Implementing Active Queue Management at the home to reduce NBN speed demands

Jordan Boerlage*, Russell Collom*
Centre for Advanced Internet Architectures, Technical Report 161107A
Swinburne University of Technology
Melbourne, Australia
jordanboerlage@gmail.com, collom_r@hotmail.com

***Abstract*—As technology continues to involve itself in our lives, more and more applications and devices are being connected to the Internet. As these devices and applications are connected, more than ever, user's are noticing unresponsiveness and the increase in the time taken to do their daily requirements. The following research report aims to address this concern, illustrating the effect that Active Queue Management (AQM) can have on reducing latency with running multiple applications at one time. This report will first explore research that has been done in the past, looking into the theory behind the implementation of AQM algorithms. From there, we have conducted a testbed scenario in order to gain raw data on how AQM can be used in an emulated network setup, detailing it's effectiveness in real world applications. With this data, we have been able to determine how AQM has the potential to reduce the current speed demands on Australia's National Broadband Network (NBN), with popular demographics traffic requirements emulated in order to illustrate AQMs effectiveness in real world situations. Additionally we have compared the impact that higher upload and download speeds can have on latency reductions at the home.**

***Index Terms*—TEACUP, AQM, FQ-CoDel, FIFO, RTT**

I. INTRODUCTION

With continued advances in online technology, as well as the introduction of services such as Netflix in Australia, average speed demands for a common broadband network are constantly on the rise. Depending on a household's typical usage patterns and traffic requirements, queue buffers on the edge of local networks can be a key factor in causing delays in a network. These delays in turn end up affecting all types of traffic regardless of their download and upload speed requirements, due to the congestion resulting from multiple services running simultaneously. In such cases, latency sensitive traffic in particular can have it's performance significantly enhanced through the use of Active Queue Management (AQM) systems implemented on network boundary devices, due to the way that they smartly handle and manage different types of traffic as opposed to traditional first in, first out (FIFO) queueing systems.

We have investigated the use of different AQM algorithms in a range of experimental household demographics, running testbed experiments that emulate traffic requirements during a households peak usage periods. These experiments directly compare and analyse how the network traffic operates, before and after an AQM is implemented in each situation. From this information, we have been able to make conclusions on how smarter queuing technologies can be used to reduce different user's speed demands, and hence determine the potential they have in reducing the load requirements on Australia's National Broadband Network (NBN). With this, we have been able to make an assessment on whether the current NBN speed tiers are appropriate for all user's with and without AQM running on the gateway.

The report is structured as follows. Section II will look into the background of why AQM is needed, as well as explore how different AQM algorithms can work to reduce network latency, and how TEACUP can be used to emulate AQM in a testbed. Section III shows the creation of six different demographics that represent commonly occurring households in Australia that we have used to run tests on. Section IV shows the experimental testbed that we used, and also the configuration file parameters we chose to use in the experiments. Section V shows the results from our experimentation along with a discussion and analysis of what they represent. Section VI explores how these results apply in a real world context and finally, Section VII concludes these results and discussions, with an additional Appendix at the end that includes results that were less crucial to the report.

---

[*] This work was originally developed as part of these author's BEng Final Year Research Project, 2016

II. BACKGROUND

A. Apparent Need for Speed

There is a common misconception in the general public that the only way to fix high latency/Round Trip Times (RTTs) in a network is to increase the speed or bandwidth levels. In reality, many applications such as online games and VoIP calls only accept small latency alterations to function optimally, however their actual speed requirements are very low. Similarly, some applications such as file transfers may have elastic bandwidth requirements, with user's being able to set up a download then perform other tasks while it loads, quick completion times may be of little importance to the user. Although high bandwidth levels are unnecessary for such applications, their performance can be significantly diminished by a full queue buffer, which can cause large delay times to a network. Higher speeds will reduce a networks average latency simply by pushing packets through the queue faster, however with smarter queue management, user's can also achieve the same effects without having to opt for higher costing speed plans. However, for applications that require high bandwidth such as video streams like Netflix, better queue management will offer little difference as such services are capped at their speed requirements.

In 2001, the Internet Telecommunications Union (ITU) [1] published a report that estimated that latency tolerance that user's have for different typically used network applications, which has been used as a basis for forming Table I. From Table I, we can see that applications are split into four separate latency classes; Interactive, Responsive, Timely and Non-critical based on their time to completion requirements for user's. Skype and other interactive applications such as gaming have very low latency tolerance, as they are performed in "real-time" by the user, with little tolerance for latency. Responsive applications such as web browsing and e-mail are shown to have slightly higher latency tolerance, as user's will still want low completion times, but applications are not real-time, and therefore tolerate some delay. Streaming and messaging applications fall under the timely latency category, with slightly more lenient completion time requirements than the responsive applications. Finally, applications such as bulk uploads and downloads have very elastic completion time requirements, as such applications can run in the background whilst user's perform other tasks. These values were used as a reference point for a user's latency tolerance in our analysis.

Type of Traffic	Latency Category	Latency Tolerance
Voice or Video (Skype, VoIP)	Interactive	0 - 150 ms
Interactive apps (gaming)	Interactive	0 - 200 ms
Online transactions (Email, web-browsing)	Responsive	1-2 s
Streaming (Netflix, Youtube)	Timely	5-10 s
Message services (WhatsApp, SMS)	Timely	5-10 s
Bulk downloads (App updates, podcast)	Non-critical	>10 s
Cloud/data uploads	Non-critical	>10 s

Table I
COMMON INTERNET TRAFFIC SEPARATED INTO LATENCY CATEGORIES [1]

B. Transmission Control Protocol

Transmission Control Protocol (TCP) is a transfer protocol that accounts for a large amount of internet traffic. TCP is considered a "reliable" protocol due to the way that it sets up a session between sender and receiver before a transfer begins (connection-oriented), and requires "ACKnowledgement" messages to be seen by the receiver for each chunk of data sent in order for the connection to remain open [2].

TCP has a congestion window (cwnd) that determines how much throughput of a link that the TCP session will consume. When a TCP link is established, the initial cwnd is set to a small multiple of the Maximum Segment Size (MSS) on the link. TCP connections begin in the "slow-start" (SS) phase. In SS, each time a set of packets is acknowledged by the receiver, the transmitter will increase the cwnd size by one MSS for every packet ACK it receives, essentially doubling the size of the cwnd after each round trip, and consequently doubling TCP's bandwidth requirements on the link. This will continue until TCP experiences congestion on the link, usually in the form of packet loss, in which case it will halve the size of it's cwnd. At this stage TCP enters the "congestion avoidance" (CA) phase, where it instead increases it's cwnd by one MSS for every RTT until it experiences congestion, in which case it halves. TCP will continue in CA until the TCP connection is terminated. It should be noted that the halving of the congestion window size is a characteristic of the NewReno TCP standard, with other standards such as Linux's CUBIC (which reduces cwnd by 30% on packet loss) performing slightly differently.

Another method for telling TCP to back-off is the Explicit Congestion Notification (ECN) flag, which is a softer method that applications can use to reduce TCP congestion. When the ECN flag is set to '1', the TCP flow is essentially told that the link is experiencing congestion without a packet being dropped. This results in the TCP flow acting the same way as if a packet were dropped, halving it's window size and hence the bandwidth that it takes up. TCP's nature results in it taking up the full available capacity of a link for data transfers. It is important to understand TCP's nature when analysing AQM, as TCP can be problematic especially when used with large router buffers, which is explained in more detail in the next section.

Although small (approximately 40 Bytes), TCP ACK messages create a minimum upload speed requirement for primarily downstream connections such as Netflix and movie downloads. As an example we can look at a TCP download that is transferring data at 12Mbps. Using a typical MTU of 1500 Bytes, we can see that the number of packets that are being downloaded per second is: 12000000/(8*1500) = 1000 packets per second. As an ACK message will be typically required for every second data packet received, the movie download will create (40*8*1000)/2 = 160 Kbps worth of ACK upload traffic.

C. Active Queue Management

As mentioned in Section I, network border devices contain a packet buffer. A buffers role is to absorb packets into a queue in the event of traffic bursts across the network [3]. When a router's buffer fills up, it can cause incoming packets to be lost, and in turn, congestion through the network. Without AQM, queues commonly use a simple 'FIFO' method to deal with network traffic. This method involves sending data out in the order that it enters the queue, and when the queue is full, dropping all incoming data. With no prioritization or queue maintenance to prevent it from filling up, this can cause potential problems with congestion on a router.

One solution to fix full router buffers is to increase their size. With more space to store more traffic, this can make it less likely that the buffer will fill up and begin to drop traffic. One major issue with this method is that the larger the queue gets, the larger RTTs that are added by the full buffer become, this is referred to as 'bufferbloat' [3]. One of the main contributors to bufferbloat is TCP, which as explained earlier will fill up the larger queues regardless of their size.

AQM algorithms can be a useful tool with which to reduce a router's queueing delay. In order to achieve this, AQM's actively monitor a router's average queue size, and when the queue starts to fill up, AQM can impose several different packet marking or dropping algorithms to make TCP senders back off early. This in turn reduces the router's average queue size and consequently, the RTTs experienced in the network [4]. One drawback of AQM is that it can negatively effect throughput speeds on a TCP link, especially if the link has a high RTT_{base} (the time it takes for a packet to reach it's destination and return based on the physical properties of a link). This is due to the way that AQM algorithms tell TCP to back off early, causing TCP to spend less time at it's optimal congestion window size, and thus throughput levels. This can be a particularly large issue on link's with a high RTT_{base}, where TCP will take significantly longer to reach it's the maximum window size, as the delay between sent packets, and received ACK's will be high. This will result in file transfers having longer completion times, the extent of which has been analysed in Section V.

Although AQM presents great potential for network improvements, it is still a relatively new technology and therefore is not widely utilized within many networks. Studies are currently being performed in order to discover the most effective AQM algorithms to use and how they work in practical environments, with companies such as Cablelabs, Cisco, and IETF providing advancements in various AQM technologies [5]. Universities, including Swinburne are also researching AQM's in order to highlight their importance in the IT industry [6]. Although this is the case, one area of study and implementation of AQM that we wanted to explore was it's ability to reduce a user's actual speed demands on the NBN.

We have added to this research by using the acceptable latency tolerances for user's shown in Table I, with a particular focus on online games as latency here is most commonly driving the idea that a user needs more speed, and hence we have explored how AQM can reduce speed requirements on the NBN after being implemented, whilst still keeping RTTs below these levels. We have investigated a handful of the many different AQM techniques available, eventually selecting the most effective algorithm to be used in our experiments for comparison against FIFO queueing. The following modern AQM systems have been examined:

1. Proportional Integral controller Enhanced (PIE) – Drops packets from the back of the queue based on a probability that is calculated based on latency [7]. This calculation takes latency samples over time to discover whether the overall latency in the queue is increasing or decreasing. If the latency is increasing the drop probability is increased, and if it is decreasing it is decreased. PIE can tolerate traffic bursts within the network by allowing a queue to be filled over it's target delay for small periods of time.

2. Controlled Delay Management (CoDel) – Drops packets that have been in the queue too long. CoDel time stamps each packet that enters the queue to calculate a packets local delay (sojourn time), with which it compares to a preset target delay value [8]. If the sojourn time is lower than the target delay the packet is forwarded normally, however if it is higher, then a time drop (Td) bit will be set for the next incoming packet. If an incoming packet that has the Td bit set again has a sojourn time that is higher than the threshold, the packet is then dropped. In a similar fashion to PIE, CoDel can also support burst tolerance. This process is shown in Figure 1.

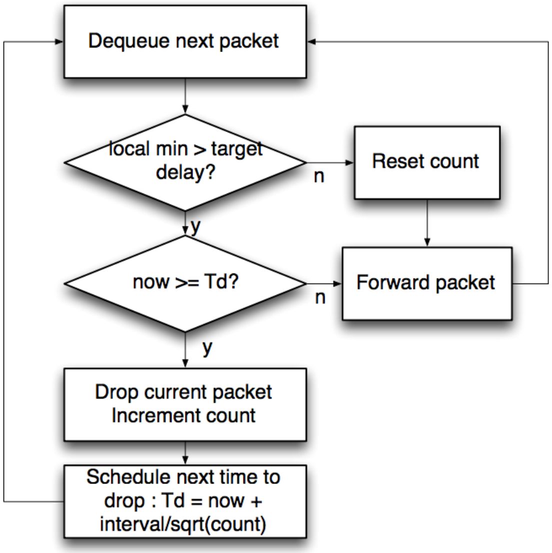

Figure 1. CoDel Algorithm [5]

3. Flow Queuing CoDel (FQ-CoDel) – An improvement to the CoDel algorithm is the added benefit of a Deficit Round Robin (DRR) queueing algorithm, creating separate queues for different types of traffic, for which CoDel is then applied on. Different queues are created by hashing a packets source IP, destination IP, port numbers, IP protocol number, and a randomly generated queue number assigned when the hashing commences [3].

D. TEACUP

TCP Experiment Automation Controlled Using Python (TEACUP), is a software that has been developed as a useful tool to automate TCP experiments in a controlled testbed by CAIA Swinburne [9]–[12]. TEACUP can emulate a variety of different network conditions and parameters such as bottleneck rate limits, packet loss rates, and different queuing algorithms. TEACUP uses a configuration file written in Python in order to emulate given network conditions, and capture an array of data including tcpdump, log files and graphed test results.

With TEACUP, it is possible to recreate network traffic that matches our case studies bandwidth requirements using applications such as DASH, iPerf, NTTCP, and Pktgen. We have been able to create a testbed network with an emulated packet queue on a router, that has captured network traffic at the bandwidth tiers of NBN using FIFO, CoDel, FQ-CoDel, and PIE. The collection and analysis of this data shows how effective AQM can be at reducing the average RTT in a network.

Using TEACUP as the backbone for the collection and running of our experiments, the following traffic generators have been utilized within TEACUP to emulate bursty traffic such as FTP downloads, interactive traffic like browsing the web, latency sensitive traffic such as online gaming, as well as any traffic that doesn't occur in real time such as Email:

1) iPerf - Can be used in TEACUP to emulate both TCP and UDP traffic at different required bandwidths over a given amount of time. We can use iPerf within TEACUP to emulate TCP or UDP traffic flows for both the server to client (download) and the client to server (upload) at given speeds for each demographic. In doing this, iPerf has been used for the majority of traffic emulations within our project testbed [13].
2) DASH - Dynamic Adaptive Streaming over HTTP (DASH) can be used to emulate our video streaming services through the network. DASH uses a HTTP server that contains both a Media Presentation Description (MPD), as well as the actual chunks of data to be streamed. DASH stores chunks of data at multiple quality levels, and can

dynamically vary the quality of chunks that it provides to it's client's through information provided by the MPD file. This file contains information about the chunk segments stored, their representation rates, and how to stream them based on a client's bandwidth capabilities [14]. TEACUP provides the capability to emulate a DASH server for a client to stream from. It does this by first, starting up a HTTP server on one of the testbed computers to emulate it as a server, and then transferring bursts of data to the second "client" computer, through port 80, at a given cycle interval in order to emulate each DASH chunk. For this reports a two second chunk size was chosen, using the BigBuckBunny dataset.

3) NTTCP – Can be used in TEACUP to specifically emulate UDP VoIP flows between a server and it's client. NTTCP is configured very similar to how iPerf is setup, and is useful for the emulation of VoIP traffic to observe how it is affected by AQM, which is expected to be quite significant due to this traffic being latency sensitive.
4) Pktgen - This tool can be used in TEACUP in order to generate UDP game traffic based on synthetically captured traffic from a real First Person Shooter (FPS) Games such as Quake 3. In TEACUP, one host is essentially acting as a game client, with the other being a server. Game traffic is bidirectional as the game state needs to receive both updates and inform a server of the current state, so it effects both down and up directions, with an additional latency requirement.

III. DEMOGRAPHICS

A. Australian Household Background

The average Australian household consists of 2.6 family members according to statistics recorded by the Australian Bureau of statistics in 2011 [15]. However, many families are either significantly larger or smaller than this, with many Australians living alone, and some larger families containing more than six people, we decided that a single case study size would not be able to be used to accurately make assumptions on how effective AQM is for all Australian homes. Furthermore, not all user's have the same internet requirements, as not everyone in a household will always use the internet at the same times, levels of intensity, or types of traffic. For these reasons, we have chosen to use data from the Australian Bureau of statistics and Telsyte in order to find some of the most commonly occurring speed requirement demographics in Australia, and create testbed experiments that emulate these speed requirements with and without AQM present on the gateway.

As mentioned, some useful information that we can use on Australian internet usage demographics may come from the research company: Telsyte. Telsyte has generated a report on a survey conducted to determine the average number of devices used in an Australian household, as well as the type of applications that are ran during peak times. Telsyte have used past data surveyed from 2013, as well as current data to predict the potential growth of various types of data usage by 2020. This information has been of a great assistance in focusing our research on specific usage patterns, as well as the amount of user's and devices running in typical households [16].

