components in game traffic are all facilitated via the UDP protocol. The first being how the client probes to find the available servers. The clients machine essentially contacts the master server for a given game, which then proceeds to probe its list of active servers, of which the client can then select and join [3]. As the game commences there is basically small UDP packets being sent back and forth between the clients and server, in order for the match to be propagated to all users equally. This is where performance requirements such as low latency, minimum packet loss and jitter all effect the players, ruining the perceived shared world if this criteria isn't meet.

A. Research on online multiplayer game traffic

With online gaming being such a growth industry, unsurprisingly there has been a lot of prior research [4] quantifying online multiplayer games and determining what elements have the greatest effect. Interestingly, the paper [17], studies the effect of latency on a non FPS called Warcraft III. This game is what is known as a Real Time Strategy (RTS) game, and games in this genre are more in the lines of chess, then a shooter game. Because of this it was found that latency was tolerable to a reasonably high level, with users in their study only being effect by RTT between 500ms - 800ms.

Another study conducted by the Institute of Information Science [9], explored the sensitivities of a Massive Multiplayer Online (MMO) game, and seeing the effect that latency, loss and jitter had on a users playtime. The findings from their study can be found in Figure 1.

From Figure 1, it was clear that they found a direct correlation between the length that a user wanted to consume a game for, and the effect latency, loss and jitter had on the user. In the report [5], it was found that even though jitter would possibly have an effect on a players experience, it may be insignificant compared to the latency effecting the game. This is because that in order for a user to experience any significant jitter, they would already be suffering from excessive latency, which in turn is most likely what the user is noticing the most. For this reason, the experiments in this report will just be focused on packet loss and latency.

A number of game traffic studies have been performed at CAIA, such as [2] where it was found by comparing two servers that appeared identical except for their

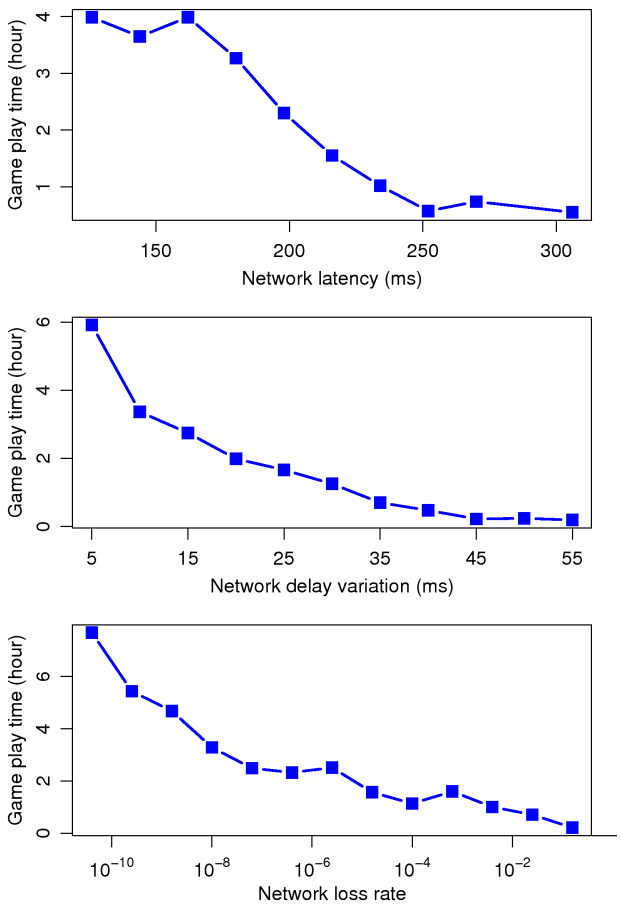

Figure 1. Player Sensitivity to latency, loss and jitter [9]

RTT_{base}, one in California and the other in London, that the majority of players would always select the server with the smaller RTT_{base}. Once again highlighting the sensitivities of players to RTT, and how in the gaming culture there is already a understanding that less latency equals better performance.

Different papers have explored different kinds of FPS games, for example [7] explored the effect of latency and packet loss on Unreal Tournament, finding loss rates where unnoticed up to 3%, and that latency of 100 ms where noticeable on shooting, but not movement until around 300 ms.

Other papers such as [18] and [21], focused on Quake IV and Quake III respectively. For Quake III, it was found that a latency was tolerable up to 150ms, and that a packet loss wasn't noticed until around 35%. This would suggest that Quake III has particularly good server-side recovery, and be able to deal with multiple losses in UDP packet's.

To see the effects ourself, we set-up our own little test scenario with two players, with one of them being aware of the traffic shaping occurring. This was done by following the process described in [19], in order to play a Quake III multiplayer match, on the custom map of the CAIA EN605 lap room. We used the tool dummynet on a FreeBSD machine in order to shape the traffic to various simulated delays and losses [16]. Table II illustrates the latency and packet loss rates we iterated over.

Table II
DELAY AND LOSS ITERATED OVER FOR QUAKE III EXPERIMENT

Delays (ms)	Loss (%)
0, 10, 25, 50, 100, 150, 200	0, 1, 3, 5, 10, 20, 30, 40

We randomly went through the the different values in the table, assessing how one felt the game played in each case and comparing it to the other player who had no knowledge of what was changing, he was just looking at how his performance was effected with each alteration. We found, like research before us [21], that between a latency of 50 ms to 150 ms, there was a delay that was noticeable, after which the latency caused the game to became very difficult to play, with the shooting constantly having a delayed rate of fire.

In terms of packet loss though, it wasn't noticeable at all between the 1-10 % range, but after that, occasionally in a match a shot would not collide, or a user would jump position slightly, not to mention the increase in time it would take to join a match. Although not as frequent, or immediately obvious as the effect latency has on a FPS game, it is definitely something that can have an impact on a player, and depending on how efficient a game is with dealing with packet loss, may cause an negative experience.

For all these reasons above, both latency and loss are the fields this report has focused on when looking at AQM's effect on FPS traffic.

B. Active Queue Management

One of the key ways in which typical home gateways manage the constant arrival of packets is to queue them in a buffer. The most common and simplest way to do this is via the packet first in first out (FIFO) method, in which there is a fixed line, and only the packet at the end moves along, sequentially going through the line. If the fixed bandwidth speed is significant enough, then a queue might rarely form or fill up, however if it is less then what traffic is being pushed down it, then a

bottleneck is created, and the buffer acts as a way to deal with this.

As technology has improved, it has become easier and cheaper to make these buffers excessively large in order to make sure nothing is dropped. Since TCP uses packet loss as a way of determining a path capacity, it will constantly fill up these buffers, causing a rise in RTT as it fills up, until TCP finally detects congestion and backs off. As highlighted in section 2.1, the delay introduced in these larger buffers will have a significant impact on game traffic. This phenomenon is what is known as Bufferbloat [15], and is driving the need to utilise smarter queue management in order to better manage network congestion.

