Generating Dynamic Adaptive Streaming over HTTP Traffic Flows with TEACUP Testbed

Jonathan Kua, Grenville Armitage
Centre for Advanced Internet Architectures, Technical Report 161216A
Swinburne University of Technology
Melbourne, Australia
jtkua@swin.edu.au, garmitage@swin.edu.au

***Abstract*—HTTP-based video streaming has become popular in recent years, with Dynamic Adaptive Streaming over HTTP (DASH) emerging as an ISO/IEC standard for live and on-demand video streaming services. Netflix and YouTube employ DASH-like streaming strategies and account for more than 50% of North American traffic in 2015, representing a significant source of Internet traffic. Consumer video streams are most likely to be bottlenecked by last-mile ISP links and impacted by emerging Active Queue Management (AQM) schemes for counteracting bufferbloat. However, the interactions between different TCP algorithms, DASH traffic (within a mix of other typical traffic) and the underlying AQMs are not well understood. Experiments in a controlled testbed allow shedding more light on this issue. We extended "TCP Experiment Automation Controlled Using Python" (TEACUP) to use dash.js and VLC clients for running repeatable DASH experiment trials with different TCP algorithms over a range of emulated network conditions, bottleneck rate limits and AQMs.**

***Index Terms*—video streaming, DASH, HTTP, TCP, TEACUP**

I. Introduction

Transmission Control Protocol (TCP) has been the dominant transport layer protocol that carries the bulk of all traffic across the Internet for many decades. Several TCP congestion control algorithms were developed for performance optimisation over the last few decades. Although TCP was traditionally used for reliable bulk transfers, recently it is also becoming the protocol of choice for multimedia streaming applications – Dynamic Adaptive Streaming over HTTP (DASH) has recently emerged as an International Organisation for Standardisation/International Electrotechnical Commission (ISO/IEC) standard for live and on-demand streaming [1]. Netflix and YouTube employ DASH-like video streaming strategies [2], [3] and account for more than 50% of North American traffic in 2015 [4], representing a significant source of Internet traffic. Consumer video streams are most likely to be bottlenecked by last-mile ISP links and impacted by emerging Active Queue Management (AQM) schemes such as Proportional Integral controller Enhanced (PIE) [5], Controlled Delay (CoDel) [6] and FlowQueue-Controlled Delay (FQ-CoDel) [7] for counteracting bufferbloat. However, the interactions between different TCP algorithms, DASH traffic (within a mix of other typical traffic) and the underlying AQMs are not well understood. Experiments in a controlled testbed allow shedding more light on this issue. Hence, we enhanced "TCP Experiment Automation Controlled Using Python" (TEACUP)[1]to run repeatable DASH experiments with industry-grade clients.

Based on a configuration file and utilising Python Fabric [8], these enhancements allow TEACUP to perform a series of DASH experiments with different traffic mixes, different bottlenecks (bandwidths, queuing schemes, buffer size), different emulated network paths (path delays, packet loss rates), and different host settings (TCP congestion control algorithm and system settings). For each experiment permutation, TEACUP automatically collects relevant information for post-analysis , such as tcpdump files, SIFTR and Web10G logs. More details on the design and implementation of TEACUP-specific testbed can be found in [9]. TEACUP also provides a number of native data analysis tasks [10] and data visualisation functions with TEAPLOT [11].

This technical report describes the enhancements made to TEACUP for generating DASH traffic flows. The rest of the report is organised as follows. Section II presents an overview of DASH architecture and industry standardisation. Section III describes the enhancements

[1]The TEACUP project originated at CAIA (http://caia.swin.edu.au/tools/teacup), and from version 1.0 the source code is freely available on SourceForge at http://sourceforge.net/projects/teacup

made to TEACUP for DASH support and provides patching/usage instructions. Section III presents the testbed setup and test conditions for an experiment trial. Section IV presents illustrative DASH experiment results. Section V provides concluding remarks and outlines future work.

II. Background

This section presents the overall DASH architecture and its applications, the benefits of using HTTP and the general principles driving the rate adaptation algorithms.

A. DASH Architecture Overview

In DASH systems [1] (summarised in Figure 1), video content is encoded into multiple versions at different discrete bitrates. Each encoded video is then fragmented into small video segments or chunks, each containing a few seconds of video. Chunks from one bitrate are aligned in the video time line to chunks from other bitrates so that the client can smoothly switch bitrates, if necessary, at the chunk boundary. Content information such as video profiles, metadata, mimeType [12], [13], codecs, byte-ranges, server IP addresses, and download URLs is described in the associated Media Presentation Description (MPD) files. The MPD describes a piece of video content within a specific duration as a *Period*. In a Period, there are multiple versions of the content, each known as a *Representation*. In a Representation, there are multiple video *segments* or *chunks*. URLs pointing to the video chunks in an MPD can either be explicitly described or be constructed via a template (client deriving a valid URL for each chunk at a certain Representation) [1]. Video chunks are 3GPP-formatted [12], [13] and in each Representation, there is a single initialisation segment which contains the configuration data and many media segments. Concatenating the initialisation segment and a series of media segments results in a continuous stream of video. Video chunks and MPDs are then served to clients by using standard HTTP servers.