Currently, NBN Co's implementation of NBN in Australia has plans set to provide user's with up to 100 Mbps download, and up to 40 Mbps upload speeds [17], as shown in Table II:

NBN Speed Tiers	
Download Speed	Upload Speed
12 Mbps	1 Mbps
25 Mbps	5 Mbps
25 Mbps	10 Mbps
50 Mbps	20 Mbps
100 Mbps	40 Mbps

Table II
NBN DOWNLOAD AND UPLOD SPEED TIERS [17]

From Table II we can see that NBN Co plans to accommodate for a wide variety of user's requirements. From our research and implementation of AQM in our experiments, we have been able to find that for the majority of consumers, speeds of 100 Mbps will be excessive and unnecessary, and most consumers can stay at a substantially lower download and upload speed and still be able to consume the exact same content. This is explained in more detail in Section V.

B. Categories of speed and latency requirements

In addition to traffic usage trend information, we estimated typical network applications speed requirements using information found from the NBN Co website [18], as well as whistleOut.com [19]. Analysis during peak time was particularly important as this is when user's most notice whether or not the internet is running poorly, and hence develop the perceived idea of needing more speed. Table III shows the estimated download and upload speeds for various types of typical network traffic that we have used in order to estimate each households usage requirements for our experimentation. We can see that primarily downstream TCP traffic has very small associated upload speeds. As mentioned in Section II.B, this is resultant from the minimum ACK message rate required in order to maintain each applications downstream packet rate. This is also the case for the download speeds associated with upstream TCP traffic such as cloud uploads.

Service	Download Speed (Kbps)	Upload speed (Kbps)	Consumption type
Netflix SD	3000	40	Fixed
Netflix HD	5000	65	Fixed
General web/Social Media	1500	20	Elastic
E-mail	300	300	Elastic
HD Movie Downloads	8000	105	Elastic
Cloud services	3000	3000	Elastic
App updates	3000	40	Elastic
Online FPS Gaming	25	25	Latency Sensitive
Phone (VoIP) (one concurrent call)	100	100	Latency Sensitive

Table III
EXPERIMENTAL APPLICATIONS DOWNLOAD AND UPLOAD SPEEDS

From the survey conducted by Telsyte, we were able to identify the main applications that are being used simultaneously during peak periods, and exclude applications from our experimentation that we felt were not commonly being used such as 4K streaming. Additionally, it is interesting to note that most applications common usage percentages are increasing, especially the streaming of online videos. Exceptions to this include general web browsing traffic, which is notably the most detected occurrence, and VoIP calls, which is quite low most likely due to many other applications such as text messaging and social services replacing the need for VoIP systems.

From this data, we have defined three generic user templates that have been used to build up our demographic case studies. These templates represent the most common traffic requirements in Australia split into low, medium and high networking profiles shown in Tables IV, V and VI.

Application	Download (Kbps)	Upload (Kbps)
App Updates	3000	40
Cloud Services	3000	3000
General Web	1500	20
E-mail	300	300
Online FPS Gaming	25	25
	7825	3385

Table IV
LOW CONSUMPTION USER APPLICATIONS

For the low consumption user, we allocated applications that we felt matched what a general internet consumer would use in their peak usage periods. This could be updating their Operating Systems or phone/computer applications, as well as uploading pictures/backups to the cloud. Additionally, due to the majority of user's surveyed in the Telsyte survey using general web applications, we have allocated the use of general web/email applications to each of the three templates. Notably, most of these applications fall under the elastic traffic category, which is the traffic that AQM will be targeting in order to reduce the demographics speed requirements. Online FPS gaming traffic is also used for each template as it will best show how latency sensitive traffic can be affected by other traffic types on a gateway.

The moderate consumer, like the low consumer, uses all the basic functionality that most internet user's require, however has the additional requirement of using the Netflix streaming service in standard definition (SD). This was added because we felt that we needed to accommodate for the fact that although on the decline, SD Netflix still falls in at the fourth highest application used by user's simultaneously with other applications. Of additional note is the fact that Netflix has fixed speed requirements, so unfortunately for user's wishing to use

Application	Download (Kbps)	Upload (Kbps)
App Updates	3000	40
Cloud Services	3000	3000
General Web	1500	20
E-mail	300	300
Netflix SD	3000	40
Online FPS Gaming	25	25
	10825	3425

Table V
MODERATE CONSUMPTION USER APPLICATIONS

this application they must have at least 3 Mbps download and 48 Kbps upload speed, regardless of whether or not AQM is present.

Application	Download (Kbps)	Upload (Kbps)
HD Movie Downloads	8000	105
App Updates	3000	40
Cloud Services	3000	3000
General Web	1500	20
E-mail	300	300
Online FPS Gaming	25	25
Netflix HD	5000	40
	20825	3530

Table VI
HIGH CONSUMPTION USER APPLICATIONS

Our high consumption user essentially runs as many typical applications that we thought were realistically possible for a single internet user at any given period. This includes, in addition to our other templates concurrently running applications, bulk HD movie downloads and streaming Netflix in HD, making sure to include all services that Telsyte suggests are on the rise.

C. Demographics

From our three templates, we have formed a variety of different typical household demographics that we have run our experiments on. In particular, we focused on household sizes of one to four, as there is a relatively low percentage of household sizes that are bigger than this according to the ABS survey mentioned previously. The speeds shown for each demographic are rounded up to the nearest Mbps as rough upper bounds. The list below represents the six demographics we determined for our experiments:

1) Single user (high consumption): 21/4 Mbps down/up. We selected this demographic as one to focus on as according to ABS survey done in 2011, a single user household is the second most common household size. Additionally, the high consumption template was used as it has the most elastic traffic speed requirements, and thus will best represent a simple scenario that can show the effect that AQM has on a gateway.
2) Couple without children (two moderate users): 19/7 Mbps down/up. For this demographic we simply doubled all the upload and download requirements for a moderate user. However, only a single SD Netflix stream was used, as we felt that it is more realistic for a couple to stream only one instance of Netflix at a time.
3) One parent, one child (one high, one low): 29/7 Mbps down/up. This demographic is focused on the performance of various applications when one user is running significantly more services than the other. This scenario was used to show how user's could be mislead to consider upgrading to a higher speed tier in order to receive the better performance, and thus is good to illustrate the effectiveness of AQM.
4) Two parents, one child (two moderate, one low): 30/11 Mbps down/up. Similar to a couple without children, this scenario is built up of multiples of the user's defined in the Tables in Section B, however now emulates two SD Netflix streams. In addition to this, we have added a one channel VoIP call in order to realistically show how some family households still may require extra latency sensitive services.
5) Three person share house (two high, one moderate): 53/11 Mbps down/up. As this three person house is essentially built up of three independent people, we have selected the high consumption template to accommodate for each user running content separately. This means multiple HD Netflix streams, online gaming, as well as HD movie downloads. This demographic was useful in deter-

mining the validity of higher NBN speed tiers, as each of the high consumption user's may be content with longer download and upload completion times as long as their game traffic operates with low latency.

6) Two parents two kids (one high, two moderate, one low): 51/14 Mbps down/up. This demographic is focused on the last relatively common household size. In this household, essentially all realistic services are being run at the same time, with at least one instance of each type of typical service we have defined, as well as multiple general internet activities such as web browsing, and uploading files to the cloud. Similar to demographic 5, this scenario will be useful in identifying the full effectiveness of AQM, as it shows the typical tiers that higher usage households could be reduced to after AQM.

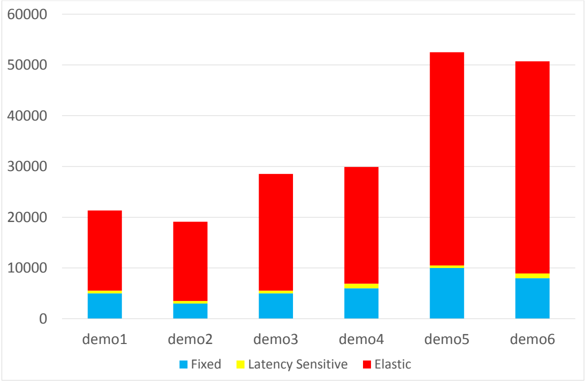

Figure 2. Traffic classifications download speed needs for each Demographic

From Figures 2 and 3, we can see that the majority of speed requirements for both download and upload is elastic, and therefore has elastic completion time requirements. This is the traffic that our AQM algorithms will focus on reducing the speed requirements for, making each demographics traffic requirements resemble speeds closer to their cumulative fixed and latency sensitive traffic speed requirements, whilst still keeping network RTTs down.

IV. METHODOLOGY

A. Testbed

The testbed we have used for this project consists of four separate physical 64 bit Acer Aspire laptop computers running FreeBSD 10.3, connected via 1 Gbps ethernet cabling by a switch, each with a unique role as represented by Figure 4.

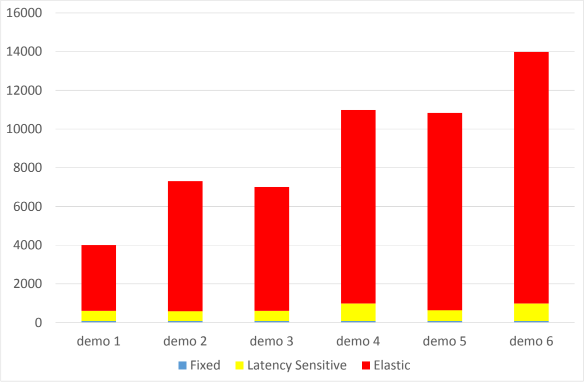

Figure 3. Traffic classifications upload speed needs for each Demographic

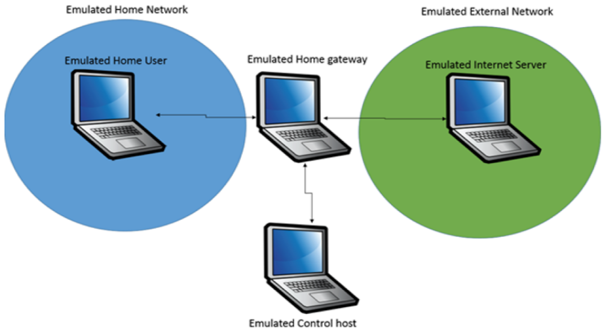

Figure 4. Testbed setup used for all TEACUP experiments

- Home User - The client, pulls data from the server through an emulated home gateway.
- Home Gateway - This sits in between the client and server, routing all packets that traverse the network. This is where the queues in each direction are present, and thus where the bottleneck was occurring, which is where we tested different AQM algorithms.
- Internet Server - Acts as a content server, using emulation tools such as iPerf and DASH (through TEACUP) to emulate the data transfer of different types of traffic for our case study.
- Control Host - Essentially does the initializing of the experiments, as well as collect all traffic that occurs for analysis.

- Switch - Model TL-SF1005D was used as a sufficiently capable switch that is fast enough to have negligible effects on the emulated bottlenecks observed in results.

B. Configuration

As mentioned earlier, all of the experiments for this report were performed using TEACUP, which requires a configuration file written in Python programming language, that is used to emulate test conditions. In order to create our experiments, we first had to come up with realistic values for RTT_{base} and our router's buffer size in number of packets, as well as decide which AQM algorithms to use, and upload and download speeds that best analyse whether or not the current NBN tiers are currently ideal for all user demographics.

For RTT_{base}, using ping statistics calculated from Wondernetwork [20], we discovered that end-to-end pings across Australia have a maximum RTT value of 60ms (Melbourne-Perth), however the largest percentage of traffic travels between Australia's two largest cities in Melbourne and Sydney, which have a RTT of between 10-20 ms depending on current network conditions. For this reason we have used a RTT_{base} value of 20 ms for the majority of our experiments. This will accurately emulate the most commonly occurring traffic that Australia would see in an NBN enabled scenario. Additionally, in order to test the effect that AQM has on throughput levels in networks with higher RTT_{base}, we have tested the throughput on a 12/1 Mbps down/up link with a RTT_{base} of 20 ms (Melbourne-Sydney), in comparison to 150 ms (Melbourne-Western United States), and 250 ms (Melbourne-Europe).

Regarding buffer size, studies suggest that the ideal size for a network buffer is equal to a link's Bandwidth Delay Product (BDP), which can be calculated by multiplying the bandwidth of a link by the delay across it. For FIFO, in order to analyse network extremities, we chose to use a bandwidth of the highest NBN tier (100 Mbps) across a link from Melbourne to Perth (60 ms), which results in a router buffer size of 500 packets. Although we are using a different RTT for this calculation than those mentioned previously, this value will result in a buffer size that NBN Co would more realistically use for home gateways they provide for the NBN all across Australia.

For AQM, the recommended buffer size for a CoDel or FQ-CoDel buffer is 1000, which is what we used for our AQM experiment. Although the FIFO and AQM buffer sizes are different, this will not change the conclusions that we can make on the effects that AQM is having on a user's latency, as the larger buffer size should have a negative effect on RTTs in a common network. By effectively doubling the buffer size, it will also further demonstrate how much more effective AQM is at managing queuing delays regardless of how big it's buffer size gets.

When it came to determining our bandwidth speeds we would run experiments on, we wanted to make sure that we explored a full range of different download and upload speeds, that encompassed NBN Co's offered speeds. We started from 8/1 Mbps, and went up to 100/40 Mbps. The download speeds initially went from 8 to 12, 16 then 20, before just increasing by five until 100 Mbps was reached. The upload speeds on the other hand started at 1 Mbps at lower download speeds, before slowly increasing as the download speeds increased. For every download speed there was also a 1 Mbps version, to see the effect that a congested link for a high download with only a 1 Mbps upstream would have. For the five NBN speed tiers shown in Table II, multiple other upload speeds were chosen for each, to clearly contract the low and high upload speeds for the same connection. A full list of speeds that we have run experiments on is included in the Appendix C code file, pages 30 to 31.

Tests were initially run on each modern AQM algorithm (PIE, CoDel, FQ-CoDel) in order to discover the algorithm that proved to be most effective. After these tests were run, we found that FQ-CoDel consistently performed better than the other algorithms, and thus chose this algorithm for most of our testing. This is shown in detail in Section V. Configuration files that were used to run experiments are included in Appendix C.

Tables VII to XIII show the traffic that was created for each demographic, along with each applications start times and running duration that represent each households peak usage patterns over a 60 second period. In addition to this, they illustrate the TEACUP tools that were used for each traffic type, a complete example of the configuration file for demographic 1 is shown in Appendix C. In terms of congestion control algorithm, the default parameter for TEACUP configuration files of NewReno was used, as that was effective enough to make sure our other configurations applied correctly.

Application	Generator	Start Time (s)	Duration (s)
HD Movie	iPerf TCP down	1	60
App Updates	iPerf TCP down	5	20
Web	iPerf TCP down	5, 20, 35, 50	1
E-mail	iPerf TCP down	0,25,50	3
Game	Pktgen	15	45
Netflix (SD)	DASH	10	50

Table VII
DEMOGRAPHIC 0.5 (NO UPLOAD) TRAFFIC START TIMES AND DURATION

Application	Generator	Start Time (s)	Duration (s)
HD Movie	iPerf TCP down	1	60
App Updates	iPerf TCP down	5	20
Cloud Upload	iPerf TCP up	0	40
Web	iPerf TCP down	5, 20, 35, 50	1
E-mail	iPerf TCP down	0,25,50	3
Game	Pktgen	15	45
Netflix (SD)	DASH	10	50

Table VIII
DEMOGRAPHIC 1 TRAFFIC START TIMES AND DURATION

Application	Generator	Start Time (s)	Duration (s)
App Updates	iPerf TCP down	5,10	20
Cloud Upload	iPerf TCP up	0,10	40
Web	iPerf TCP down	5, 10, 20, 25, 35, 40, 50, 55	1
E-mail	iPerf TCP down	0, 5, 25, 30, 50, 55	3
Game	Pktgen	15	45
Netflix (SD)	DASH	10	50

Table IX
DEMOGRAPHIC 2 TRAFFIC START TIMES AND DURATION

Application	Generator	Start Time (s)	Duration (s)
HD Movie	iPerf TCP down	1	60
App Updates	iPerf TCP down	5,10	20
Cloud Upload	iPerf TCP up	0,10	40
Web	iPerf TCP down	5, 10, 20, 25, 35, 40, 50, 55	1
E-mail	iPerf TCP down	0, 5, 25, 30, 50, 55	3
Game	Pktgen	15	45
Netflix (HD)	DASH	10	50

Table X
DEMOGRAPHIC 3 TRAFFIC START TIMES AND DURATION

V. RESULTS

A. Traffic Types

Figures 5 to 14 are used to show how each of the different traffic types act seperately from one another on an uncongested link.