This is where AQM's comes in, as it is essentially the idea of getting TCP to back off early through pre-emptive packet drops and marking, in order to prevent the buffer from ever filling up to the stage where significant RTT could occur. The following is a list of different AQM's that can be implemented:

1) Proportional Integral controller Enhanced (PIE) - Randomly drops a packet from the tail end of a queue based on a latency calculation [15]. The latency calculation involves using the delay samples over a period of time to predict whether or not the latency in a particular queue is increasing or decreasing, and thus whether to increase the drop probability, or decrease it. In addition to this, PIE supports a level of burst tolerance, by accommodating for brief periods where the queue can be filled above it's target delay.
2) Controlled Delay Management (CoDel) - Identifies whether a packet has been in a queue too long, and then drops the packet from the head of the queue if this is the case. CoDel essentially does this by time-stamping packets that enter the queue to get the local delay (sojourn time), and comparing it to the set target delay [14]. Provided the local delay is below the target delay, then packets are forwarded fine, however, if this delay is above the target threshold, a time drop (Td) value will be set for the next packet, which if the condition is meet again, will be dropped. Like PIE, CoDel also features support for burst tolerance.
3) Flow Queuing CoDel (FQ-CoDel) - Adds the benefits of a modified scheduling algorithm DRR (Deficit Round Robin) to create different sub-queues to fairly share between for each form of traffic, which CoDel is then applied on. These queues are essentially formed by hashing the packets source, destination IP address and port numbers, in addition to the IP protocol number and a randomly assigned number when the hashing commenced [11]. This allows different types of traffic to have its own queue, and if they occur infrequently enough, potentially never queue more than one packet at a time.
4) Flow Queuing PIE (FQ-PIE) - Recently being tested at CAIA, it is essentially the same process as FQ-CoDel, except the sub-queue's are utilising the PIE algorithm instead of CoDel [1].

III. Experimental Testbed

In order to conduct this research, a tool was required that could emulate realistic traffic generators and observe the effect with various AQM's. Thankfully a tool developed by CAIA already exist, known as TCP Experiment Automation Controlled Using Python (TEACUP). In addition to this, a physical testbed had already been set-up at CAIA, illustrated in Figure 2, to run these experiments.

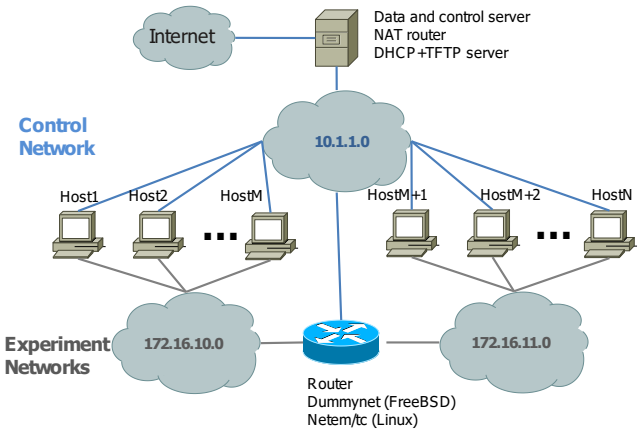

Figure 2. TEACUP testbed topology [6]

Additional documentation on TEACUP and how to configure or setup experiments can be found via the following references [13], [20], [22]–[24]. This documentation was used as resource in order to set-up the experiments for this report.

A. TEACUP Tools

For the experiments conducted in this report, the three main tools that that where used to generate different types of traffic where:

1) Pktgen - Developed at CAIA in order to generate synthetic game traffic based on pre-existing traffic [12]. This in itself was based on prior research

done at CAIA on using Autoregressive moving average (ARMA) models to use empirically gathered two and three player game length traffic, to generate a realistic version of that game with more player in it [8].

2) Iperf - TEACUP uses a slightly modified version of the iperf tool for the generation of TCP or UDP flows [23].
3) Httperf - Used to emulate HTTP type traffic, a modified version is used in TEACUP to set-up DASH like traffic for streaming content at different representation rates [23].

B. Case Scenario

For this experimental research, analysis of a case scenario with a variety of different requirements was required in order to fully understand the effects of a AQM on game traffic. Figure 3 highlights what a typical household may be using at once, which I used as the basis to form all the experiments to run. AQM's were set-up at router where the bottleneck is, for which all flows will go through.

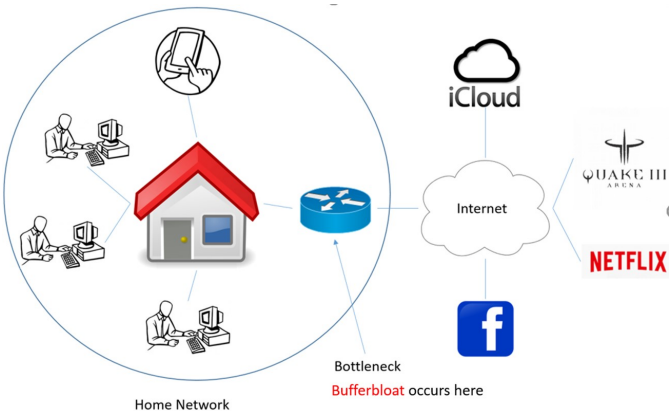

Figure 3. Possible Home network

From the CAIA TEACUP testbed I utilised four devices. One was acting as a client on subnet 172.16.10.0 (from Figure 2), another was acting as a server on the other subnet 172.16.11.0, and then a router was used to facilitate all communication between them. In addition to this, a control host is required in order to set-up and gather all of the experimental data.

Pktgen was used to generate all of the game traffic, being the key traffic being analysed in the experiments. This was done by telling TEACUP that one of the host was acting as a game client, and the other was acting as the game server. As game traffic needs to both send and receive status updates, this effects both the downstream and upstream direction.

Iperf was used to generate a bulk TCP upload to emulate typical traffic such as uploading files from a phone to the cloud. This occurred 10 seconds into each experiment, and occurred for the remaining time the experiment was captured for. As this was simulating a cloud upload, this traffic's impact is on the upstream connection.

Additionally, iperf was also used to generate random bursts of TCP traffic to emulate someone doing web browsing. Every 10 seconds of the experiment a different TCP connection would occur for a amount of time calculated using pseudo-random numbers, to emulate the variance of time required to download different web pages. This connection was acting as retrieval of web pages, and thus its effect is in downstream direction.

Finally httperf was used to set up a DASH stream of different rates to produce a streaming like application such as Netflix. A HTTP server was started on the host emulating a web server, and DASH like content was created on it in terms of five second chunks, with the representation rates of 500, 1000 and 5000 Kbps. In order to emulate DASH like traffic, these five second chunks occurred two seconds apart, going from lower to higher rates, before repeating the process again every 10 seconds to represent a adaptive type stream. As the httperf was acting as bursts of DASH like traffic, its impact is in the downstream connection.

The following is all the main parameters that where set-up for this reports experiments:

- Hosts & Router OS: FreeBSD
- Congestion Control Algorithm: NewReno
- Bandwidths: 12/1 and 50/20 Mbps down/up to see the effects of AQM's on both a smaller and larger available bandwidth. As the upsteam connection speed is smaller than the downsteam, this is more likely where queuing delays will be introduced.
- RTT_{Base} (ms): 0, 20, 40, 100, 160, 200, 300 and 400
- AQM: FIFO, PIE, CoDel, FQ-PIE and FQ-CoDel
- Buffer-size: 100 & 1000 packet buffer
- Games: Quake III & IV, Half life death match 1 & 2, Half life counter strike 1 & 2 and Wolfenstein Enemy Territory 2 Pro

The FPS game traffic was emulated as a four player game for each case, with the server sending out a single UDP stream with a packet size distribution equivalent to if it had four clients. On the other end, one machine was acting as a single client, receiving the servers updates and sending back periodically its own state.