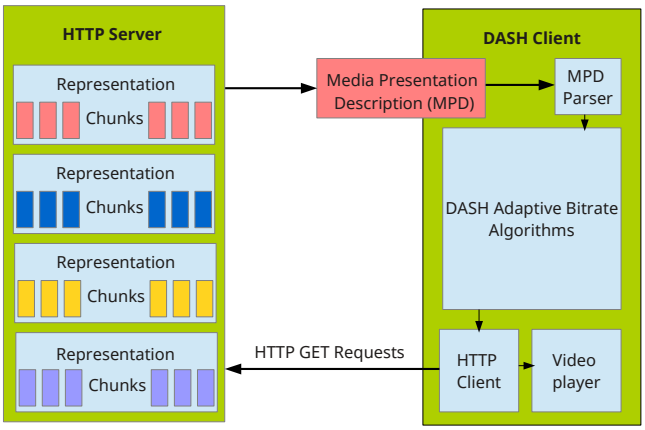


Figure 1: DASH client-server architecture

Unlike traditional streaming strategies, DASH does not control the video transmission rate directly. It depends on the underlying TCP algorithm to regulate the video transmission rate, which is determined by the congestion feedback from the client-server network path. When a streaming session starts, the client requests the MPD file from the HTTP server and then starts requesting video chunks (typically in sequential order) as fast as possible to fill the playout buffer. Once this buffer is full, the player enters a steady state phase where it periodically downloads new chunks. In the steady state, the player is in the ON state when it is downloading a chunk, and in the OFF state otherwise (resulting in an alternating ON-OFF traffic pattern illustrated in Figure 2). The time between the start of two consecutive ON periods is termed cycle time (typically the chunk size – the amount of multimedia content within each chunk – in seconds). The client typically keeps a few chunks in the buffer to maintain adequate playback.

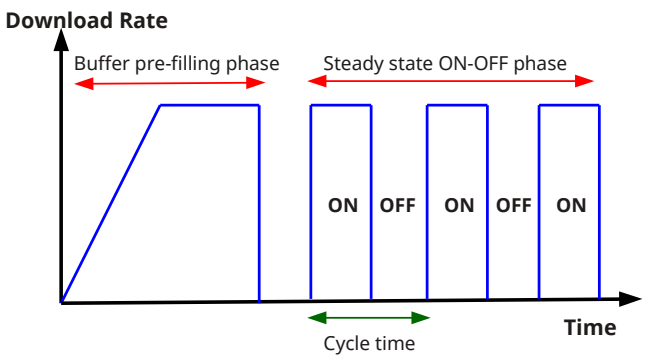


Figure 2: DASH's bursty ON-OFF behaviour

The video player uses various feedback signals observed for each chunk (such as recently achieved

throughput[2] and/or playout buffer occupancy) to select a suitable video rate for the next chunk to be downloaded. Consider an example when using achieved throughput as a criteria for its Adaptive Bitrate (ABR) algorithm. If the throughput is high, ABR should select a higher video rate to provide better Quality of Experience (QoE) for the user. On the other hand, if the throughput is low, ABR should dynamically switch to a lower video rate to avoid playout buffer under-run. A good ABR algorithm is responsive to fluctuating network conditions and adapts smoothly to provide better QoE [14].

The presented video bitrate (or quality) is limited by the video rates provided by the server, the information contained in the MPD and the network bandwidth. The DASH clients cannot match the network throughput perfectly; they can only achieve the (discrete) video rates described by the MPD. It will select a rate below the estimated throughput to sustain video playback and in the case where the network bandwidth exceeds the maximum video bitrate, the video rate is capped to the maximum video bitrate. Hence, in some ABR approaches, the server can artificially limit the video rate by only providing specific rates in the MPD to protect the network. The "smoothness" between video bitrate transitions depend on the encoding granularity (the number of video representations) of the video content provided at the server.