Figures 6 and 8 show that the iPerf TCP stream in each of the up and down directions takes up the full capacity of the 12/1 Mbps down/up link, with the link's throughput maxed out in each case. Figure 5 shows the single iPerf flow's RTT vs Time graph, where we can see that congestion is experienced at approximately 12 seconds into the flow, and therefore the congestion window and RTT are halved. Similarly, Figure 7 shows how the RTT of the flow slowly increases as the congestion window increases. However, the 1Mbps bottleneck and 500 packet buffer means RTT quickly exceeds one second, and the TCP connection never exits slow-start mode before our 60 second trial ends. This supports our analysis of the way TCP works in Section II.A.

As DASH uses TCP sessions to transfer media content from the server to the user, the traffic patterns are very similar to the regular iPerf TCP stream for both RTT and throughput. The extra spread of results for the throughput in Figure 10 is due to the different nature of how DASH is being run compared to iPerf in our experiments, with

Application	Generator	Start Time (s)	Duration (s)
App Updates	iPerf TCP down	3,5,10	20
Cloud Upload	iPerf TCP up	0,5,10	40
Web	iPerf TCP down	5, 10, 12, 20, 25, 27, 35, 40, 42, 50, 55, 57	1
E-mail	iPerf TCP down	0, 5, 7, 25, 30, 32, 50, 55, 57	3
Game	Pktgen	15	45
Netflix (SD)	DASH	10	50
VoIP	NTTCP	15	45

Table XI
DEMOGRAPHIC 4 TRAFFIC START TIMES AND DURATION

Application	Generator	Start Time (s)	Duration (s)
HD Movie	iPerf TCP down	0,10	60
App Updates	iPerf TCP down	3,5,10	20
Cloud Upload	iPerf TCP up	0,5,10	40
Web	iPerf TCP down	5, 10, 12, 20, 25, 27, 35, 40, 42, 50, 55, 57	1
E-mail	iPerf TCP down	0, 5, 7, 25, 30, 32, 50, 55, 57	3
Game	Pktgen	15	45
Netflix (HD)	DASH	10,15	50,45

Table XII
DEMOGRAPHIC 5 TRAFFIC START TIMES AND DURATION

the extra steps in starting a chrome window on the server, then connecting and streaming content from the client causing minor throughput variations. The segmenting of the flow in Figure 9 is due to the two second chunk sizes that we are using.

We can also see that the Pktgen and NTTCP sessions that are used to emulate game and VOIP traffic respectively are very similar, with very low throughput requirements (Game 17 Kbps VoIP 100 Kbps), the RTT in each case remains consistently low at around 20-22 ms shown in Figures 11 and 13, only slightly higher than the $\mathrm{RTT}_{\mathrm{base}}$ of 20 ms in each case. This accurately represents our earlier analysis of latency sensitive traffic requirements. As mentioned previously, game traffic has been used as the focus of the majority of tests done in the following experiments, as it will best show how

Application	Generator	Start Time (s)	Duration (s)
HD Movie	iPerf TCP down	0	60
App Updates	iPerf TCP down	3,5,10,12	20
Cloud Upload	iPerf TCP up	0,5,10,15	40
Web	iPerf TCP down	0, 5, 10, 12, 15, 20, 25, 27, 30, 35, 40, 42, 45, 50, 55, 57	1
E-mail	iPerf TCP down	0, 5, 7, 10, 25, 30, 35, 32, 50, 55, 57, 60	3
Game	Pktgen	15	45
Netflix (HD)	DASH	10,15	50,45
VoIP	NTTCP	15	45

Table XIII
DEMOGRAPHIC 6 TRAFFIC START TIMES AND DURATION

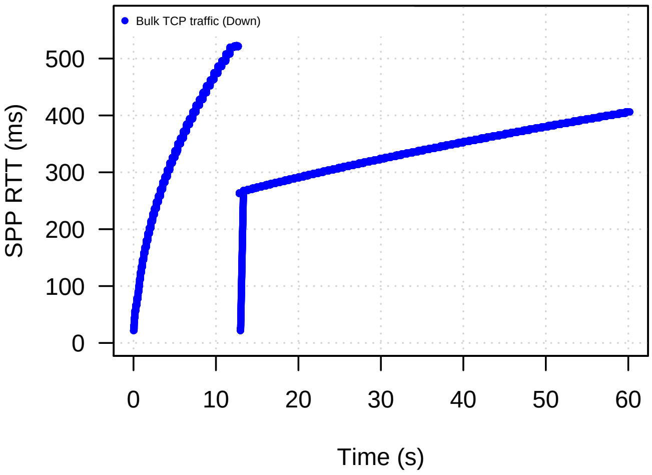

Figure 5. RTT vs Time for a single iPerf traffic representing a Bulk TCP download for 12/1 Mbps down/up, 20 ms $\mathrm{RTT}_{\mathrm{base}}$ and 500 pkt buffer FIFO queue

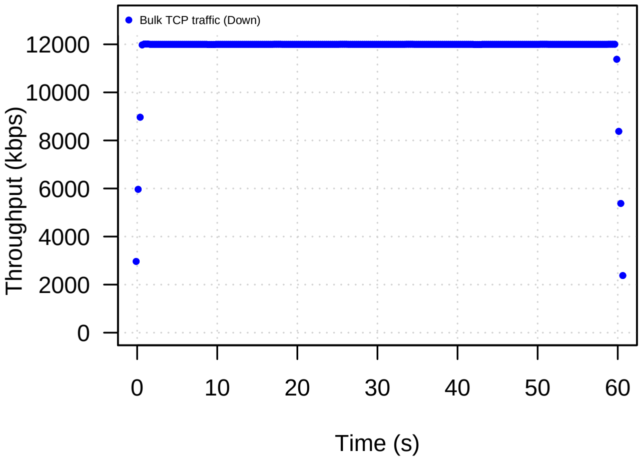

Figure 6. Throughput vs Time for a single iPerf traffic representing a Bulk TCP download for 12/1 Mbps down/up, 20 ms $\mathrm{RTT}_{\mathrm{base}}$ and 500 pkt buffer FIFO queue

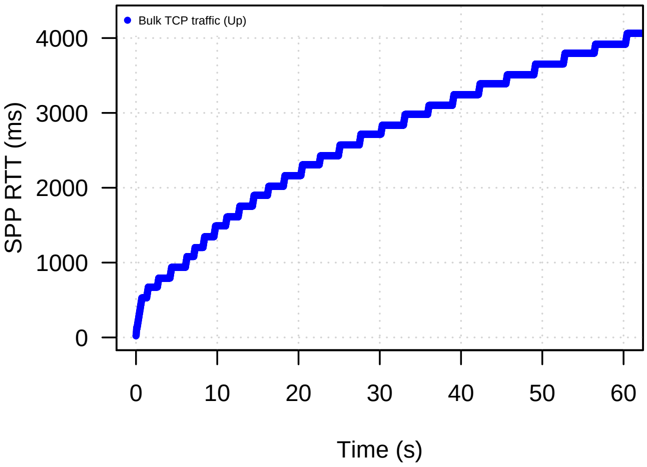

Figure 7. RTT vs Time for a single iPerf traffic representing a Bulk TCP upload for 12/1 Mbps down/up, 20 ms $\mathrm{RTT}_{\mathrm{base}}$ and 500 pkt buffer FIFO queue

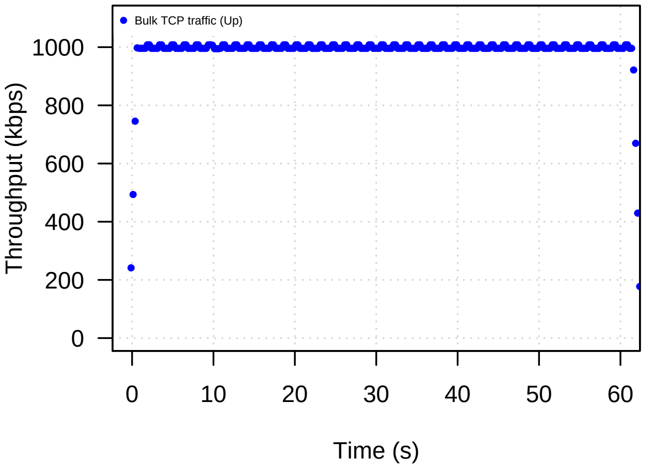

Figure 8. Throughput vs Time for a single iPerf traffic representing a Bulk TCP download for 12/1 Mbps down/up, 20 ms $\mathrm{RTT}_{\mathrm{base}}$ and 500 pkt buffer FIFO queue

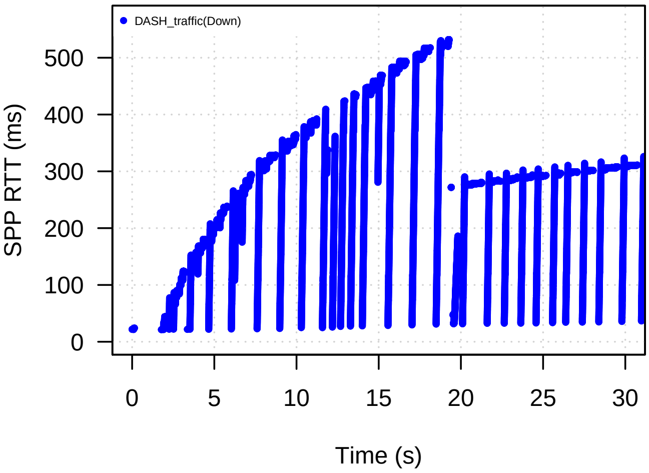

Figure 9. RTT vs Time for a single DASH traffic representing a Netflix stream for 12/1 Mbps down/up, 20 ms $\mathrm{RTT}_{\mathrm{base}}$ and 500 pkt buffer FIFO queue

latency sensitive traffic can be negatively effected by other applications causing congestion on a gateway.

B. High RTT with FIFO: flooded downstream

Figures 15 to 18 emulate the traffic requirements of demographic 1, however the cloud upload traffic, as well as the e-mail upload traffic has been removed creating demographic 0.5. This was done in order to show how traffic performs with traffic congestion in a single direction (down), in potentially non-peak usage periods.

From Figures 15 and 16 we can see that the added TCP download streams are causing a reasonable amount of congestion on the FIFO gateway, which is in turn causing the latency sensitive traffic to have increased RTTs. Figure 15 shows that with the added traffic, the game traffic now has an average RTT of around 300 ms, which is substantially up on the 20-22 ms previously observed without congestion. From Table I, we can see that this is not an acceptable latency level for game traffic (200 ms). Additionally, from Figure 16 we can also see the effects that the added traffic is having on throughput, as each of the DASH and movie download applications can no longer reach the full capacity of the 12 Mbps link whilst competing with the additional traffic on the gateway.

Figure 17 further shows how important upload speeds can be in FIFO networks. Here we can see the RTT vs

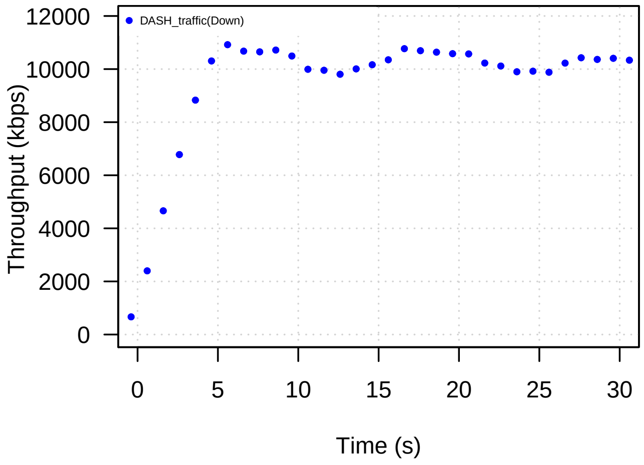

Figure 10. Throughput vs Time for a single DASH traffic representing a Netflix stream for 12/1 Mbps down/up, 20 ms RTT_{base} and 500 pkt buffer FIFO queue

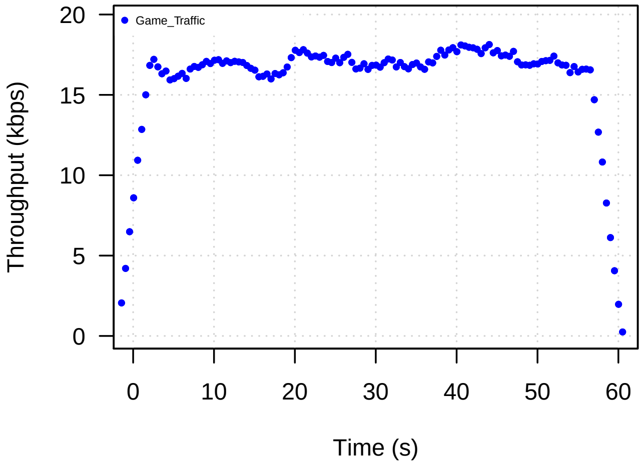

Figure 12. Throughput vs Time for a single Pktgen traffic representing a FPS Game for 12/1 Mbps down/up, 20 ms RTT_{base} and 500 pkt buffer FIFO queue

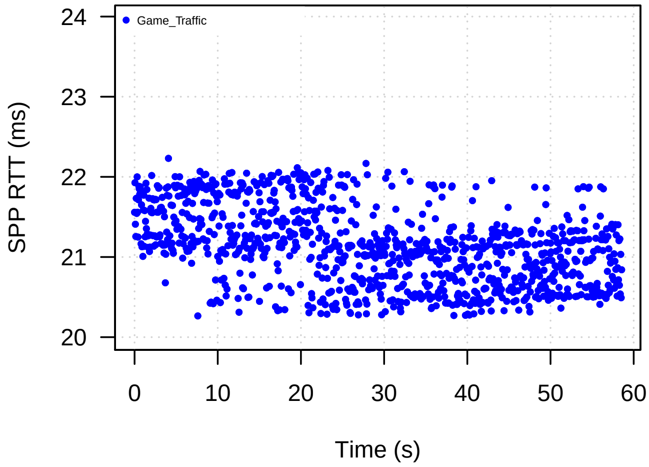

Figure 11. RTT vs Time for a single Pktgen traffic representing a FPS Game for 12/1 Mbps down/up, 20 ms RTT_{base} and 500 pkt buffer FIFO queue

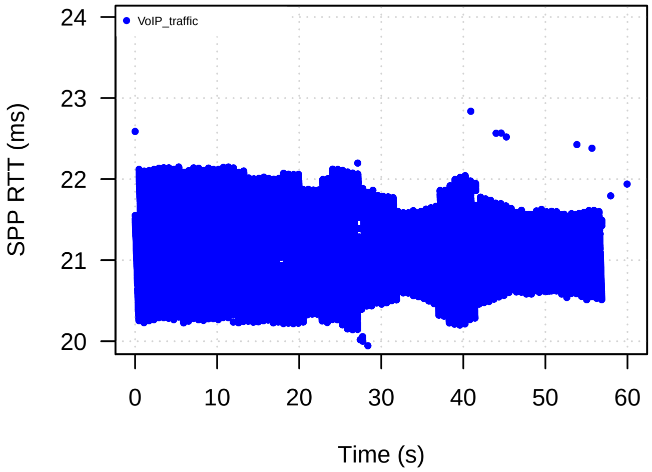

Figure 13. RTT vs Time for a single NTTCP traffic representing a VoIP call for 12/1 Mbps down/up, 20 ms RTT_{base} and 500 pkt buffer FIFO queue

Time graph with increasing download speeds at a fixed upload speed of 1 Mbps. As expected, the average RTT experienced by the game traffic is reduced as download speed increases, however at a download speed of 50 Mbps the average RTT stops reducing and instead starts to actually increase as the link speed increases. This is due to the upstream TCP ACK traffic from the concurrent downstream TCP flow. As explained in Section II.B, TCP is considered a reliable protocol due to the way that a TCP receiver is required to acknowledge every packet that it receives from a sender. These ACK packets are only small (40 Bytes) in the up direction, however as the download speed increases, a significantly high number of ACK packets is required for the greater number of packets that the client receives. This can be a problem with highly skewed asymmetric link's, which is what we are observing in Figure 17. As the download speed has reached a level where it is too much higher than the upload traffic (50 Mbps), we can observe that the ACK packets start to flood the upload link (1 Mbps), causing congestion and thus higher RTTs for the FPS game traffic observed.