TEACUP iterated through all of these experiments in every combination, each being captured for 60 second intervals.

IV. Experimental Results

All graphs shown for this section where done using Quake III synthetic game data, as all the other games showed very similar graphs, or there wasn't anything particularly unique about their traffic.

A. Just Game traffic

In this sub-section, only Quake III synthetic game data is featured, with no additional traffic to highlight what the game traffic looks like captured in TEACUP without being effected by other traffic.

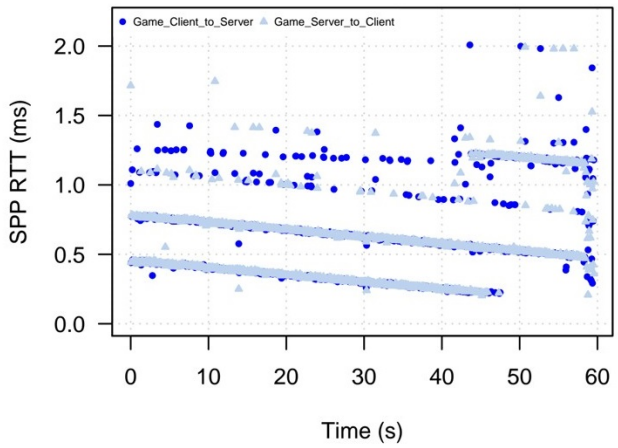

Figure 4. FPS Game RTT vs Time for 12/1 Mbps down/up, 0 ms RTT_{base} and 100 pkt buffer FIFO queue

Figure 4 shows that in the absence of competing traffic, that FPS game traffic experiences below 2ms of RTT, on a path with 0ms RTT_{base}. Additionally this is using simple FIFO queuing, as AQM's wouldn't have an impact on one form of FPS traffic at this speed.

From Figure 5, the throughput footprint isn't very large for game data, which is what you would expect from traffic that only sends small updates of the current game state. As the UDP steam for the server to client side needs to incorporate more game state data to inform a player of multiple other players states, its throughput is at a higher rate compared to just the client to server side.

With only a single game traffic flow, there was no packet loss observed in either direction during the 60 second trial.

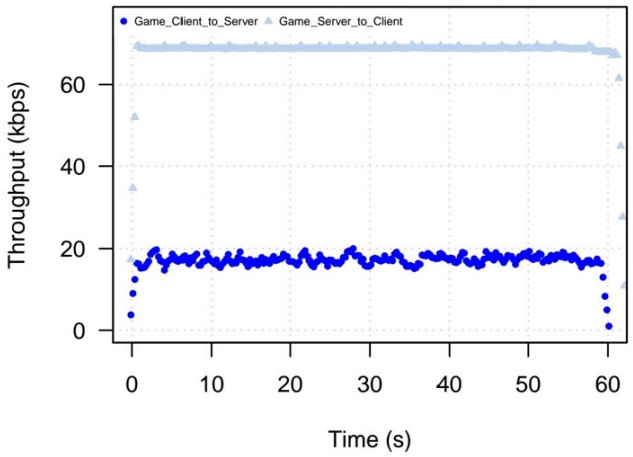

Figure 5. FPS Game Throughput vs Time for 12/1 Mbps down/up, 0 ms RTT_{base} and 100 pkt buffer FIFO queue

B. Game traffic in combination with other traffic

For this subsection, in addition to Quake III game traffic, a bulk TCP upload acting as a cloud upload is being transmitted, as well as random bursts of TCP web browsing traffic every 10 seconds, and a DASH video stream with a chunk size of five seconds at the representation rates of 500, 1000 and 5000 kbps.

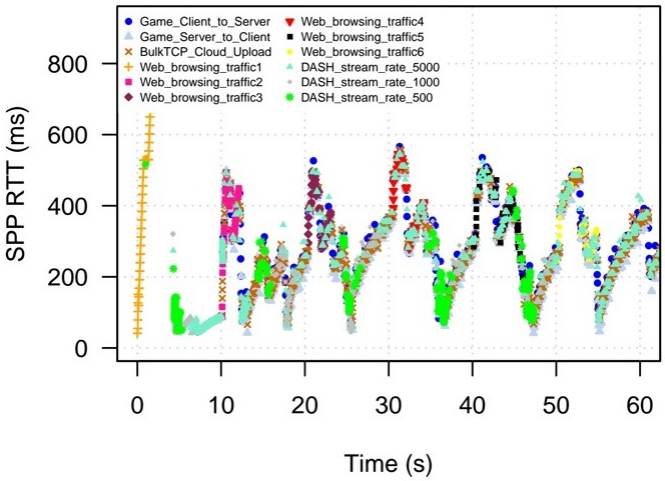

Figure 6. All traffic RTT vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 100 pkt buffer FIFO queue

In Figure 6, all the traffic has been included to highlight the effect that a combination of traffic has on RTT. Compared to that seen in Figure 4, there is a significant jump in RTT. Since congestion is now occurring, RTT for all traffic sharing the link has increased. As the connection has a bandwidth of 12 Mbps downstream, and

1 Mbps upstream, the upstream link is greatly effecting the RTT being experienced for all other traffic.

Since the bulk TCP cloud upload is the main contributing factor to the upstream, as Figure 4 and 5 showed FPS traffic consumes only very minor bandwidth, then the cloud upload is the main traffic causing the RTT to greatly rise for game traffic. Though DASH and the web traffic still have an effect, since they are in the downstream direction which has a lot more bandwidth, their impact isn't as noticeable. Additionally the web traffic only occurs briefly every 10 seconds, and the DASH traffic is adjusting between 500 to 5000 Kbps about every 10 seconds as well, so their combined impact has a greater effect around every 10 second or so.

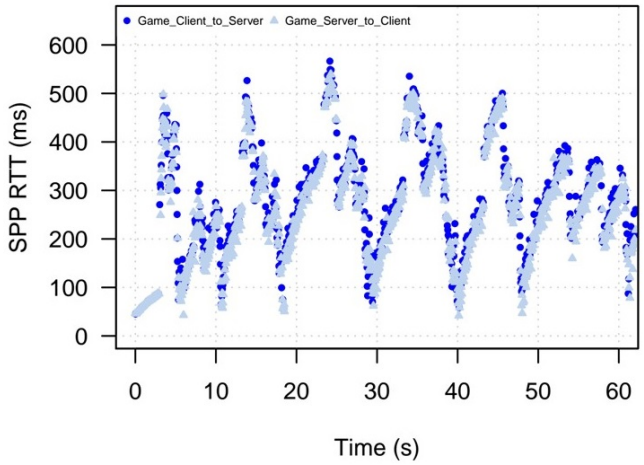

Figure 7. FPS Game RTT vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 100 pkt buffer FIFO queue

Figure 7 highlights just the game traffic, as opposed to Figure 6, which also features the additional competing traffic. It can be seen after the initial web traffic and DASH traffic commences, the game traffic starts and takes the form that can be seen after about eight seconds onwards in Figure 6. In Figure 7 it can be more clearly seen its effect on game traffic specifically, and that with simple FIFO queuing, the delay is sitting at a peak RTT of around 570 ms.

The throughput, as shown in Figure 8, is of a similar form to what it is was before the extra traffic, which is good as the same amount of game data is being pushed through. It does however fluctuate now, once again due to the congestion that is now occurring at the bottleneck.