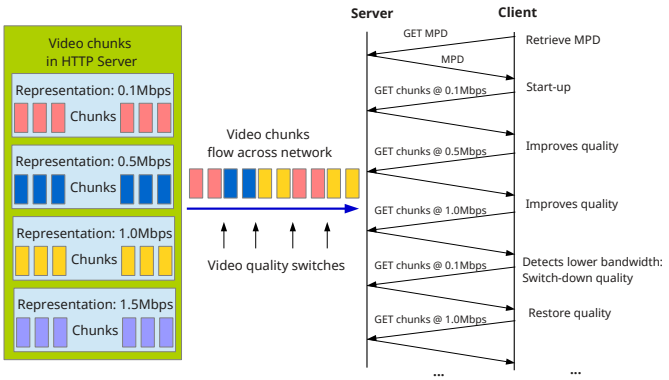


Figure 3: Timeline illustration of DASH adaptive behaviour

As illustrated in Figure 3, the client quickly ramps up its video bitrate request during start-up to pre-fill its playout buffer. When the client detects a reduction in bandwidth capacity (by utilising feedback signals from previous chunks), it "backs off" by requesting a lower video bitrate. When the network capacity increases again, it then restores its video quality. As a result, the client is able to stream the video seamlessly without having to over-provision the network or keep an oversize playout buffer.

B. *Standardisation*

MPEG-DASH has been standardised by 3GPP (first open as Work Item in January 2009 and finalised in March 2010) and became an ISO/IECstandard for adaptive streaming [15]. The standard defines guidelines for media presentation, segmentation, and a collection of standard XML formats for the manifest file (MPD). However, specific client implementation and rate adaptation techniques are not part of the standard [1]. Hence, commercial streaming services that use DASH implement their own proprietary techniques both for media representation and for client adaptation.

The DASH Industry Forum [16] (a group containing leading streaming companies) drives the adoption and research in modern adaptive streaming technologies. The community has developed an open-source dash.js [17] reference player, which employs Media Source Extensions (MSEs) in a web/HTML5-based video player for research and testing purposes.

III. TEACUP Enhancements for DASH Support

In this section, we describe our extensions to TEACUP for running and analysing DASH experiments.

The TEACUP DASH traffic generator starts dash.js v2.0.0 [17] in Chromium [18] or Firefox [19] web browser with Xorg [20], and starts requesting video chunks from the lighttpd [21] server (with persistent HTTP connections) that hosts the DASH dataset. The adaptation heuristics are implemented on the client side, and are dependent on the network conditions.

A. *DASH server and dataset*

DASH dataset is hosted on a regular lighttpd HTTP server in the testbed hosts. We utilise the dataset made available by ITEC-DASH [22]. The dataset[3] comprise of full-length sequences at different representations in terms of bitrates, resolutions and quality (*representation rates*). It is encoded and multiplexed using different chunk sizes (1, 2, 4, 10, 15 seconds) and are made available with

[2]Estimated from the size of previous chunks and the time to retrieve them.

[3]The BigBuckBunny video content is an open-source animation video, with source quality of 1080p, 9mins 46secs long by the Blender Institute. It can be downloaded from http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/BigBuckBunny

corresponding Media Presentation Description (MPD) files for chunk-by-chunk video request. There is also an unsegmented version of the videos and the corresponding MPD, which allows for byte-range requests.

For example, Table I shows 20 levels of representation rates for the 10-sec dataset.

Resolution	Encoding Level	Representation Rates
320x240	1 - 3	45, 88, 127kbps
480x360	4 - 8	177, 217, 253, 317, 369kbps
854x480	9 - 10	503, 569kbps
1280x720	11 - 14	0.8, 1.0, 1.2, 1.4Mbps
1920x1080	15 - 20	2.1, 2.4, 2.9, 3.2, 3.5, 3.8Mbps

Table I: Representation rates available for 10-sec dataset

B. DASH clients

1) dash.js HTML5 player: We integrated the BSD-3 licensed dash.js DASH client into our testbed to serve as our DASH client retrieving content from the lighttpd server. dash.js is an initiative of the DASH Industry Forum [17] to establish a production quality framework for building multimedia players for MPEG-DASH content playback using client-side open source JavaScript libraries leveraging the Media Source Extensions API set as defined by the the World Wide Web Consortium (W3C). All source codes are covered by the BSD-3 license.

Figure 4: dash.js Web User Interface

To install Xorg, Chromium or Firefox on FreeBSD:

```
> pkg install xorg
> pkg install chrome

AND/OR

> pkg install firefox
```

To install Xorg, Chromium or Firefox on Linux OpenSUSE:

```
> zypper install xorg-x11
> zypper install xorg-x11-server
> zypper install chromium

AND/OR

> zypper install firefox
```

2) VLC media player : Although dash.js is our main DASH streaming client, we also explore VLC as our alternative streaming client. The VLC-DASH plugin [23] has been integrated into the official release of VLC with continued improvements. We use the development branch of VLC to keep up to date with the adaptive streaming mechanisms implemented in VLC.

Appendix A and B describes the necessary steps to build VLC development code for DASH support with minimum `Xorg` environment and library packages. These instructions are tested against the TEACUP host image based on [9].