Figure 18 shows a small example of how higher upload speeds can reduce the RTT experienced in the same scenario. Looking at speeds from 65 Mbps down to 75 Mbps down, where with a previous upload speed of 1 Mbps the RTTs slowly increases with higher speeds, we can now see that with a higher upload speed (4 Mbps),

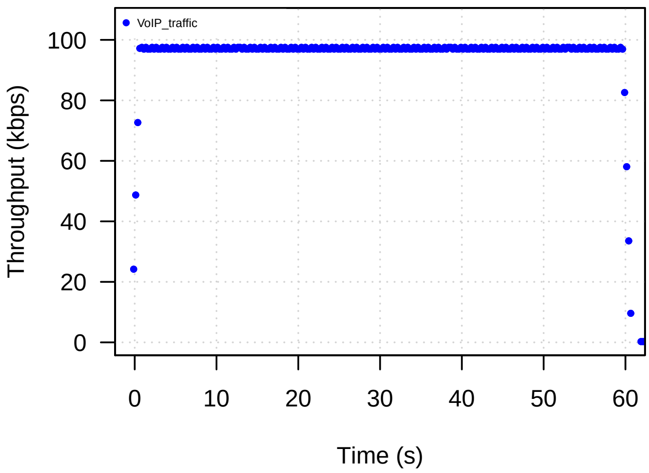

Figure 14. Throughput vs Time for a single NTTCP traffic representing a VoIP call for 12/1 Mbps down/up, 20 ms RTT_{base} and 500 pkt buffer FIFO queue

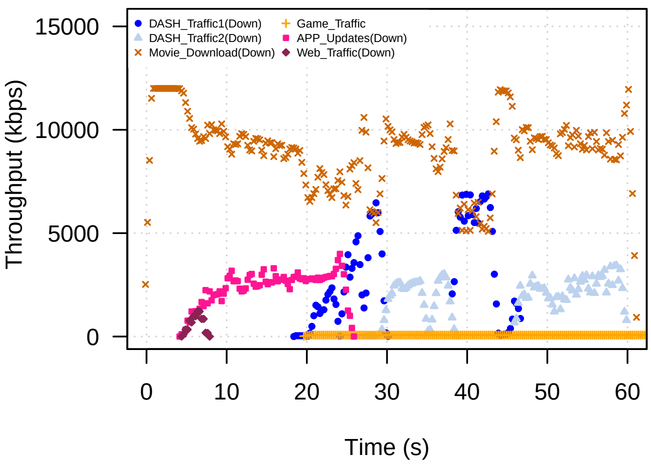

Figure 16. Throughput vs Time for Demo 0.5 traffic at 12/1 Mbps down/up, 20 ms RTT_{base} and 500 pkt buffer FIFO queue

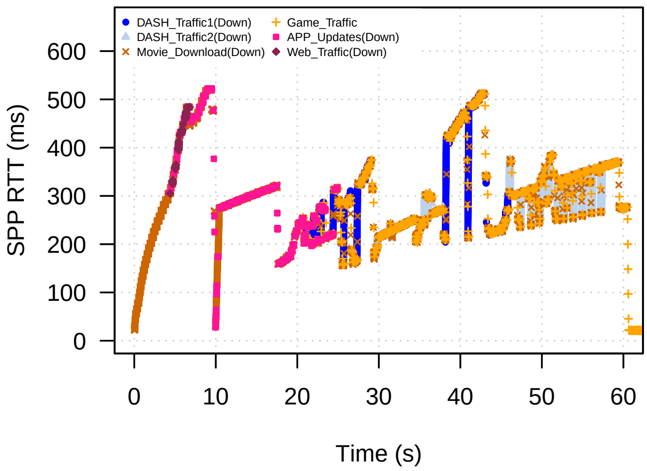

Figure 15. RTT vs Time for Demo 0.5 traffic at 12/1 Mbps down/up, 20 ms RTT_{base} and 500 pkt buffer FIFO queue

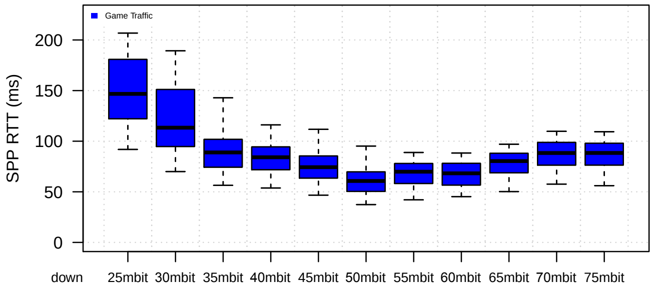

Figure 17. RTT Boxplot for Demo 0.5 FPS Game traffic at fixed 1 Mbps upload with changing download speed, 20 ms RTT_{base} and 500 pkt buffer FIFO queue

that can accommodate the download traffic ACK speed requirements, as expected the RTT now reduces as the download speed increases.

C. High RTT with FIFO: flooded downstream and upstream

Figures 19 to 23 show examples of how the traffic from demographic 1, which includes a bulk cloud upload as well as e-mail upload traffic on top of the traffic from demographic 0.5, thus showing how flooding both the downstream and the upstream can effect latency sensitive traffic.

From Figures 19 and 20, we can see that the single cloud upload has significant effects on the latency and throughput levels observed on the network. Figure 19 shows latency levels with an average of around 1000 ms, reaching as high as 2500 ms for each traffic type shown, which is significantly up on the latency levels shown with no upload in Figure 15, where the game traffic reached a maximum RTT of around 500 ms, and had an average of about 250 ms.

Additionally, Figure 20 shows the negative effects that the congestion is having on the throughput, with a congested uplink causing the TCP traffic to be unable to reach optimal throughput levels for the majority of the transmission, and much of the link's 12 Mbps capacity going unused. This is again due to the ACK traffic associated with the download traffic. TCP ACK messages need to travel un-hindered between client and server in order for the server to optimally increase it's TCP cwnd value, and thus it's throughput on the link at an ideal rate. This is not possible with a congested uplink, as the TCP ACK messages are competing with the bulk cloud upload traffic on the gateway. To confirm this, we can see that this is remedied shortly after the

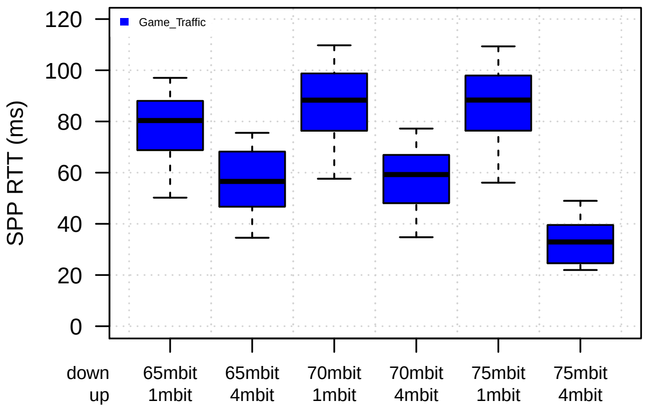

Figure 18. RTT Boxplot for Demo 0.5 FPS Game traffic with changing download/upload speed from 65/1 - 75/4 Mbps, 20 ms RTT_{base} and 500 pkt buffer FIFO queue

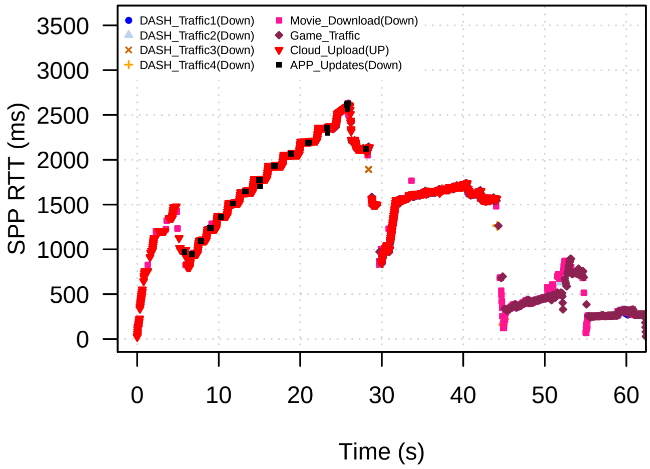

Figure 19. RTT vs Time for Demo 1 traffic at 12/1 Mbps down/up, 20 ms RTT_{base} and 500 pkt buffer FIFO queue

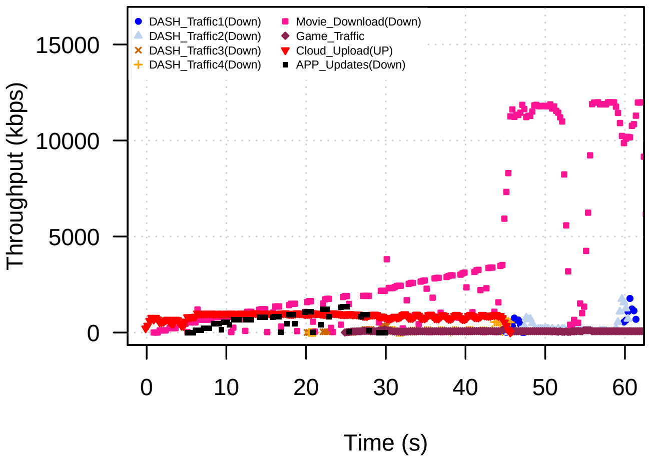

Figure 20. Throughput vs Time for Demo 1 traffic at 12/1 Mbps down/up, 20 ms RTT_{base} and 500 pkt buffer FIFO queue

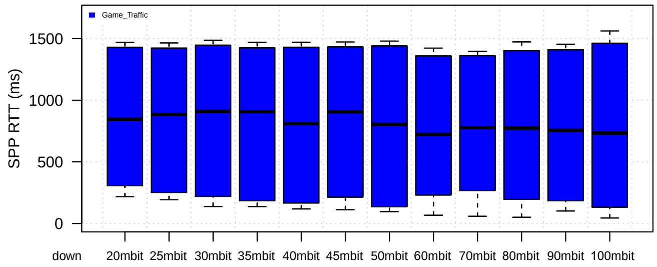

Figure 21. RTT Boxplot for Demo 1 FPS Game traffic at fixed 1 Mbps upload with changing download speed, 20 ms RTT_{base} and 500 pkt buffer FIFO queue

cloud upload finishes at 40 seconds into the graph, when the TCP ACK messages can be sent without congestion.

Figure 21 is an example of increasing the download speeds on a link for demographic 1, where a bulk cloud upload is significantly effecting the RTT throughout the entire graph. Although as explained earlier, higher download speeds should in theory cause lower RTTs for the game traffic, we can see that this is not the case for this example, with the median RTT not far from 1000 ms at each speed shown, which is not acceptable for latency sensitive traffic. This is due to the low upload speed of 1 Mbps used for this experiment, which is being filled by cloud upload TCP traffic. With the uplink full, TCP ACKs for the download traffic are being congested, thus causing the latency we are observing on the link. This shows how important upload speeds can be to a user, as in this case, no matter how high a user's download speeds are, they will still experience undesirable RTT levels for their latency sensitive traffic if cloud uploads are performed simultaneously.

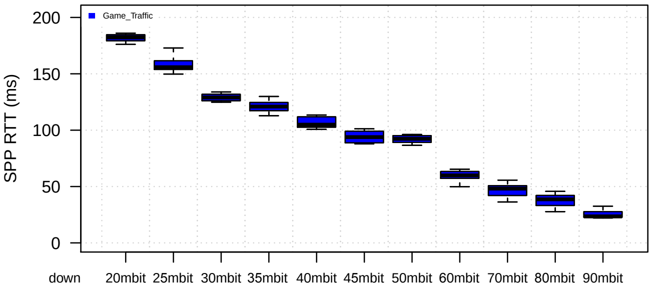

Figure 22. RTT Boxplot for Demo 1 FPS Game traffic in the period 45-60s, at fixed 1 Mbps upload with changing download speed, 20 ms RTT_{base} and 500 pkt buffer FIFO queue

Figure 22 shows an example of increasing the download speeds with a fixed upload speed similarly to Figure 29, however highlights the period of 45-60 seconds where the cloud upload is finished. Now that there is no TCP traffic flooding the uplink, we can see that as expected, the higher download consistently cause lower RTTs for the game traffic on the link.

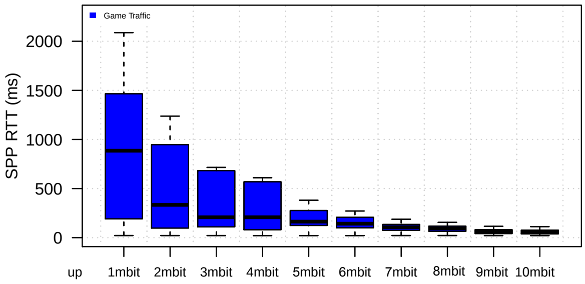

Figure 23. RTT Boxplot for Demo 1 FPS Game traffic at fixed 25 Mbps download with changing upload speed, 20 ms RTT$_{\text{base}}$ and 500 pkt buffer FIFO queue

We can see from Figure 23 that at a fixed download speed of 25 Mbps, as each upload speed increases, the RTT is reduced, with traffic reaching acceptable RTT values (under 200 ms) at 7 Mbps up. This is because with both a relatively high download and upload speed, neither the downlink or uplink are congested as severely as in previous examples, allowing higher speeds to have positive effects on network RTTs.

D. Low RTT with AQM: flooded downstream and upstream

Figures 24 to 26 aim to compare the effectiveness that CoDel, PIE and FQ-CoDel have in the reduction of network RTTs, using the first NBN tier as an example. This was done in order to select the most effective AQM algorithm to be used in the remainder of experiments with which to test against FIFO.

As we can see from Figures 24 (PIE), 25 (CoDel), and 26 (FQ-CoDel), each algorithm has significantly reduced the RTTs experienced by the game traffic on the link when compared to results shown for FIFO queuing in Figure 19, with each algorithm reducing the average RTT experienced to below 100 ms, as apposed to FIFO shown in Figure 19, which had an average RTT of over 1000 ms. Although this is the case, there are some definite differences between the results shown for each algorithm. We can see from Figure 24 that PIE has the highest average RTT for all traffic of the three algorithms at around 90 ms, and game traffic averaging approximately 60 ms. This is likely due to the fact that PIE uses a random drop method based on a latency calculation.

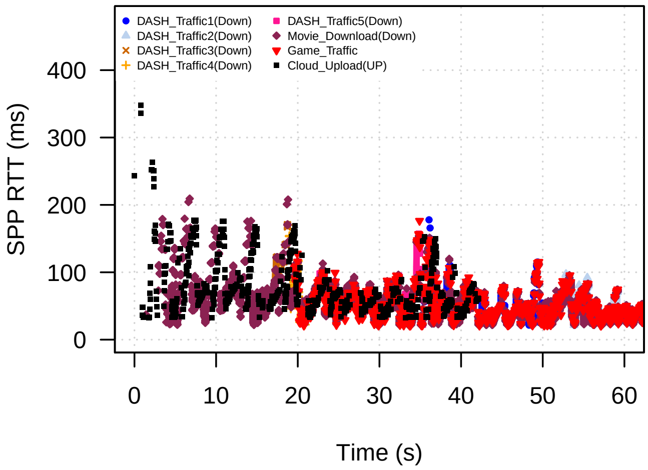

Figure 24. RTT vs Time for Demo 1 traffic at 12/1 Mbps down/up, 20 ms RTT$_{\text{base}}$ and 1000 pkt buffer PIE queue

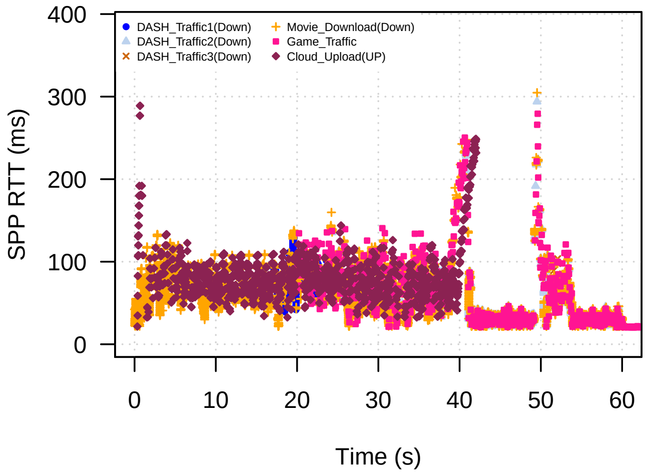

Figure 25. RTT vs Time for Demo 1 traffic at 12/1 Mbps down/up, 20 ms RTT$_{\text{base}}$ and 1000 pkt buffer CoDel queue

Figure 25 shows that CoDel is a close second with an average RTT of approximately 80 ms for all traffic, and similar game traffic average of about 60 ms. This makes sense, as unlike PIE, CoDel does not drop packets based on a random probability. Finally, from Figure 26 we can see that FQ-CoDel is the most effectively performing algorithm, with the lowest average RTT at around 40 ms for most of the traffic, with only the upload traffic reaching RTT levels that resemble the other algorithms RTTs, and the game traffic staying at approximately 20-30 ms for the majority of the experiment. In addition to this, FQ-CoDel also has a significantly lower spread than the other two algorithms for the majority of the

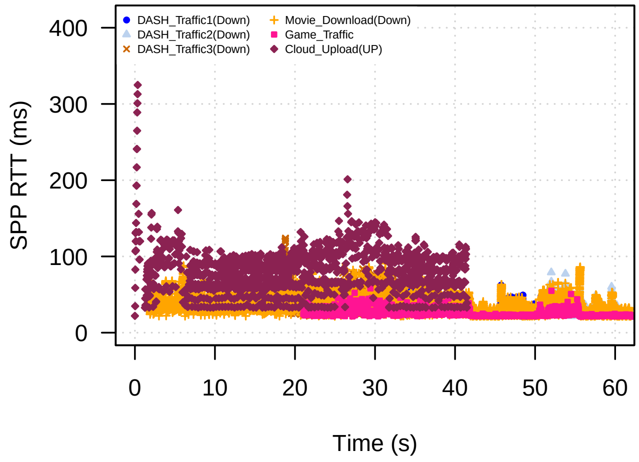

Figure 26. RTT vs Time for Demo 1 traffic at 12/1 Mbps down/up, 20 ms RTT_{base} and 1000 pkt buffer FQ-CoDel queue

traffic. This is due to FQ-CoDel adding FlowQueue scheduling and flow isolation to the basic CoDel AQM. By separating different flows (and returning ACKs) into separately scheduled queues, FQ-CoDel isolates the ACKs and game packets from competing TCP Data packets. Consequently, game traffic experiences far lower queuing delays and ACK traffic receives a sufficient share of bottleneck capacity. Additional graphs showing the impact for AQM over a range of upload speeds are shown in Appendix A.