Having illustrated the effects of a simple FIFO queue when dealing with multiple flows of traffic under a smaller bandwidth speed, Figure 9 shows the RTT over the same traffic, but applying the PIE AQM. Straight

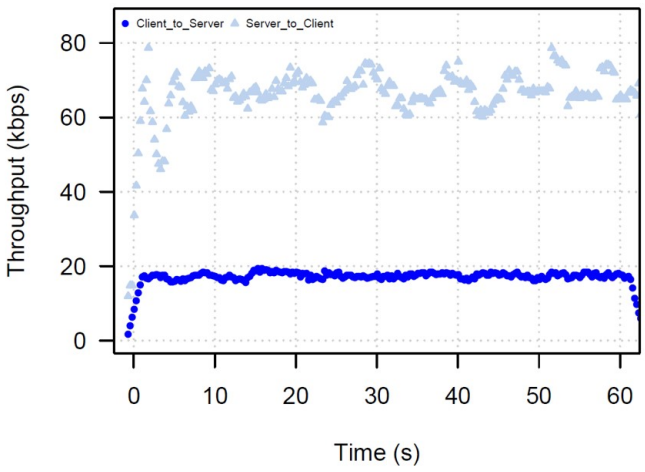

Figure 8. FPS Game Throughput vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 100 pkt buffer FIFO queue

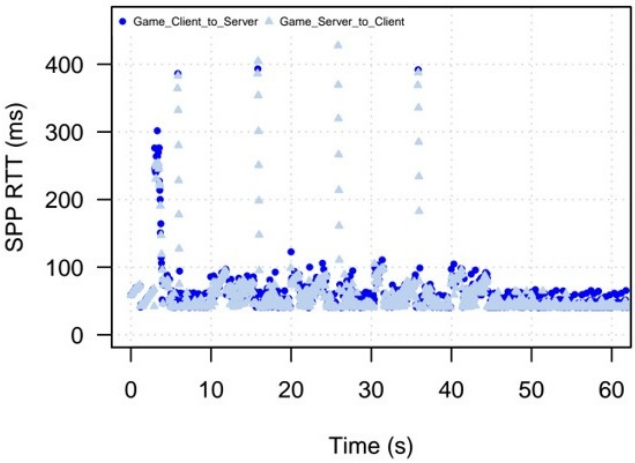

Figure 9. FPS Game RTT vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 1000 pkt buffer PIE queue

away it can be seen that the RTT has reduced itself from around 570 ms peak, to about 420 ms peak, with the majority of the RTT being experienced in the 50 to 60 ms range. Due to PIE tolerance for random bursts of traffic, when DASH is acting at it higher representation rate around every 10 seconds, PIE would be allowing it briefly through, causing the queue to fill up and the RTT spikes to occur.

Like PIE, Figure 10 shows the RTT vs Time, but now for CoDel. Similar to PIE, it performs a lot better in terms of average RTT then FIFO, but its peak RTT is at about 250ms. This is because it doesn't utilise random properties to cause it to drop packets, so even though it does still get a greater RTT when those spikes occur, they are not quite as large as for PIE.

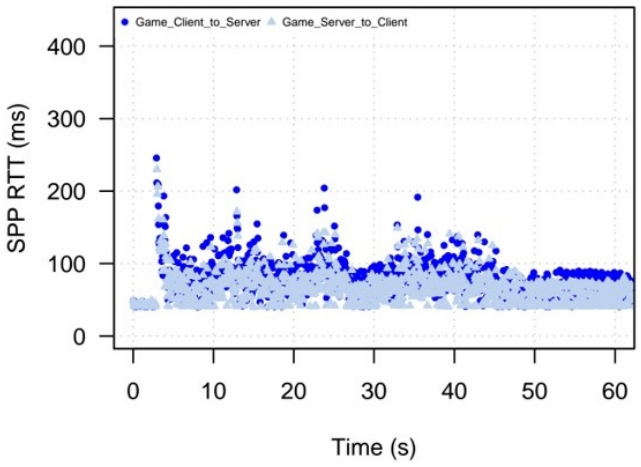

Figure 10. FPS Game RTT vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 1000 pkt buffer CoDel queue

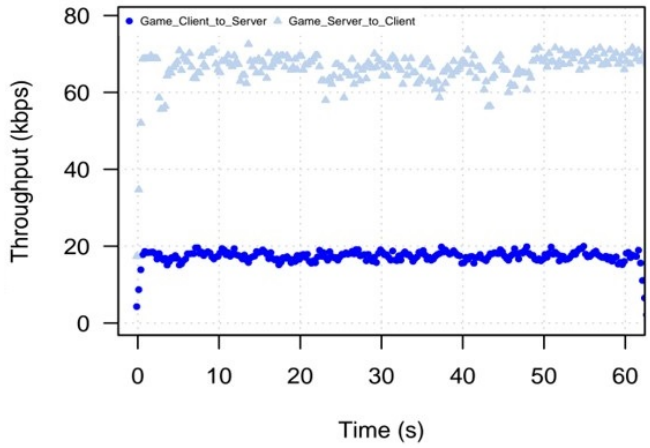

Figure 11. FPS Game Throughput vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 1000 pkt buffer CoDel queue

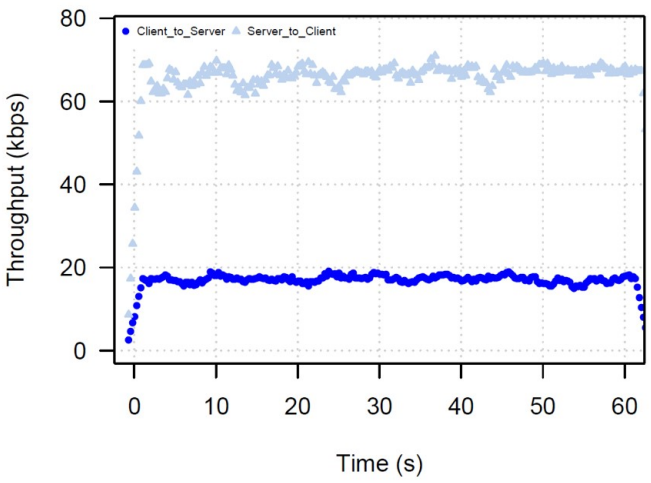

Figure 12. FPS Game Throughput vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 1000 pkt buffer PIE queue

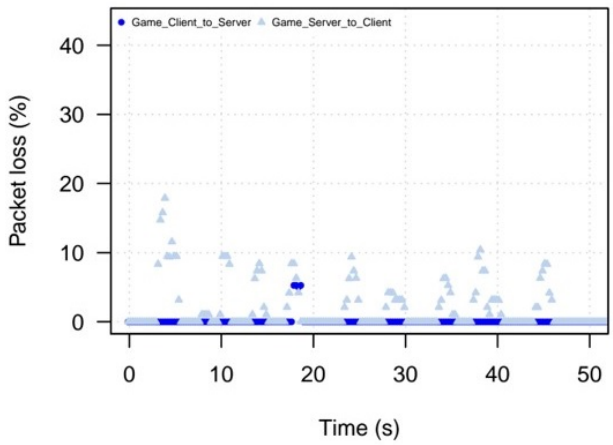

Figure 13. FPS Game Packet loss vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 100 pkt buffer FIFO queue

Looking now at the throughput, Figure 11 shows how the throughput takes a slightly better form when using CoDel in this case, compared to FIFO shown in Figure 8. As game traffic is now going through in a more timely manner, its throughput is a bit smoother.