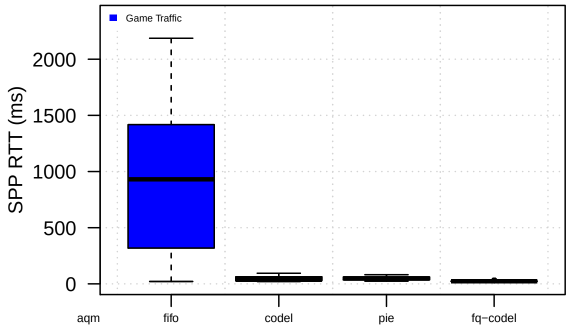

Figure 27. RTT Boxplot comparing for Demo 1 FPS Game traffic for FIFO and other AQM at fixed 12/1 Mbps down/up speed with a 20 ms RTT_{base}

Figure 27 shows a side-by-side Boxplot comparison of FIFO against the three algorithms shown previously for the first NBN tier of 12/1 Mbps down/up. Here we can more easily compare FIFO to AQM, with FIFO's median of approximately 1000 ms better standing out against the other three algorithms, each with median RTTs of below 100 ms.

Figure 28 compares the game traffics RTTs for FIFO against the three AQM algorithms at each of the four remaining NBN speed tiers. Similar to Figure 27, we can see that CoDel and PIE produce similar results to each other, with significant RTT reductions shown in comparison to FIFO at each of the speed tiers, and higher speeds causing lower RTTs for each of the two algorithms. Conversely, FQ-CoDel bottoms out at a consistent RTT of around 25 ms at each speed. This lower than the other algorithms and only slightly higher than the RTT_{base} of 20 ms, even when the gateway is being flooded by multiple TCP traffic streams, again showing that FQ-CoDel is superior to the other two algorithms. For these reasons we chose to use FQ-CoDel as the most suitable algorithm to compare FIFO tests against for the rest of our experiments, as it would be the most likely algorithm chosen for real life applications. Additionally, DOCSIS 3.0 supports FQ-CoDel as an optional AQM to be enabled on it's routing appliances.

Figure 29 compares FQ-CoDel to the FIFO experiment we previously observed in Figure 21, highlighting the effect that increasing the download speed has on RTTs experienced by game traffic with FQ-CoDel on the gateway. Here we can see that unlike FIFO, where the RTT consistently remains around 1000 ms, the cloud upload traffic has little effect on the RTTs observed by FQ-CoDel, as it's pre-emptive packet dropping method does not allow the traffic to congest the uplink. Instead, the RTTs for FQ-CoDel remain constant throughout the graph, consistently staying only slightly higher than the RTT_{base} of 20 ms.

Figure 30 similarly compares FQ-CoDel to the FIFO experiment previously shown in Figure 23, instead looking at the effect that increasing the upload speed has on the RTTs experienced by game traffic with FQ-CoDel on the gateway. Although the increased upload speeds have a positive effect on the RTTs experienced by game traffic with FIFO, we can see that with FQ-CoDel these higher speeds are not necessary, as again the RTT remains only slightly above the RTT_{base} for the entire graph.

E. Demographic Comparisons

This section explores the application of FQ-CoDel in comparison to FIFO for the all six demographics detailed in Section III.C. Median RTT values for FPS game traffic were taken and graphed for demographics 1-6. This was done for a full range of bandwidth speeds up to the highest NBN speed tier (100/40 Mbps down/up).

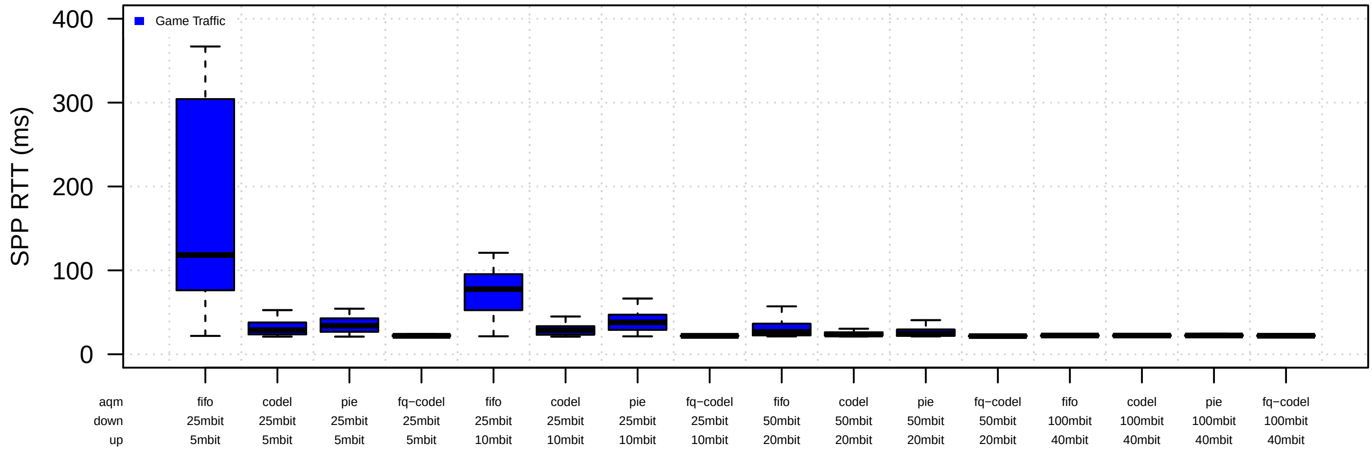

Figure 28. RTT Boxplot comparing for Demo 1 FPS Game traffic for FIFO and other AQM at NBN down/up speed tiers, with a 20 ms RTT_{base}

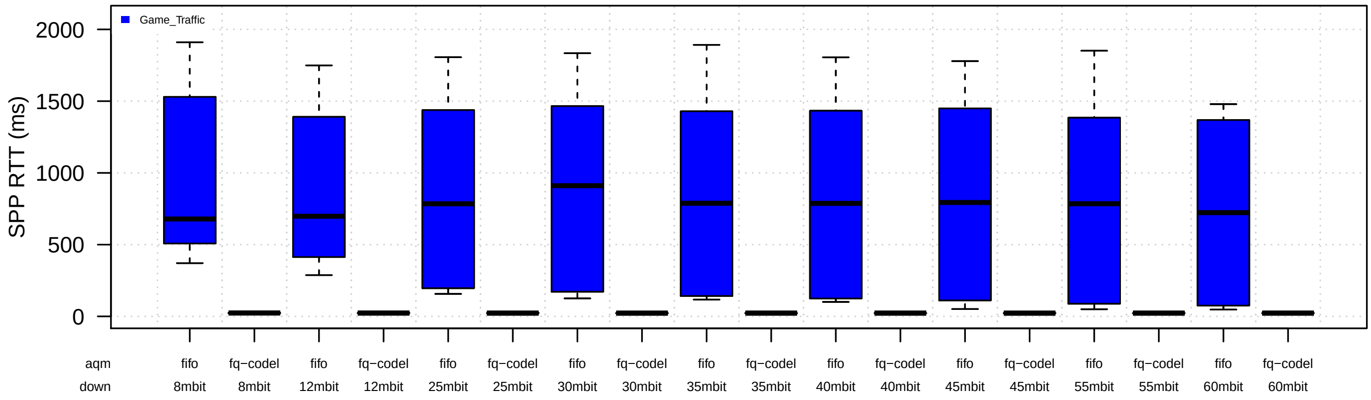

Figure 29. RTT Boxplot comparing for Demo 1 FPS Game traffic for FIFO & FQ-CoDel at fixed 1 Mbps upload with changing download speed, and a 20 ms RTT_{base}

Boxplots that these medians were derived from are included in Appendix B, and a full list of these speeds can also be seen in the configuration file included in Appendix C.

From Figures 31 and 32 we can see the clear correlation's that increasing speeds have with each of the six demographics. From Figure 31, we can see that all six demographics show unacceptable median RTT levels for the game traffic at various stages of the graph. Despite the very different traffic requirements between demographics, each graph follows the same RTT patterns, with the RTT for the game traffic dropping off as both the download and upload speeds gradually increase throughout the graph. This shows that even though our later demographics have higher download and upload application requirements, this has little effect on the median RTT experienced by latency sensitive game traffic on the link.

Consequently we can apply our conclusions from observations to all Australian household sizes, even those that are not shown in our demographics tests. Similar to other results from Section V, we can see that both download and upload speeds are clearly showing improvements in median game traffic RTTs as they are increased for each demographic. We can see that due to the smaller upload speeds in comparison to the download speeds, higher bandwidths on the uplink have a particularly strong effect on RTT reduction, with each demographic except for demographic 1 (which has smaller traffic levels) requiring at least 5 Mbps upload speeds in order to reduce the game traffic latency below an acceptable 200 ms at each stage of the graph.

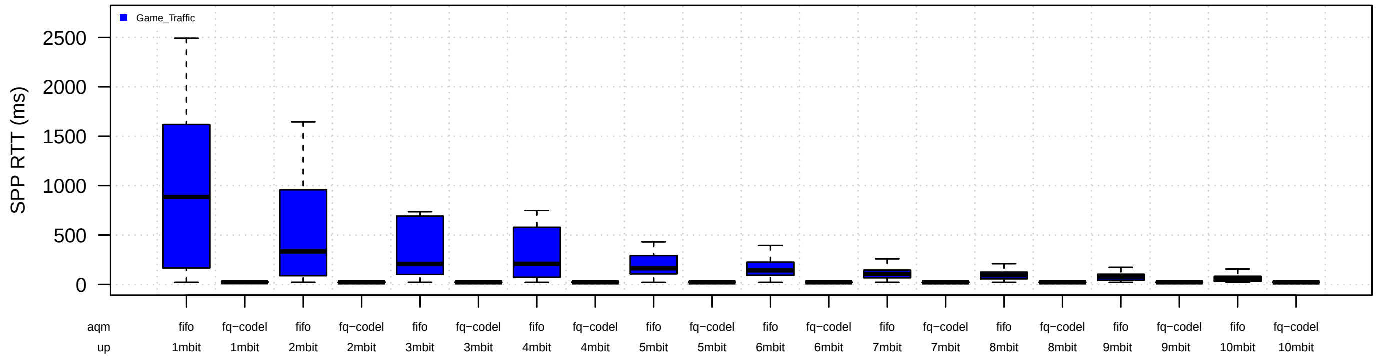

Figure 30. RTT Boxplot comparing for Demo 1 FPS Game traffic for FIFO & FQ-CoDel at fixed 25 Mbps download with changing upload speed, and a 20 ms RTT_{base}

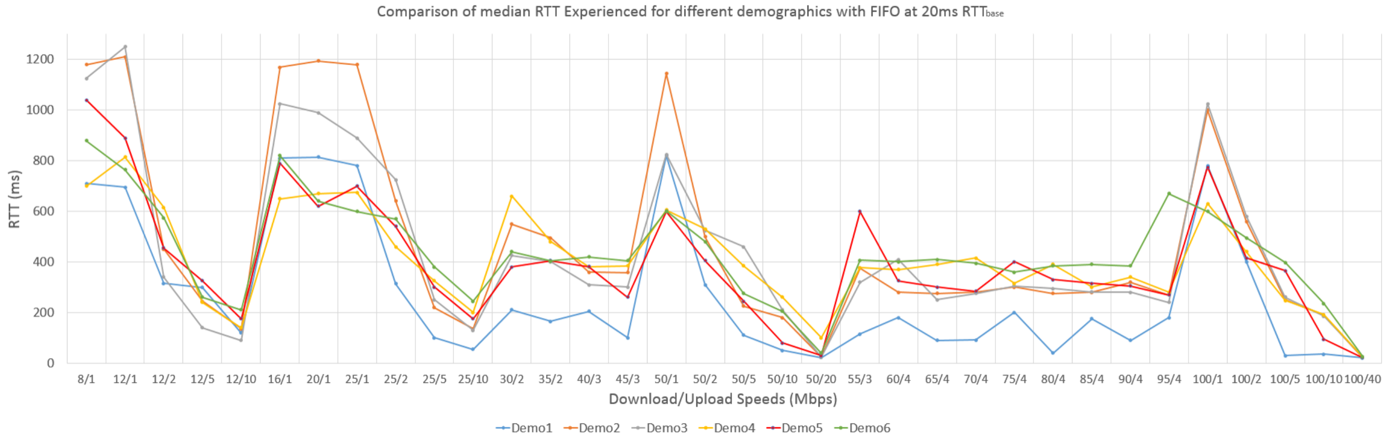

Figure 31. RTT Boxplot medians comparing for Demo 1-6 FPS Game traffic for FIFO at full range of down/up speed tiers, with a 20 ms RTT_{base}

Conversely, Figure 32 shows the RTT levels for game traffic with FQ-CoDel running on the gateway remain consistently low, with game traffic RTT medians staying below 26 ms at each speed for all six demographics shown, which is only slightly higher then the RTT_{base} of 20 ms. Although we can see that the RTT patterns follow the same trends as those shown with FIFO in Figure 31, with download and upload speeds effecting the shape of the graph, we can see that the RTT values only span a range of 21-26 ms (5ms spread), which is so small as to be almost negligible, and not significant enough for user's to notice.

F. AQM penalises throughput to/from distant servers

As shown in previous sections, AQM has a significant effect on reducing the RTTs experienced by latency sensitive game traffic in a network, however as mentioned in Section II.C, AQM can have a negative effect on the throughput levels that TCP connections can achieve on a link. Figures 33 to 36 explore this trade-off, analysing Throughput vs Time graphs for RTT_{base} values of 20 ms, 150 ms, and 250 ms. Figures 33 and 34 look at the

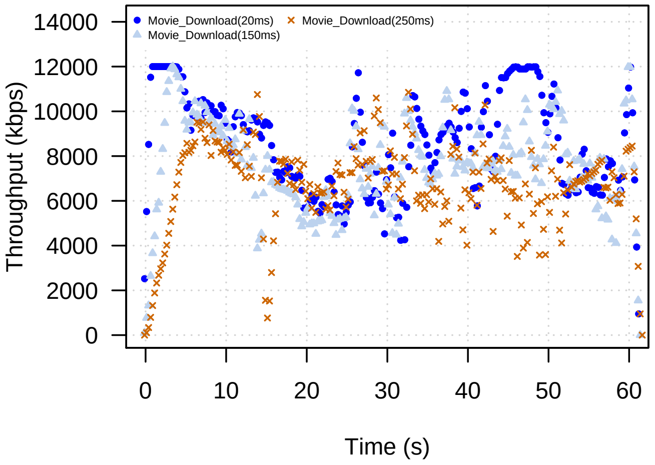

Figure 33. Throughput vs Time for Demo 0.5 Elastic Movie Download traffic at 12/1 Mbps down/up, 20, 150 and 250 ms RTT_{base}, 500 pkt buffer FIFO queue

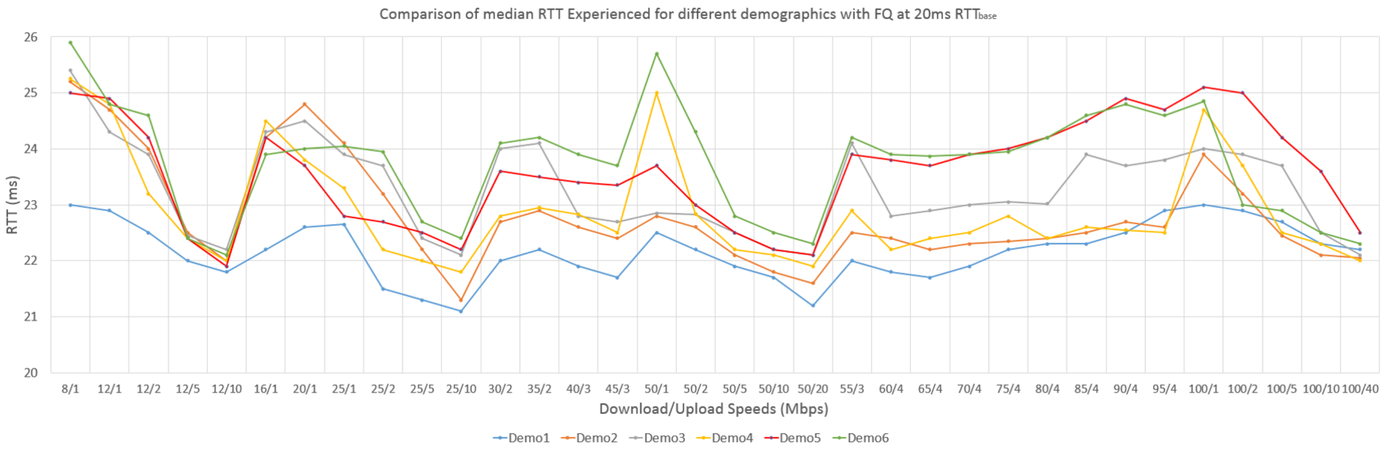

Figure 32. RTT Boxplot medians comparing for Demo 1-6 FPS Game traffic for FQ-CoDel at full range of down/up speed tiers, with a 20 ms RTT$_{\text{base}}$

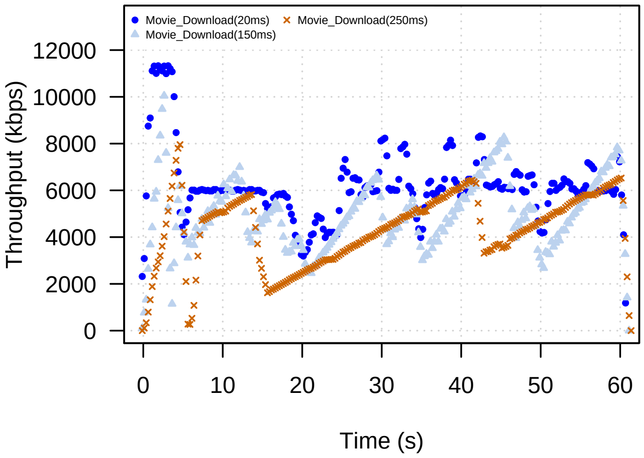

Figure 34. Throughput vs Time for Demo 0.5 Elastic Movie Download traffic at 12/1 Mbps down/up, 20, 150 and 250 ms RTT$_{\text{base}}$, 1000 pkt buffer FQ-CoDel queue

TCP throughput levels for the movie download traffic in demographic 0.5, where there was no cloud traffic congesting the uplink. From Figure 33, we can see that the increases in RTT from 20 ms to 150 ms to 250 ms, have only a small effect on the TCP throughput levels for FIFO, with an average throughput of approximately 8 Mbps for each of the flows on the 12 Mbps link.

Conversely, from Figure 34 we can see that the higher RTT$_{\text{base}}$ for the network are significantly effecting the throughput levels with FQ-CoDel running, with the throughput dropping off as the RTT$_{\text{base}}$ is increased, and throughput levels reaching as low as 2 Mbps at an average of around 4 Mbps for a RTT$_{\text{base}}$ of 250 ms. As explained in Section II.C, this is due to the FQ-CoDel preemptively dropping packets from the movie downloads TCP flow, causing TCP to back off early. With a higher RTT, it takes TCP longer to reach it's optimal throughput levels, and AQM can hinder this process. This means that that if a user was in Melbourne and wanting to download a file from a server based in Europe (250 ms RTT$_{\text{base}}$) it would take twice as long for the file download to complete with FQ-CoDel (4 Mbps) than with FIFO (8 Mbps) on a 12 Mbps link. It is important to note however, that TCP applications usually have elastic completion time requirements that ultimately depend on how long a user is willing to wait.

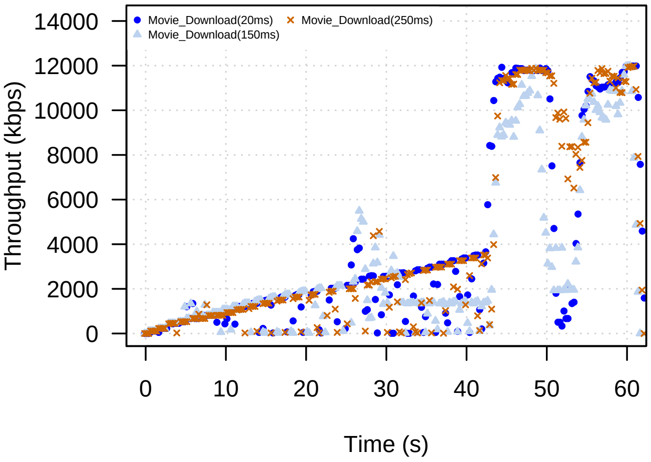

Figure 35. Throughput vs Time for Demo 1 Elastic Movie Download traffic at 12/1 Mbps down/up, 20, 150 and 250 ms RTT$_{\text{base}}$, 500 pkt buffer FIFO queue

Figures 35 and 36 show an example how FQ-CoDel does not always negatively effect throughput levels on a link. Here we can see results for throughput levels of a TCP movie download in demographic 1, where

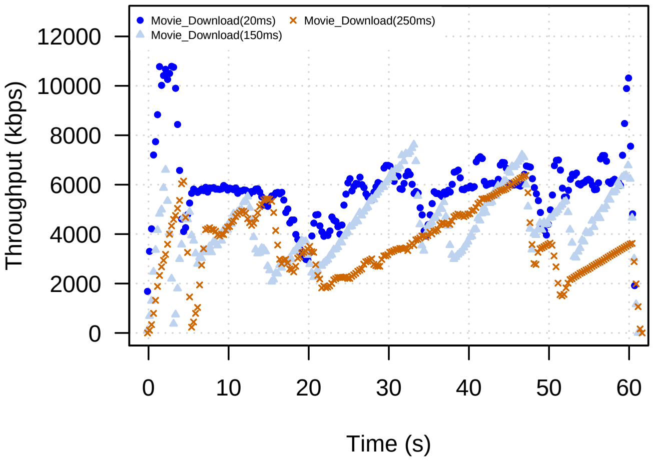

Figure 36. Throughput vs Time for Demo 1 Elastic Movie Download traffic at 12/1 Mbps down/up, 20, 150 and 250 ms RTT_{base}, 1000 pkt buffer FIFO queue

the uplink is being congested by bulk cloud upload traffic. Figure 35 shows that for each of the three RTT_{base} values, the movie download cannot reach optimal throughput levels due to the congested uplink, with throughput speeds staying below 4Mbps for each of the RTT_{base} examples until the movie upload finishes at 40 seconds. Conversely, we can see from Figure 36 that although in theory AQM should usually negatively effect TCP throughput, with AQM reducing congestion on the uplink, throughput levels for each of the RTT_{base} examples instead reach consistently higher levels than those shown for FIFO.

VI. REAL WORLD IMPLICATIONS OF AQM

From our results we have been able to show the distinctive effect that AQM algorithms on a home bottleneck gateway in significantly reducing the RTTs of latency sensitive game traffic. Section V.B shows how ineffective FIFO queueing can be in typical single user household, with Figure 23 showing that minimum upload speeds of 8 Mbps are required for acceptable latency levels (under 200 ms) for game traffic. From the same section we can see that download speeds have little effect on reducing the RTT of game traffic if the uplink is congested. This means that with FIFO, a user would have to opt for the third NBN tier (25/10 Mbps down/up) in order to experience acceptable RTTs for their game traffic if other upload applications were present in the household. Conversely, looking at Figure 29 we can see that FQ-CoDel can consistently offer RTTs only slightly higher than the RTT_{base} of 20 ms regardless of the upload speeds. This means that for this example, with FQ-CoDel a user could opt for the bottom NBN tier of 12/1 Mbps down/up, whilst still achieving even lower RTTs for their latency sensitive game traffic than if they had the third NBN tier with FIFO on the gateway.

From Figure 32 we can see that FQ-CoDel has a consistent RTT for each demographic that did not exceed 26ms for any of the demographics, even with flooded gateways at the lowest speed of 8/1 Mbps down/up. This means that provided user's are happy for their download times to take longer with lower speeds, they can simply base their NBN tier choice off of their fixed traffic speeds requirements whilst achieving minimal RTTs for their latency sensitive traffic, regardless of their other application requirements in an FQ-CoDel situation. However, as shown in Figure 31, in a FIFO situation, user's that want low RTTs for their game traffic become heavily reliant on their download and upload speeds, with only the top two NBN tiers (50/20, 100/40) achieving acceptable game traffic RTTs (under 200 ms) for all six demographics. We can also see that the bottom two NBN tiers are ineffective for all demographics with the exception of demographic 1 at 25/5 Mbps down/up, suggesting that these tiers are not at suitable download and upload levels for typical Australian households if they require cloud upload applications. From this, we can say that FQ-CoDel has the potential to save user's money on higher costing speed tiers, by allowing latency sensitive traffic to have minimal RTTs at the lowest NBN tiers when running in conjunction with other traffic.

Additionally, this may have the potential to reduce the load on the NBN, as with AQM, the high number of user's that would opt for lower speed tiers would reduce the overall maximum download and upload usage levels experienced by the NBN infrastructure, potentially saving on the future costs associated with the NBN rollout. Furthermore, AQM may have great potential for use in remote locations, where NBN Co is unable to justify the costs of rolling out expensive cabling. With AQM used in these situations, low back-haul speeds would still be able to achieve low RTTs for small towns of user's that were previously unable to use latency sensitive traffic such as online games and VoIP calling. Similarly, AQM could have the potential to change the scope of potential business plans, enabling businesses to scope projects in areas that they would not have been able to consider with FIFO, due to their latency sensitive requirements.

Optus currently offers a cable internet "top speed pack" that has speeds of 100/2 Mbps down/up [21]. Although most of the general public would neglect to look too far into the upload speeds, we can see that with FIFO running on the gateway they can be very important. From Figure 31, we can see that at 100/2 Mbps down/up all six demographics have unacceptable RTTs for their latency sensitive game traffic, with all median RTTs shown to be higher than 400 ms. Although these demographics represent a situation where the uplink is flooded with cloud updates, we can also see from Figure 17, that highly skewed asymmetric link's can have their download rates capped by the number of TCP ACK messages that need to be sent on the uplink. This is the case even when there is no other upload traffic to compete with, as the RTT of game traffic increased at a point where the TCP ACK messages flooded the upload (50 Mbps). This congested uplink also puts a cap on the throughput achieved on the link, as the slowing down of the TCP ACK traffic on the uplink will in turn disallow TCP from reaching it's optimal packet throughput levels in ideal timeframes. From this we can say that highly skewed asymmetric link's are only feasible in AQM environments if user's desire low RTTs for their latency sensitive traffic.

It is also important to note that for a user's home gateway, by applying an AQM, it will only effect the uplink bottleneck. In order for the downlink bottleneck to be changed, an AQM would need to be implemented on the ISP's gateway that provided the downlink traffic to the user's home premises. As this would be out the user's control, the fact that the uplink traffic is typically what is offered at a much lower rate to customers of an ISP further validates the need for an AQM at the home gateway. As an AQM can reduce the speed requirement for latency sensitive traffic by so much in the uplink direction, it should not matter that an ISP has not implemented an AQM on their gateway, provided the user does not consume a lot more traffic then their downlink speeds can provide.

VII. CONCLUSIONS

With so many forms of traffic with different requirements such as elastic, fixed and latency sensitive, often they are competing with each other, causing congestion, and increasing the overall delays experienced by each application. For latency sensitive applications such as FPS games, they only produce a very minimal amount of traffic, but in order for the game to remain immersive, require their RTT experienced to be low. This is where AQM comes into play, as opposed to simple FIFO queuing, as it can implement smarter queuing algorithms such as FQ-CoDel in order to ensure TCP backs off sooner, as to not fully congest a home gateways buffer, and allow the overall RTT being experienced by certain traffic to be reduced.

This is important when it comes to latency sensitive traffic, as it is in turn driving the apparent need for speed, as gamers are noticing that the latency they are experiencing is going up when with a congested bottleneck, making the experience a lot less immersive. This in turn is driving idea that across Australia, we need NBN Co to implement a faster service, providing speeds of up to 100/40 Mbps. This is with just FIFO queuing though, as opposed to AQM, so by setting up a physical testbed running a TCP analysis software developed at CAIA Swinburne known as TEACUP, we were able to observe how effective AQM can be.

Firstly, we were able to quickly determine that FPS game traffic goes from experiencing RTT in the realm of 2 ms on top of the initial RTT_{base} when just by itself, to an average of 1000 ms of delay for lower speed tiers when it is competing with different types of traffic in both the down and upstream direction, which is not ideal for game traffic, which requires under 200ms RTTs. Additionally, it was found over a large range of different download and upload speeds, that the upload speed in particular had the most impact when increasing the speed capacity for our demographics. This is due to the fact that the upload speed typically ranges a lot lower, with the bottom NBN speed only offering 1 Mbps. The download speed increasing still improves performance, however as it typically is a lot higher, with the bottom NBN speed being 12 Mbps, it can be enough to reduce the RTT for a congested bottleneck without large amounts of traffic in the down direction.

Additionally, it was found that for bottlenecks where there was no significant uploads, for higher download speeds with only a small upload counterpart, that the RTT being experienced will actually go up due to the amount of ACKs that a fully congestion down-link of speeds such as 60 Mbps down would introduce. This is important to note, as currently Optus is offering an NBN speed tier of 100/2 Mbps, which for a user who is consuming a lot of content in the down direction, will suffer additional RTT even without having any extra traffic in the up direction such as uploading photos to the cloud.

With three modern AQM algorithms run in comparison to FIFO, it was found that although PIE and CoDel did significantly reduce the RTT being experienced for game traffic, FQ-CoDel reduced the RTT more at around 1-5 ms on top of the 20 ms RTT_{base}, constantly for all demographics we tested, regardless of the available bandwidth speeds. This was due to the fact that with the added benefit of FQ scheduler, the game traffic was able to be constantly routed as it is only small packets that arrive at regular intervals, allowing them to be automatically routed without being dropped or delayed.

This makes FQ-CoDel very ideal for user's who often consume their home gateway with bulk TCP traffic in the up and down direction, as well as use latency sensitive applications such as online games, or make VoIP calls. It is important to note though that there is a trade-off in order for FQ-CoDel to be implemented, and that is the completion times for any elastic traffic such as uploading files to the cloud, or downloading a big movie may take longer to and from content servers whose $\mathrm{RTT}_{\mathrm{base}}$ are high, such as servers located outside Australia. For a connection to Europe via a 12/1 Mbps NBN speed tier, downloads could take twice as long using FQ-CoDel instead of FIFO. Really this is a time that is elastic for most user's, as they are typically doing something else when this is happening anyway.

Where AQM is not helpful though is fixed download requirements such as Netflix, so unfortunately for user's who require potentially multiple HD or 4K Netflix streams, they will require a speed that will facilitate for these base requirements. With this research though, it is possible to see that regardless of the user's traffic requirements, latency sensitive application performance will be greatly improved for FQ-CoDel. This in turn means that for user's who don't necessarily require multiple Netflix streams, and are willing for elastic traffic to take a bit longer to download, that they will be able to use an NBN bandwidth speed tier significantly lower then some of the higher tiers offered, saving them money.

Not only is this the case for NBN speed tiers, but also for user's who are currently on a smaller available bandwidth, or will not be able to get better speeds due to their location. This also means that businesses may be able to further scope in additional locations when planning a particular business area, due to the type of traffic they are implementing. Overall, AQM, in particular FQ-CoDel, will be able to reduce the growing need for user's to have faster speeds, especially if their main reason for justification in getting a faster speed is due to latency sensitive applications such as online games, reducing the requirement for most NBN speed tiers, saving the home user from having to upgrade and incur the additional costs required for these greater speeds.

ACKNOWLEDGMENTS

We would like to express our gratitude for all the assistance provided for this project. In particular, the assistance of Professor Grenville Armitage, our project supervisor, who assisted us with forming our ideas, and offering suggestions and general feedback at each stage of the project. Additionally, we would like to thank PhD Candidate Jonathan Kua for his variant and more accurate way to emulate DASH traffic, as well as his general assistance and advice throughput the project. Lastly, we would like to express gratitude to Camberwell High School for granting us the use of the four physical computers that we used to run all of our experiments on this semester.