Like Figure 11, the PIE version of the throughput graph featured in Figure 12 also takes on a very similar form with smaller fluctuations in the throughput. In general though, both of these Figures match the throughput for the same game traffic generated as when it was just game traffic.

Figure 13 indicates how now when using FIFO queuing, packet loss is happening due to the competing traffic forcing TCP to back off when a congestion occurs, and packets are being lost in this transition. These short bursts of packets being lost are likely to cause players of certain FPS games to start to notice that there is something effecting game play.

In Figure 14, interestingly PIE is experiencing more packet loss than FIFO. This would definitely cause noticeable effect to FPS games without efficient server recovery mechanisms in place for lost packets. Also this puts it above FIFO, which would be because there is a chance the packets are getting dropped just from congestion, then on top of that PIE is dropping packets early which potentially could be game packets.

In Figure 15, the CoDel packet loss figure, like the PIE and FIFO ones before it is experiencing a packet loss. Like PIE, is appears to be performing slightly worse than FIFO when it comes to packet loss, as the packet it drops based on its latency calculation could end up being a game packet.

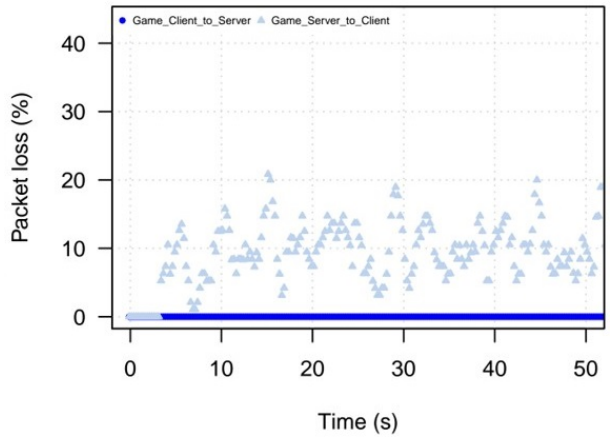

Figure 14. FPS Game Packet loss vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 1000 pkt buffer PIE queue

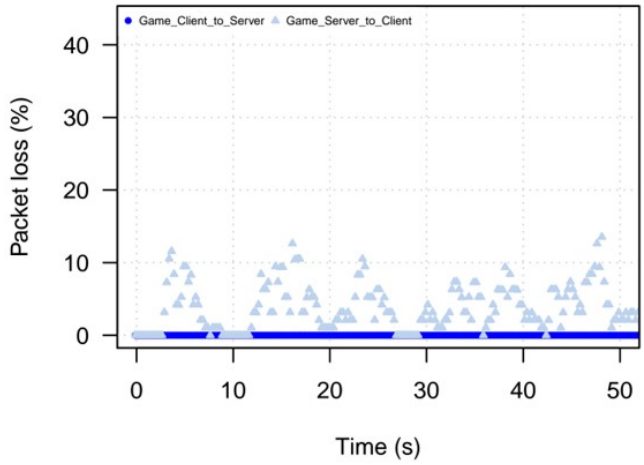

Figure 15. FPS Game Packet loss vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 1000 pkt buffer CoDel queue

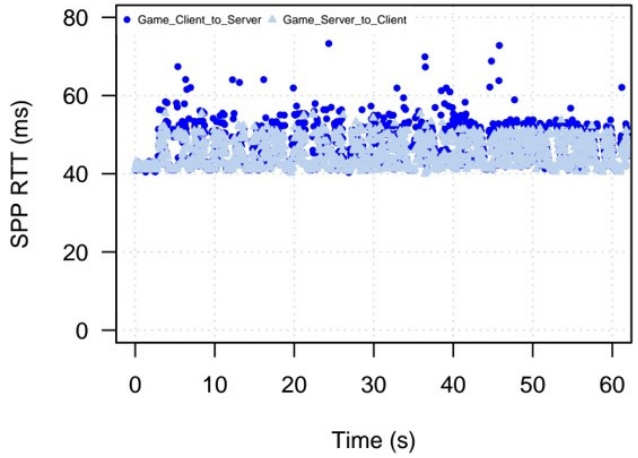

Figure 16. FPS Game RTT vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 1000 pkt buffer FQ-PIE queue

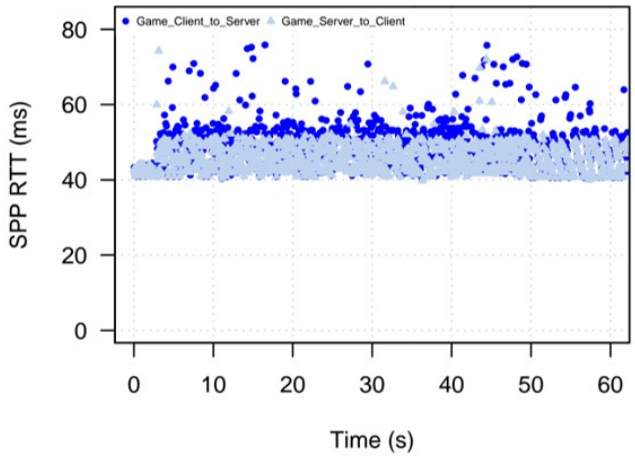

Figure 17. FPS Game RTT vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 1000 pkt buffer FQ-CoDel queue

Shown in Figure 16, FQ-PIE takes on a significantly smaller RTT then that of PIE, CoDel and FIFO. Averaging at around 60 ms, with a maximum at around 75 ms, with a RTT_{base} of 40 ms means only about 20 to 35 ms of RTT is introduced through the congestion. As discussed in section II, the lower the RTT, the better experience for a game player, and RTT's of this level would be acceptable, making this AQM effective for FPS online games.

Likewise, Figure 17 features RTT vs time for the FQ-CoDel version, generating a very similar graph to Figure 16 , just going a bit higher at around 80 ms. Even at this level, FQ-CoDel is more than efficient enough for this type of traffic. Both FQ versions highlight how good the separation of the flows into their own queue thorough FlowQueue isolation and then using the DRR scheduler to manage them is for reducing RTT. Since each queue deals with a specific type of traffic, they can be effectively managed without each traffic effecting each other.

The throughput shown in Figure 18 applies to both FQ-CoDel and FQ-PIE, as they were practically the same graph. As can be seen by this Figure, the throughput has mostly smoothed its self out, and is close to what it was in Figure 5 when it was just game traffic. Since each the flow is being individually queued, certain traffic like game UDP packets are able to get through with mostly the same throughput each time, smoothing it out compared to their non-FQ counterparts.

The FQ-CoDel and FQ-PIE versions of Figure 19 produced the exact same graph for every game variant and delay. As illustrated, no packet loss at all occurred

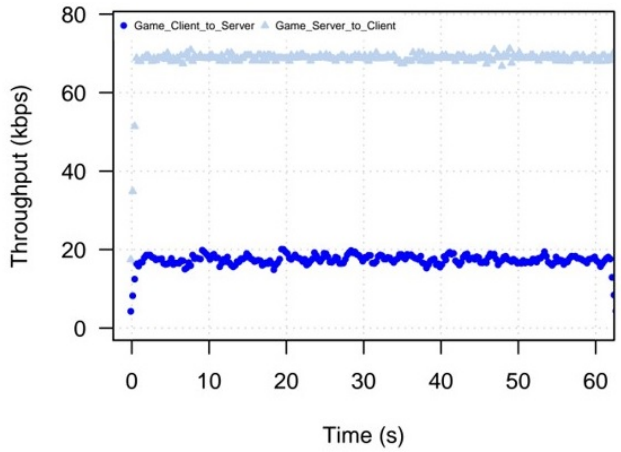

Figure 18. FPS Game Throughput vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 1000 pkt buffer FQ-CoDel queue

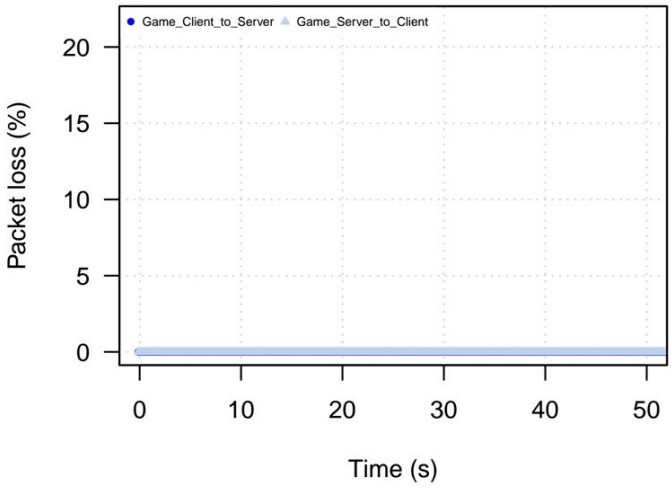

Figure 19. FPS Game Packet loss vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 1000 pkt buffer FQ-CoDel queue

for both versions of FQ, indicating these are by far the best AQM's for FPS games. Because game traffic has gaps between each update it sends, this packet would be treated as a unique sub flow each time it hits the bottleneck. This in turn would then allow the packet to get through without a significant delay, or being dropped, before a new packet arrives creating yet another sub flow for this traffic, and the cycle repeats.

Having gathered data for a variety of different base delays, we plotted the boxplots of RTT vs time for FQ-CoDel, as FQ-PIE was essentially the same, for these different delay as shown in Figure 20. The Figure shows the general idea that using one of the better AQM's to reduced RTT, with the RTT_{base} already introduced via the physical distance, a user could look at this graph

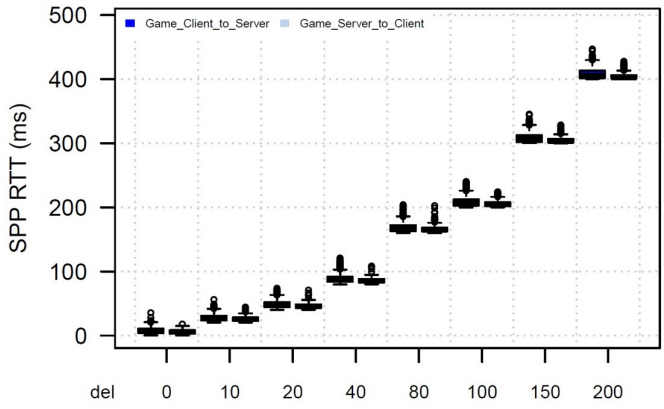

Figure 20. Boxplots for FPS Game RTT vs Time for 12/1 Mbps down/up and 1000 pkt buffer FQ-CoDel queue

and see with this AQM the typical RTT they would expect to get for servers with a certain RTT_{base}. This is under the condition of a smaller download and upload bandwidth, and a variety of different traffic congesting the bottleneck.

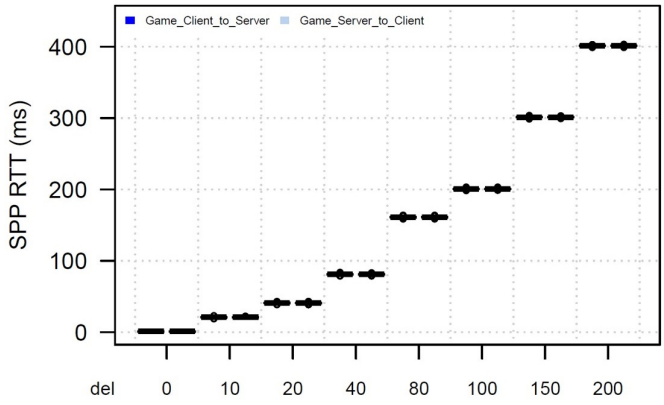

Figure 21. Boxplots for FPS Game RTT vs Time for 50/20 Mbps down/up, 40 ms RTT_{base} and 1000 pkt buffer FQ-CoDel queue

In Figure 21, the graph illustrates the same point as Figure 20, except that this was for faster bandwidth speeds. As shown, the boxplots size are a fair bit smaller than their 12 Mbps down version, since the available bandwidth allows a lot more traffic to be on a link without queuing in the buffer. This in turn highlights how with greater speeds, AQM's don't necessarily have an effect, as the congestion causing the spike in RTT to happen may not occur if all the traffic happily goes down or up a link.

From Figure 19 and 20, it is clear to see that for the FQ variants of PIE and CoDel, they only have a positive impact on FPS game traffic, and even when there is a

lot traffic going down a link of smaller capacity, they can effectively reduce RTT and packet loss to acceptable levels.

C. Impact of upstream vs downstream traffic on FPS game traffic

As certain types of traffic exhibit different traits, we also wanted to explore the effect that just traffic in the smaller 1 Mbps upstream direction had, compared to traffic in 12 Mbps downstream direction. For this, DASH like traffic was used for the competing game traffic in the downstream direction, and a bulk TCP upload was used in the upstream direction separately.

DASH operates in small chunks, in this case of five seconds, with different representation rates, in the downstream direction, so it's may only have minor impact compared to something like a bulk TCP upload which will consume all bandwidth it can in the upstream direction. For this sub-section, we have graphs illustrating the different effects that DASH traffic has on game traffic, compared to a simple bulk TCP upload.

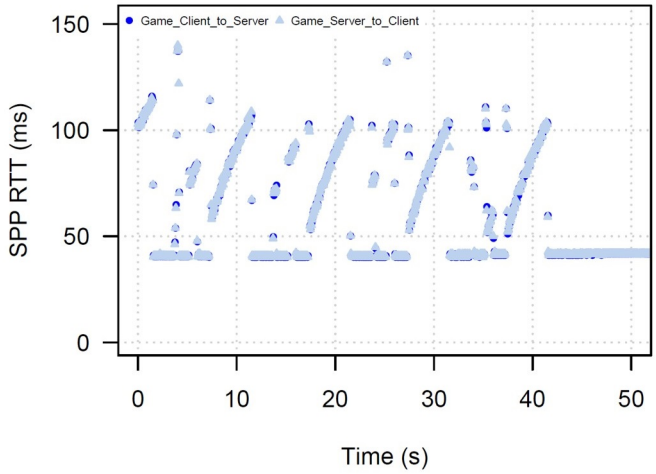

Figure 22. FPS Game RTT vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 100 pkt buffer FIFO queue with competing DASH traffic

From Figure 22, the RTT experienced is already a lot lower then was found in Figure 7, in which there where four competing forms of traffic. As the DASH like traffic used in this TEACUP experiment is essentially three representation rates operating at 10 second intervals for five seconds, the graph is slowly growing and decreasing based on these rate forming the graph seen. Additionally because DASH utilises the downstream connection, and 12 Mbps is available, the game traffic is performing better even for just simple FIFO, as its impact on the upstream isn't significant.