REFERENCES

[1] I. R. G.1010, "End-user multimedia qos categories," 29 November 2001. Available at https://www.itu.int/rec/T-REC-G.1010-200111-I/en.

[2] W. Stevens, "Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," January 1997. Available at https://tools.ietf.org/html/rfc2001.

[3] T. Hoeiland-Joergensen, P. McKenney, J. Gettys, and E. Dumazet, "The flowqueue-codel packet scheduler and active queue management algorithm," March 2016. Available at https://tools.ietf.org/html/draft-ietf-aqm-fq-codel-06.

[4] F. Baker and R. Panr, "On queueing, marking, and dropping." IETF Draft, April 2016. Available at https://tools.ietf.org/html/rfc7806.

[5] G. White, "Active queue management algorithms for docsis 3.0." IETF Draft, April 2013. Available at http://www.cablelabs.com/wp-content/uploads/2013/11/Active_Queue_Management_Algorithms_DOCSIS_3_0.pdf.

[6] G. Armitage, "Technical reports," October 2016. Available at http://caia.swin.edu.au/reports/4.

[7] P. Natarajan, R. Pan, and F. Baker, "Pie: A lightweight control scheme to address the bufferbloat problem," April 2016. Available at https://tools.ietf.org/html/draft-ietf-aqm-pie-06.

[8] K. Nichols, V. Jacobson, A. Mcgregor, and J. Iyengar, "Controlled delay active queue management." IETF Draft, March 2016. Available at https://tools.ietf.org/html/draft-ietf-aqm-codel-03.

[9] S. Zander, "Teacup v1.0 - command reference," No. 150529C, (Melbourne, Australia), 29 May 2015. Available at http://caia.swin.edu.au/reports/150529C/CAIA-TR-150529C.pdf.

[10] S. Zander and G. Armitage, "Teacup v1.0 - data analysis functions," Tech. Rep. 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.

[11] S. Zander and G. Armitage, "Caia testbed for teacup experiments version 2," tech. rep., Centre for Advanced Internet Architectures, Swinburne University of Technology, 2010. Available at http://caia.swin.edu.au/reports/150210C/CAIA-TR-150210C.pdf.

[12] S. Zander and G. Armitage, "Teacup v1.0 - a system for automated tcp testbed experiments," Tech. Rep. 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.

[13] V. Gueant, "The tcp, udp and sctp network bandwidth measurement tool." Available at https://iperf.fr/.

[14] "Dynamic adaptive streaming over http (mpeg-dash)." Available at http://www.encoding.com/mpeg-dash/.

[15] Profile.Id, "Australia household size." Available at http://profile.id.com.au/australia/household-size.

[16] Telsyte, "Internet uninterrupted." Available at http://www.nbnco.com.au/content/dam/nbnco2/documents/Internet%20Uninterrupted%20Australian%20Households%20of%20the%20Connected%20Future.pdf.

[17] N. co ltd, "Nbn - australiaŠs new broadband network," 2016. Available at http://www.nbnco.com.au/sell-nbn-services/products-services-pricing/Product-identifier/wholesale-speeds.html.

[18] N. co ltd, "How much speed do you need?." Available at http://www.nbnco.com.au/blog/connected-homes/how-much-speed-do-you-need.html.

[19] A. Angove, "Broadband usage guide: How much data do you need?." Available at https://www.whistleout.com.au/Broadband/Guides/Broadband-Usage-Guide.

[20] Wondernetwork, "Ping times between cities." Available at https://wondernetwork.com/pings.

[21] Optus, "How fast are optus' internet speeds." Available at http://www.optus.com.au/shop/broadband/home-broadband/network/internet-speed.

APPENDIX

Appendix A. Figures 37 to 39 show the AQM Boxplots for changing upload speed, fixed 25 Mbps download speed, at 20ms RTT

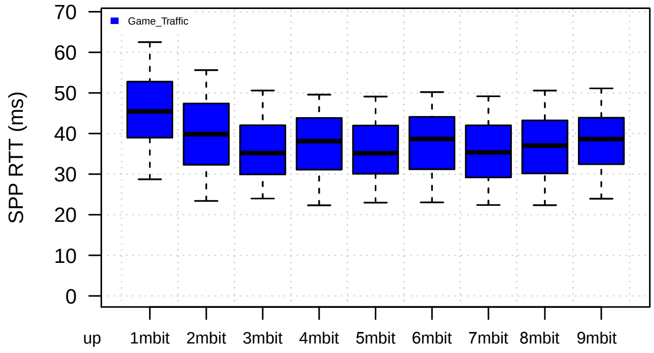

Figure 37. RTT Boxplot for Demo 1 FPS Game traffic at fixed 25 Mbps download with changing upload speed, 20 ms RTT_{base} and 1000 pkt buffer PIE queue

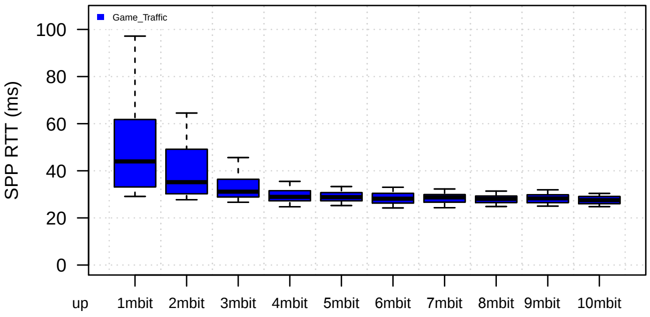

Figure 38. RTT Boxplot for Demo 1 FPS Game traffic at fixed 25 Mbps download with changing upload speed, 20 ms RTT_{base} and 1000 pkt buffer CoDel queue

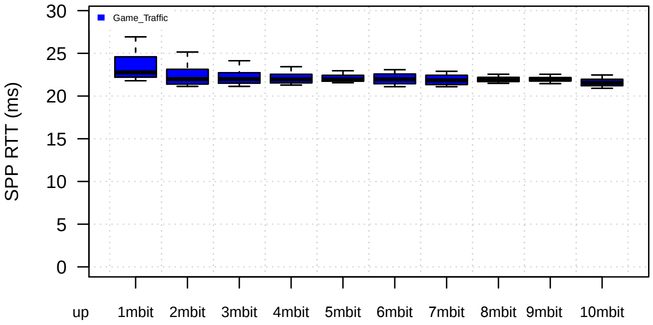

Figure 39. RTT Boxplot for Demo 1 FPS Game traffic at fixed 25 Mbps download with changing upload speed, 20 ms RTT_{base} and 1000 pkt buffer FQ-CoDel queue

Appendix B. Figures 40 to 51 show RTT Boxplots for Demo's 1-6 FPS game traffic with full range of bandwidth speeds for both FIFO and FQ-CoDel, at 20ms RTT_{base}

Appendix C. TEACUP Configuration example for demographic 1 shown on pages 27 to 32.

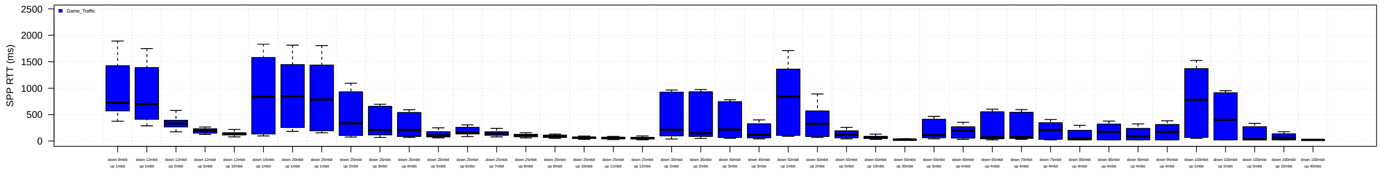

Figure 40. RTT Boxplot for Demo 1 FPS Game traffic for FIFO at full range of down/up speed tiers, with a 20 ms $\mathrm{RTT}_{\mathrm{base}}$

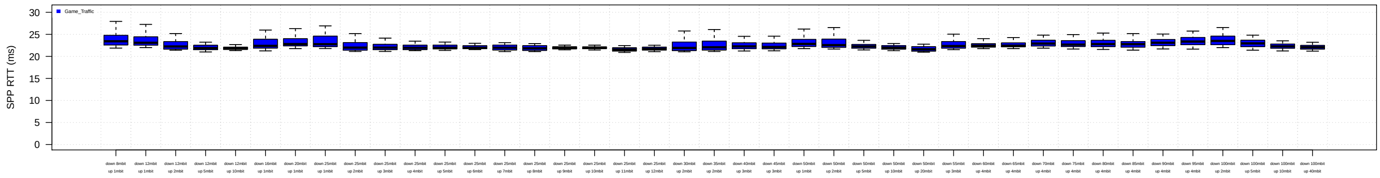

Figure 41. RTT Boxplot for Demo 1 FPS Game traffic for FQ-CoDel at full range of down/up speed tiers, with a 20 ms $\mathrm{RTT}_{\mathrm{base}}$

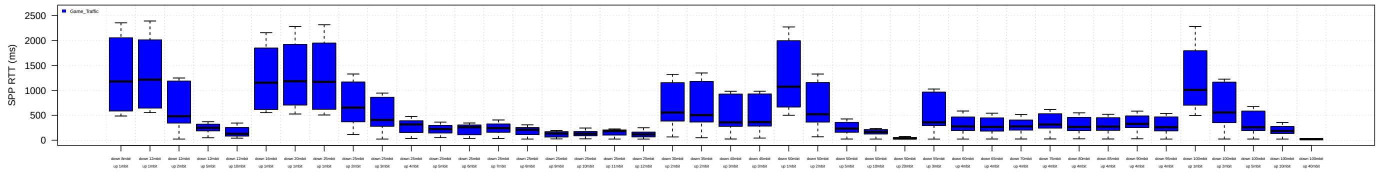

Figure 42. RTT Boxplot for Demo 2 FPS Game traffic for FIFO at full range of down/up speed tiers, with a 20 ms $\mathrm{RTT}_{\mathrm{base}}$

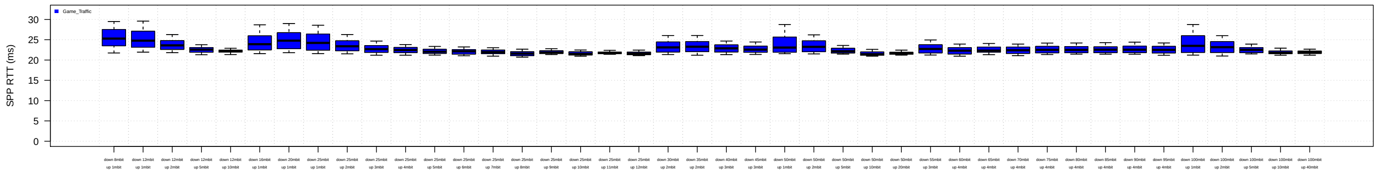

Figure 43. RTT Boxplot for Demo 2 FPS Game traffic for FQ-CoDel at full range of down/up speed tiers, with a 20 ms $\mathrm{RTT}_{\mathrm{base}}$

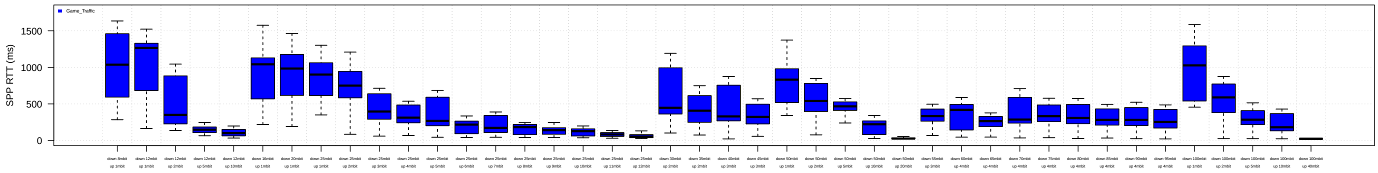

Figure 44. RTT Boxplot for Demo 3 FPS Game traffic for FIFO at full range of down/up speed tiers, with a 20 ms $\mathrm{RTT}_{\mathrm{base}}$

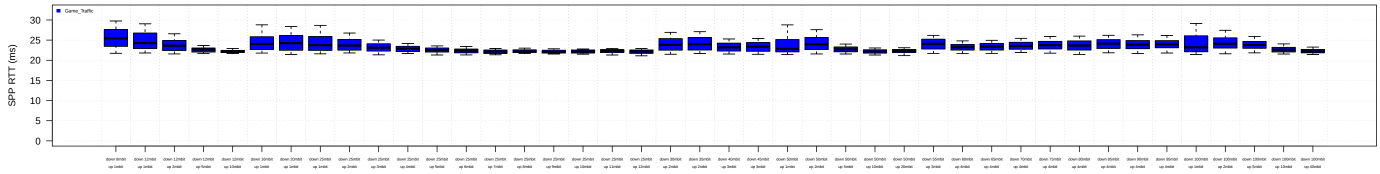

Figure 45. RTT Boxplot for Demo 3 FPS Game traffic for FQ-CoDel at full range of down/up speed tiers, with a 20 ms $\mathrm{RTT}_{\mathrm{base}}$

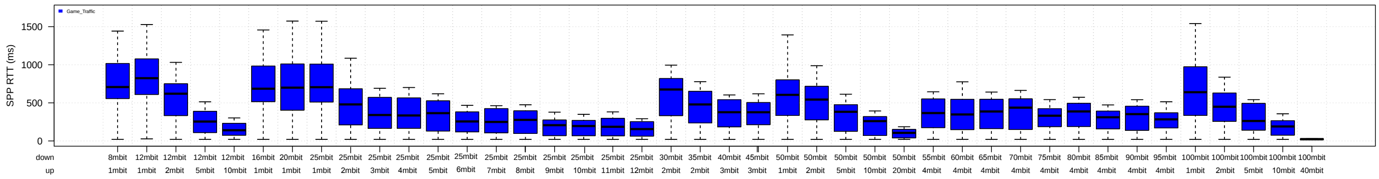

Figure 46. RTT Boxplot for Demo 4 FPS Game traffic for FIFO at full range of down/up speed tiers, with a 20 ms RTT_{base}

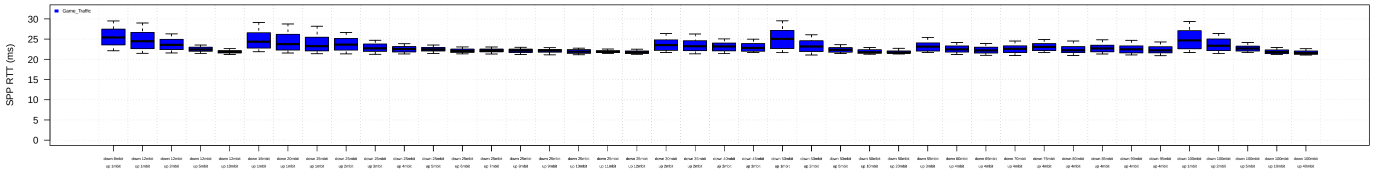

Figure 47. RTT Boxplot for Demo 4 FPS Game traffic for FQ-CoDel at full range of down/up speed tiers, with a 20 ms RTT_{base}

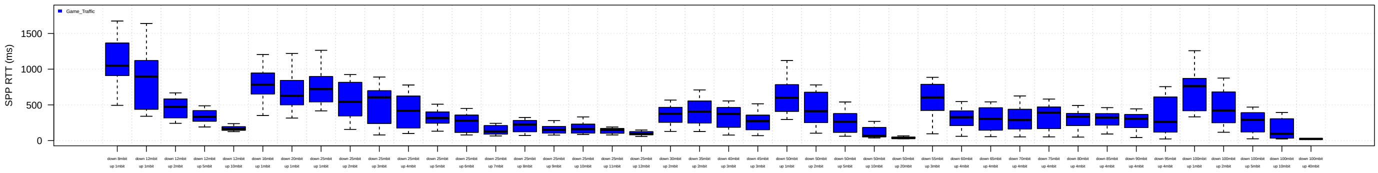

Figure 48. RTT Boxplot for Demo 5 FPS Game traffic for FIFO at full range of down/up speed tiers, with a 20 ms RTT_{base}

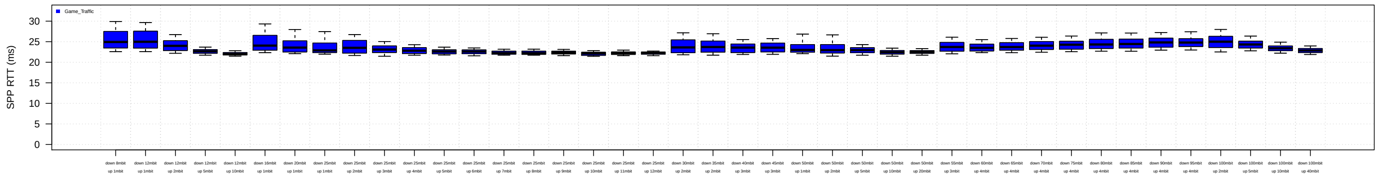

Figure 49. RTT Boxplot for Demo 5 FPS Game traffic for FQ-CoDel at full range of down/up speed tiers, with a 20 ms RTT_{base}