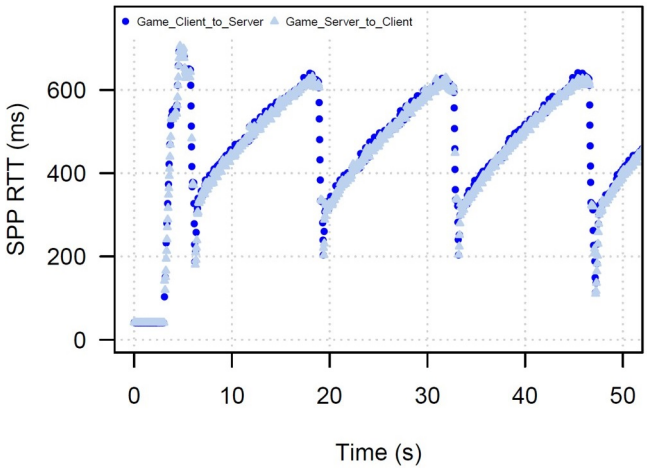

Figure 23. FPS Game RTT vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 100 pkt buffer FIFO queue, with competing Bulk TCP traffic

Comparing Figure 23, to Figure 22, it can be seen how the bulk TCP upload is one of the main factors causing the RTT to be a lot higher when all these forms of traffic are combined. This is because the bulk TCP upload is simply consuming all available bandwidth it can in the upstream direction which is limited to 1 Mbps, for as long as it can, before TCP experiences a congestion and backs of, and then proceeds to do the same thing again for the entirety of the experiment.

Additionally, the throughput from the game traffic and DASH traffic, compared to with the bulk upload, take on a different form. With the DASH traffic, the game traffic throughput remains mostly smooth as game traffic is able to more easily get through without fluctuations. The bulk upload on the other hand, introduces a lot of fluctuations in the throughput for the game traffic, getting something very similar to what was found in Figure 8. Once again this is because the bulk TCP upload is significantly increasing the RTT being experienced by the game traffic, and thus the more significant fluctuations in throughput.

Figure 24 shows how for the bulk upload, a packet loss is being experience in the game traffic for a simple FIFO queue. For the same situation with DASH traffic though, no loss is being experience at all. Once again, showing how DASH is able to operate along side game traffic, without too much impact on it's RTT, and with no packet loss being experiences for this connection provided enough speed is present. Since the speed is more limited in the upload direction, it also is not as surprising that the bulk TCP upload is creating greater RTT for the game traffic compared to the DASH traffic in the downstream, and thus the packet loss occurring.

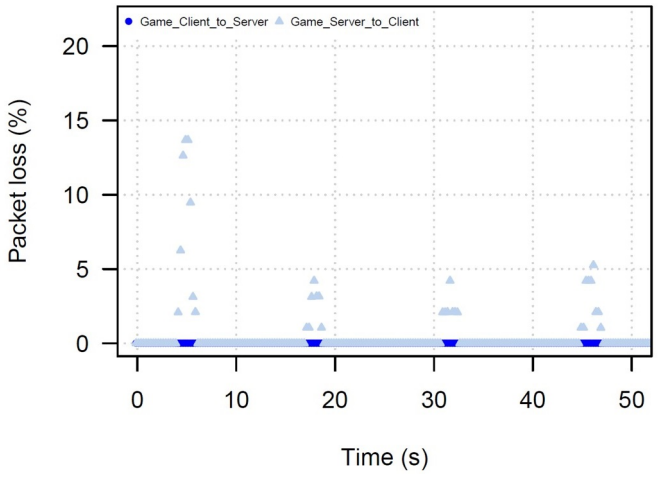

Figure 24. FPS Game Packet loss vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 100 pkt buffer FIFO queue, with competing Bulk TCP traffic

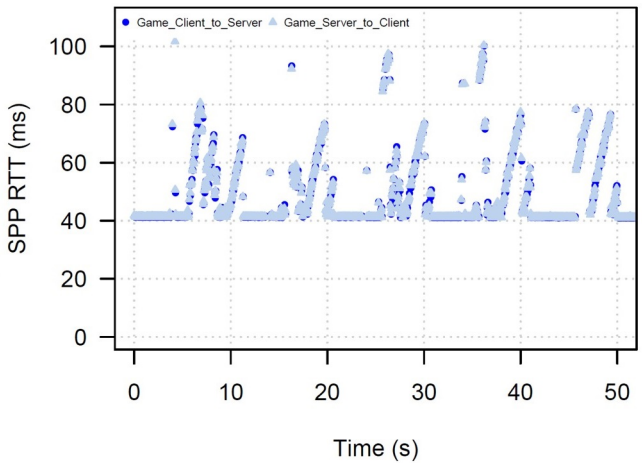

Figure 25. FPS Game RTT vs Time for 12/1 Mbps down/up, 40 ms RTT_{base} and 1000 pkt buffer PIE queue, with competing DASH traffic

Illustrated in Figure 25, the RTT is being reduced a little bit when using PIE compared to RTT experienced for simple FIFO queue, shown in Figure 22. In addition to this though, a tiny packet loss is being experienced, which would be due to the random probability latency calculation causing a couple game packets to get dropped.

The CoDel version of this graph reduces the RTT experienced to about 80 ms, with no packet loss, making it similar to what is experienced by FQ-PIE and CoDel with all traffic, as shown in earlier Figures. With just the extra DASH traffic through, FQ-PIE and CoDel are performing even better, adding only minimum RTT, close to what was found in the upload and download speeds of Figure 21.

The bulk TCP on the other hand, for each AQM, essentially takes on a form that is very close to the Figures when all traffic was added, illustrating how big of an impact that this traffic has on the emulated experiment. Since this was consuming all that it could in the upstream direction, and game traffic needs to be able to upload small amounts of packets in that direction, it's impact is a lot more noticeable than the DASH like traffic, at least for a 12/1 Mbps down/up link.

All this goes to show how certain traffic have different traits, and that depending on what type of traffic a household consumes, determines whether certain AQM's will work, or even whether just FIFO queuing will be efficient enough for a user when playing an online FPS game.

V. Conclusions

From this report, it can be seen how certain AQM's effect online FPS game traffic. By using old research conducted, as well as some quick experiments on shaping Quake III game traffic, we confirmed that latency has the most significant effect on a players game experience. Packet loss on the other hand did have an impact, but to what extent depends upon how efficient the server side is with dealing with lost game state packets.

By running experiments using TEACUP, we found that all AQM's have some beneficial impact compared to simple FIFO, especially over a smaller upload and download speed for a combination of bandwidth consuming traffic. It is clear that the FQ versions of PIE and CoDel operate very effectively, and are able to reduce RTT significantly to an operational level for FPS gamers. In addition to this, due to the way they separate flows and add in a scheduling algorithm, as well as the frequency of game state updates, no packet loss was found at all.