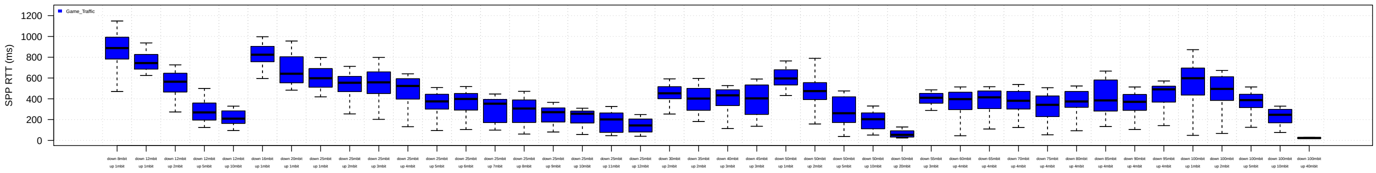

Figure 50. RTT Boxplot for Demo 6 FPS Game traffic for FIFO at full range of down/up speed tiers, with a 20 ms RTT_{base}

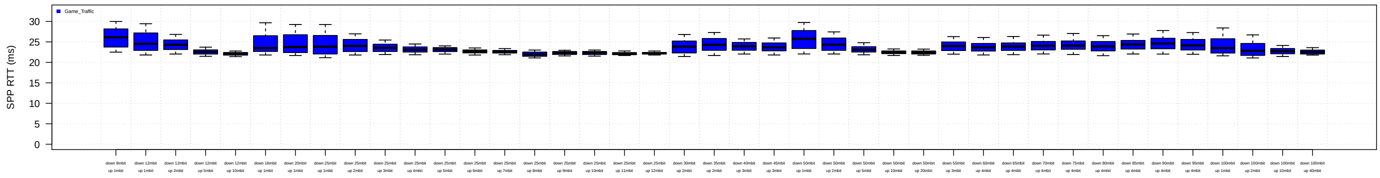

Figure 51. RTT Boxplot for Demo 6 FPS Game traffic for FQ-CoDel at full range of down/up speed tiers, with a 20 ms RTT_{base}

```
# Simple experiment with two tcp flows with two hosts and one
# router
#
# $Id: config-scenario1.py,v ac150f598540 2015/05/21 06:45:42 sebastian $

import sys
import datetime from fabric.api
import env

#
# Fabric config
#

# User and password
env.user = ’root’
env.password = ’testr00t’

# Set shell used to execute commands
env.shell = ’/bin/sh -c’

#
# Testbed config
#

# Path to TEACUP scripts
TPCONF_script_path = ’/usr/home/test/TEACUP-1.0’
# DO NOT remove the following line
sys.path.append(TPCONF_script_path)

# Set debugging level (0 = no debugging info output)
TPCONF_debug_level = 0

# Host lists
TPCONF_router = [’192.168.1.4’, ]
TPCONF_hosts = [ ’192.168.1.2’, ’192.168.1.3’, ]

# Map external IPs to internal IPs
TPCONF_host_internal_ip = {
’192.168.1.4’: [’172.16.10.1’, ’172.16.11.1’],
    ’192.168.1.2’:  [’172.16.10.2’],
    ’192.168.1.3’:  [’172.16.11.2’],
}

#
# Reboot configuration
#

#
# Experiment settings
#

# Maximum allowed time difference between machines in seconds
# experiment will abort cause synchronisation problems
TPCONF_max_time_diff = 4
TPCONF_bc_ping_enable=’1’
TPCONF_bc_ping_rate=1
TPCONF_bc_ping_address=’224.0.1.199’

# Experiment name prefix used if not set on the command line
# The command line setting will overrule this config setting
now = datetime.datetime.today()
TPCONF_test_id = now.strftime("%Y%m%d-%H%M%S") + ’_scenario1’
```

```
# Directory to store log files on remote host
TPCONF_remote_dir = '/tmp/'

# Number of runs for each setting
TPCONF_runs = 2

#
# List of router queues/pipes
#

# Each entry is a tuple. The first value is the queue number and the second value
# is a comma separated list of parameters (see routersetup.py:init_pipe()).
# Queue numbers must be unique.

# Note that variable parameters must be either constants or or variable names
# defined by the experimenter. Variables are evaluated during runtime. Variable
# names must start with a 'V_'. Parameter names can only contain numbes, letter
# (upper and lower case), underscores (_), and hypen/minus (-).

# All variables must be defined in TPCONF_variable_list (see below).

# Note parameters must be configured appropriately for the router OS, e.g. there
# is no CoDel on FreeBSD; otherwise the experiment will abort witn an error.

TPCONF_router_queues = [
    # Set same delay for every host
    ('1', " source='172.16.10.0/24', dest='172.16.11.0/24', delay=V_delay,"
    " loss=V_loss, rate=V_up_rate, queue_disc=V_aqm, queue_size=V_bsize "),
    ('2', " source='172.16.11.0/24', dest='172.16.10.0/24', delay=V_delay, "
    " loss=V_loss, rate=V_down_rate, queue_disc=V_aqm, queue_size=V_bsize "), ]

#
# List of traffic generators
#

# Each entry is a 3-tuple. the first value of the tuple must be a float and is the
# time relative to the start of the experiment when tasks are excuted. If two tasks
# have the same start time their start order is arbitrary. The second entry of the
# tuple is the task number and  must be a unique integer (used as ID for the process).
# The last value of the tuple is a comma separated list of parameters (see the tasks
# defined in trafficgens.py); the first parameter of this list must be the
# task name.

# Client and server can be specified using the external/control IP addresses or host
# names. Then the actual interface used is the _first_ internal address (according to
# TPCONF_host_internal_ip). Alternativly, client and server can be specified as
# internal addresses, which allows to use any internal interfaces configured.

traffic_total = [
    #HD Movie Downloads = 60 second download starting at t=0. iPerf TCP DOWN
    #App Updates = 20 second update starting at t=5. iPerf TCP DOWN
    #Cloud Services = 40 second transfer starting at t=0. iPerf TCP UP
    #General Web = 1 second burst every 15 seconds starting at t=5. iPerf TCP DOWN.
    #E-mail: 3 second burst every 25 seconds starting at t=0. iPerf TCP UP.
    #Online Gaming = Pktgen quake 3 game traffic (4 player game) starting at t=15.
    #Netflix SD = 50 second stream starting at t=10. Chunk sizes can be 1,2,4,6,10 and 15s
      selected further down

    #HD MOVIE
    ( '1.0', '1', " start_iPerf, client='192.168.1.3', server='192.168.1.2', port=5020,
    duration=60 " ),
    #APP UPDATES
    ( ’5.0’, ’2’, " start_iPerf, client=’192.168.1.3’, server=’192.168.1.2’, port=5025,
    duration=20 " ),
    #CLOUD
    ( ’0.0’, ’3’, " start_iPerf, client=’192.168.1.2’, server=’192.168.1.3’, port=5030,
    duration=40 " ),
    #WEB
    ( ’5.0’, ’4’, " start_iPerf, client=’192.168.1.3’, server=’192.168.1.2’, port=5035,
    duration=1 " ),
    ( ’20.0’, ’5’, " start_iPerf, client=’192.168.1.3’, server=’192.168.1.2’, port=5040,
    duration=1 " ),
    ( ’35.0’, ’6’, " start_iPerf, client=’192.168.1.3’, server=’192.168.1.2’, port=5045,
    duration=1 " ),
    ( ’50.0’, ’7’, " start_iPerf, client=’192.168.1.3’, server=’192.168.1.2’, port=5050,
    duration=1 " ),
    #Email
    ( ’0.0’, ’8’, " start_iPerf, client=’192.168.1.2’, server=’192.168.1.3’, port=5055,
    duration=3 " ),
    ( ’25.0’, ’9’, " start_iPerf, client=’192.168.1.2’, server=’192.168.1.3’, port=5060,
    duration=3 " ),
    ( ’50.0’, ’10’, " start_iPerf, client=’192.168.1.2’, server=’192.168.1.3’,
    port=5065, duration=3 " ),
    #GAMING
    ( ’15.0’, ’11’, " start_fps_game, clients=’192.168.1.2:10000’,
    server=’192.168.1.3:27960’, game_type=’q3’, duration=45 , noclients_game=’4’  " ),
    #Netflix SD
    ( ’0.0’, ’12’, " start_http_server, server=’192.168.1.3’, port=’8000’,
    docroot=’/data/dash_dataset’ "),
    ( ’10.0’, ’13’, " start_dash_streaming_dashjs, client=’192.168.1.2’,
    dest=’192.168.1.3’, serv_port=’8000’, chunk_size=V_chunksize, mpd=V_mpd, duration=50 " ),
]

# THIS is the traffic generator setup we will use TPCONF_traffic_gens = traffic_total

#
# Traffic parameters
#

# Duration in seconds of traffic
TPCONF_duration = 60

# TCP congestion control algorithm used
# Possible algos are: default, host<N>, newreno, cubic, cdg, hd, htcp, compound, vegas
# Note that the algo support is OS specific, so must ensure the right OS is booted
# Windows: newreno (default), compound
# FreeBSD: newreno (default), cubic, hd, htcp, cdg, vegas
# Linux: newreno, cubic (default), htcp, vegas
# Mac: newreno
# If you specify ’default’ the default algorithm depending on the OS will be used
# If you specify ’host<N>’ where <N> is an integer starting from 0 to then the
# algorithm will be the N-th algorithm specified for the host in TPCONF_host_TCP_algos
# (in case <N> is larger then the number of algorithms specified, it is set to 0
TPCONF_TCP_algos = [’newreno’, ’cubic’, ]

# Specify TCP congestion control algorithms used on each host
TPCONF_host_TCP_algos = {

}

# Specify TCP parameters for each host and each TCP congestion control algorithm
# Each parameter is of the form <sysctl name> = <value> where <value> can be a constant
# or a V_ variable
TPCONF_host_TCP_algo_params = {
}

# Specify arbitray commands that are executed on a host at the end of the host
# intialisation (after general host setup, ecn and tcp setup). The commands are
# executed in the shell as written after any V_ variables have been replaced.
# LIMITATION: only one V_ variable per command
TPCONF_host_init_custom_cmds = {

}

# Emulated delays in ms
TPCONF_delays = [10, 75, 125]

# Emulated loss rates
TPCONF_loss_rates = [0]

# Emulated bandwidths (downstream, upstream)
TPCONF_bandwidths = [
    (’8mbit’, ’1mbit’),
    (’12mbit’, ’1mbit’),
    (’12mbit’, ’2mbit’),
    (’12mbit’, ’5mbit’),
    (’12mbit’, ’10mbit’),
    (’16mbit’, ’1mbit’),
    (’20mbit’, ’1mbit’),
    (’25mbit’, ’1mbit’),
    (’25mbit’, ’2mbit’),
    (’25mbit’, ’3mbit’),
    (’25mbit’, ’4mbit’),
    (’25mbit’, ’5mbit’),
    (’25mbit’, ’6mbit’),
    (’25mbit’, ’7mbit’),
    (’25mbit’, ’8mbit’),
    (’25mbit’, ’9mbit’),
    (’25mbit’, ’10mbit’),
    (’25mbit’, ’11mbit’),
    (’25mbit’, ’12mbit’),
    (’30mbit’, ’1mbit’),
    (’30mbit’, ’2mbit’),
    (’35mbit’, ’1mbit’),
    (’35mbit’, ’2mbit’),
    (’40mbit’, ’1mbit’),
    (’40mbit’, ’3mbit’),
    (’45mbit’, ’1mbit’),
    (’45mbit’, ’3mbit’),
    (’50mbit’, ’1mbit’),
    (’50mbit’, ’2mbit’),
    (’50mbit’, ’5mbit’),
    (’50mbit’, ’10mbit’),
    (’50mbit’, ’20mbit’),
    (’55mbit’, ’1mbit’),
    (’55mbit’, ’3mbit’),
    (’60mbit’, ’1mbit’),
    (’60mbit’, ’4mbit’),
    (’65mbit’, ’1mbit’),
    (’65mbit’, ’4mbit’),
    (’70mbit’, ’1mbit’),
    (’70mbit’, ’4mbit’),
    (’75mbit’, ’1mbit’),
    (’75mbit’, ’4mbit’),
    (’80mbit’, ’1mbit’),
    (’80mbit’, ’4mbit’),
    (’85mbit’, ’1mbit’),
    ('85mbit', '4mbit'),
    ('90mbit', '1mbit'),
    ('90mbit', '4mbit'),
    ('95mbit', '1mbit'),
    ('95mbit', '4mbit'),
    ('100mbit', '1mbit'),
    ('100mbit', '2mbit'),
    ('100mbit', '5mbit'),
    ('100mbit', '10mbit'),
    ('100mbit', '40mbit'),

]

# AQM
# Linux: FIFO (mapped to pFIFO), pFIFO, bFIFO, fq_CoDel, CoDel, PIE, red, ...
#        (see tc man page for full list)
# FreeBSD: FIFO, red
TPCONF_aqms = ['FIFO']

# Buffer size
# If router is Linux this is mostly in packets/slots, but it depends on AQM
# (e.g. for bFIFO it's bytes)
# If router is FreeBSD this would be in slots by default, but we can specify byte sizes
# (e.g. we can specify 4Kbytes)
TPCONF_buffer_sizes = [500]

# Video chunk size and MPD pair
TPCONF_chunksize_mpd = [
    # ('1','BigBuckBunny_1s.mpd'),
    ('2','BigBuckBunny_2s.mpd'),
    # ('4','BigBuckBunny_4s.mpd'),
    # ('6','BigBuckBunny_6s.mpd'),
    # ('10','BigBuckBunny_10s.mpd'),
    # ('15','BigBuckBunny_15s.mpd'),
]

#
# List of all parameters that can be varied and default values
#

# The key of each item is the identifier that can be used in TPCONF_vary_parameters
# (see below).
# The value of each item is a 4-tuple. First, a list of variable names.
# Second, a list of short names uses for the file names.
# For each parameter varied a string '_<short_name>_<value>' is appended to the log
# file names (appended to chosen prefix). Note, short names should only be letters
# from a-z or A-Z. Do not use underscores or hyphens!
# Third, the list of parameters values. If there is more than one variable this must
# be a list of tuples, each tuple having the same number of items as teh number of
# variables. Fourth, an optional dictionary with additional variables, where the keys
# are the variable names and the values are the variable values.

TPCONF_parameter_list = {
#   Vary name       V_ variable   file name values           extra vars
    'delays'        :  (['V_delay'],      ['del'],  TPCONF_delays,       {}),
    'loss'          :  (['V_loss'],       ['loss'],     TPCONF_loss_rates,   {}),
    'tcpalgos'      :  (['V_tcp_cc_algo'],['tcp'],  TPCONF_TCP_algos,    {}),
    'aqms'      :  (['V_aqm'],    ['aqm'],  TPCONF_aqms,         {}),
    'bsizes'        :  (['V_bsize'],      ['bs'],   TPCONF_buffer_sizes,     {}),
    'runs'      :  (['V_runs'],       ['run'],  range(TPCONF_runs),      {}),
    'bandwidths'    :  (['V_down_rate', 'V_up_rate'], ['down', 'up'], TPCONF_bandwidths, {}),
    'chunksize_mpd' :  (['V_chunksize','V_mpd'], ['chunksize','mpd'], TPCONF_chunksize_mpd,
      {}),
}

# Default setting for variables (used for variables if not varied)

# The key of each item is the parameter  name. The value of each item is the default
# parameter value used if the variable is not varied.

TPCONF_variable_defaults = {
#   V_ variable          value
    'V_duration'     :   TPCONF_duration,
    'V_delay'        :   TPCONF_delays[0],
    'V_loss'         :   TPCONF_loss_rates[0],
    'V_tcp_cc_algo'      :   TPCONF_TCP_algos[0],
    'V_down_rate'        :   TPCONF_bandwidths[0][0],
    'V_up_rate'          :   TPCONF_bandwidths[0][1],
    'V_aqm'          :   TPCONF_aqms[0],
    'V_bsize'            :   TPCONF_buffer_sizes[0],
    'V_chunksize'    :   TPCONF_chunksize_mpd[0][0],
    'V_mpd'      :   TPCONF_chunksize_mpd[0][1],
}

# Specify the parameters we vary through all values, all others will be fixed
# according to TPCONF_variable_defaults
TPCONF_vary_parameters = [ 'aqms' , 'bsizes' , 'chunksize_mpd', 'bandwidths' , 'delays', '
  runs', ]
```