The normal version of PIE and CoDel did reduce RTT a bit, however due to the packet loss they induce, while running other applications simultaneously, the would have an effect on a players experience. We also found that the impact for certain types of traffic, such as ones using DASH simultaneously, are not nearly as severe to FPS game traffic when under a limited bandwidth compared to traffic such as a bulk TCP upload.

This indicates that not only the amount of traffic being sent down a link of a fixed available bandwidth plays a factor in the effectiveness of an AQM for game traffic, but also what type of form does it take in terms of consuming a users bandwidth in the downstream or

upstream direction. As the upstream link has a smaller capacity, this is where typically the congestion and rise in RTT occurs, and is a key direction the FPS game traffic operates in. To what extent this effects game traffic can be significantly reduced through the use of FQ-PIE or FQ-CoDel at the home.

Acknowledgements

I would like to express my gratitude for all the assistance provided for this research project from various CAIA members. In particular, I wanted to acknowledge the assistance of Professor Grenville Armitage for offering suggestions and general feedback for the project. I also would like to acknowledge Dr Sebastian Zander assistance on getting pktgen to work correctly in TEACUP. Additionally, I would like to thank PhD Candidate Jonathan Kua, for his support on various problems I had with TEACUP, as well as utilising the physical testbed at CAIA. The author's internship was supported in part by a Swinburne University of Technology / Cisco Australia Innovation Fund project titled "An evaluation of household broadband service requirements for educational innovation and Internet of Things".

References

[1] R. Al-Saadi and G. Armitage. "Dummynet AQM v0.2 – CoDel, FQ-CoDel, PIE and FQ-PIE for FreeBSD's ipfw/dummynet framework". Technical Report 160418A, Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, 18 April 2016. Available at http://caia.swin.edu.au/reports/160418A/CAIA-TR-160418A.pdf.

[2] G. Armitage. "An experimental estimation of latency sensitivity in multiplayer Quake 3". Technical report, Sept 2003. Available at http://ieeexplore.ieee.org/xpl/abstractCitations.jsp?arnumber=1266180&tag=1.

[3] G. Armitage. "Fun and Games - why online FPS games are interesting to network engineers". RMIT School of Computer Science & IT Seminar, 18 May 2007. Available at http://caia.swin.edu.au/talks/CAIA-TALK-070518A.pdf.

[4] G. Armitage, M. Claypool, and P. Branch. *"Networking and online games: understanding and engineering multiplayer Internet games"*. John Wiley & Sons, 2006.

[5] G. Armitage and L. Stewart. "Limitations of Using Real-world, Public Servers to Estimate Jitter Tolerance of First Person Shooter Games". Technical report, New York, NY, USA, 2004. Available at http://doi.acm.org/10.1145/1067343.1067377.

[6] G. Armitage and S. Zander. "TCP Experiment Automation Controlled Using Python (TEACUP)", June 2015. Available at http://caia.swin.edu.au/tools/teacup/.

[7] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool. "The Effects of Loss and Latency on User Performance in Unreal Tournament 2003". In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '04, pages 144–151, New York, NY, USA, 2004. ACM. Available at http://doi.acm.org/10.1145/1016540.1016556.

[8] P. A. Branch, A. L. Cricenti, and G. J. Armitage. "An ARMA(1,1) prediction model of first person shooter game traffic". Technical report, Oct 2008. Available at http://ieeexplore.ieee.org/xpl/abstractSimilar.jsp?arnumber=4665172.

[9] K.T Chen, P. Huang, and C.L. Lei. "How Sensitive Are Online Gamers to Network Quality?". *Commun. ACM*, 49(11):34–38, November 2006. Available at http://doi.acm.org/10.1145/1167838.1167859.

[10] Gameservers.com. "Counter Strike Source Servers", July 2016. Available at https://www.gameservers.com/game_servers/css-counter-strike-server.php.

[11] T. Hoeiland-Joergensen, P. McKenney, D. Taht, and J. Gettys. "The FlowQueue-CoDel Packet Scheduler and Active Queue Management Algorithm". IETF Draft, March 18 2016. Available at https://tools.ietf.org/html/draft-ietf-aqm-fq-codel-06.

[12] C. Javier. "Live Generation & Simulation of First Person Shooter Game Traffic". Technical Report 100226A, Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, February 2010. Available at http://caia.swin.edu.au/reports/100226A/CAIA-TR-100226A.pdf.

[13] J. Kua, R. Al-Saadi, and G. Armitage. "Using Dummynet AQM - FreeBSD's CoDel, PIE, FQ-CoDel and FQ-PIE with TEACUP v1.0 testbed". Technical Report 160708A, Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, 08 July 2016. Available at http://caia.swin.edu.au/reports/160708A/CAIA-TR-160708A.pdf.

[14] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar. "Controlled Delay Active Queue Management". IETF Draft, 2 June 2016. Available at https://tools.ietf.org/html/draft-ietf-aqm-codel-04.

[15] R. Pan, P. Natarajan, F. Baker, B. VerSteeg, M. Prabhu, C. Piglione, V. Subramanian, and G. White. "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem". IETF Draft, 26 March 2015. Available at https://tools.ietf.org/html/draft-ietf-aqm-pie-01.

[16] L. Rizzo. "The Dummynet Project", 2009. Available at http://info.iet.unipi.it/~luigi/dummynet/.

[17] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu. "The Effect of Latency on User Performance in Warcraft 3". In *"Proceedings of the 2nd Workshop on Network and System Support for Games"*, NetGames '03, pages 3–14, New York, NY, USA, 2003. ACM. Available at http://doi.acm.org/10.1145/963900.963901.

[18] A. F. Wattimena, R. E. Kooij, J. M. van Vugt, and O. K. Ahmed. "Predicting the Perceived Quality of a First Person Shooter: The Quake IV G-model". In *"Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games"*, NetGames '06, New York, NY, USA, 2006. ACM. Available at http://doi.acm.org/10.1145/1230040.1230052.

[19] N. Williams and C. Holman. "CAIA Lab Maps", December 2012. Available at http://caia.swin.edu.au/genius/labmaps/.

[20] S. Zander. "TEACUP v1.0 - Command Reference". Number 150529C, Melbourne, Australia, 29 May 2015. Available at http://caia.swin.edu.au/reports/150529C/CAIA-TR-150529C.pdf.

[21] S. Zander and G. Armitage. "Empirically Measuring the QoS Sensitivity of Interactive Online Game Players". Technical report, Sydney, Australia, 8-10 December 2004. Available at http://caia.swin.edu.au/pubs/ATNAC04/zander-armitage-ATNAC2004.pdf.

[22] S. Zander and G. Armitage. "CAIA Testbed for TEACUP Experiments Version 2". Technical report, Centre for Advanced Internet Architectures, Swinburne University of Technology, 2010. Available at http://caia.swin.edu.au/reports/150210C/CAIA-TR-150210C.pdf.

[23] S. Zander and G. Armitage. "TEACUP v1.0 - A System for Automated TCP Testbed Experiments". Technical Report 150529A, Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, 29 May 2015. Available at http://caia.swin.edu.au/reports/150529A/CAIA-TR-150529A.pdf.

[24] S. Zander and G. Armitage. "TEACUP v1.0 - Data Analysis Functions". Technical Report 150529B, Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, 29 May 2015. Available at http://caia.swin.edu.au/reports/150529B/CAIA-TR-150529B.pdf.