C. TEACUP DASH Traffic Generator

The `start_dash_streaming_dashjs` and `start_dash_streaming_vlc` traffic generator starts a video streaming client (dash.js or VLC) that request video chunks from the lighttpd server that stores the DASH dataset.Table II and III describes the configurable parameters for both traffic generators.

Parameters	Default	Description
client	‘’	IP address of DASH client
serv	‘’	IP address of DASH server
duration	‘’	Duration in seconds
wait	‘’	Time to wait before process is started
serv_port	‘’	Port number of DASH server serving the dataset
browser	‘chrome’	Browser type (chrome or firefox)
chunk_size	‘’	Video chunk size (depending on dataset)
mpd	‘’	File name of Media Presentation Description (depending on dataset)

Table II: Configurable parameters for `start_dash_streaming_dashjs` traffic generator

Parameters	Default	Description
client	‘’	IP address of DASH client
serv	‘’	IP address of DASH server
duration	‘’	Duration in seconds
wait	‘’	Time to wait before process is started
serv_port	‘’	Port number of DASH server serving the dataset
chunk_size	‘’	Video chunk size (depending on dataset)
verbosity	‘2’	Logging verbosity of VLC client (1-10)
mpd	‘’	File name of Media Presentation Description (depending on dataset)
vlc_location	‘’	Location of VLC source code directory
base_url	‘’	Location of video files relative to the `docroot` of the DASH server

Table III: Configurable parameters for `start_dash_streaming_vlc` traffic generator

D. Patching TEACUP revision 71e6cf on the default branch

We created a patch for TEACUP to support DASH traffic generators natively. To apply the patch, first clone TEACUP source code (revision 71e6cf)[4] from the Mercurial repository on SourceForge as follows:

```
hg clone -r 71e6cf http://hg.code.sf.net/
   p/teacup/code teacup-code
```

Then, download the patch[5] and apply it as follows:

1) Extract the patch file:

```
tar -xvf teacup-dash-patch-0.1.tgz -C
    teacup-code
```

2) Apply the patch:

```
cd teacup-code
patch -p1 < teacup-dash-patch-0.1/
   teacup-dash-0.1.patch
```

E. TEACUP Configuration

TEACUP configuration file, `config.py` should include an additional `TPCONF` variable, `TPCONF_chunksize_mpd` to setup DASH streaming experiments. `TPCONF_chunksize_mpd` is a list of video chunk sizes and corresponding Media Presentation Description (MPD) file. Each entry is a tuple. The first value is the video chunk size in seconds (as provided in the dataset) and the second value is the file name of the corresponding MPD.

[4]This is the latest revision at the time of writing.

[5]The patch can be downloaded from http://caia.swin.edu.au/tools/teacup/downloads/teacup-dash-patch-0.1.tgz

```
# Generating DASH streaming traffic

traffic_dash_streaming = [

        ( '0.0', '1', " start_http_server, server='newtcp33', port='8000',
                        docroot='/data/dash_dataset' "),
        ( '0.1', '2', " start_dash_streaming_dashjs, client='newtcp20',
                        serv='newtcp33', serv_port='8000', browser='firefox',
                        chunk_size=V_chunksize, mpd=V_mpd, duration=V_duration "),
]


# This is the traffic generator setup we will use
TPCONF_traffic_gens = traffic_dash_streaming
```

Figure 5: DASH traffic generator definition example

```
# Video chunk size and MPD pair
TPCONF_chunksize_mpd = [
    ('10','BigBuckBunny_10s.mpd'),
]
```

Figure 6: `TPCONF` variable defining video chunk size and MPD

```
TPCONF_parameter_list = {
#   Vary name           V_ variable       file name     values                  extra vars
        [...omitted...]
    'chunksize_mpd' :  (['V_chunksize','V_mpd'], ['chunksize','mpd'], TPCONF_chunksize_mpd, {}),
}
```

Figure 7: Parameter list definition example

```
TPCONF_variable_defaults = {
#   V_ variable                 value
        [...omitted...]
    'V_chunksize'       :       TPCONF_chunksize_mpd[0][0],
    'V_mpd'             :       TPCONF_chunksize_mpd[0][1],
}
```

Figure 8: Variable defaults definition example

```
TPCONF_vary_parameters = ['tcpalgos', 'delays', 'bandwidths', 'aqms','bsizes', 'chunksize_mpd']
```

Figure 9: Varying parameters for experiment example

```
exp_20161114-134000_tcp_host0_del_20_down_12mbit_up_1mbit_aqm_pfifo_bs_180
_chunksize_10_mpd_BigBuckBunny-10s.mpd
```

Figure 10: Example experiment directory name

IV. Illustrative DASH Experiments

Here we describe the experiment testbed setup and test conditions for our illustrative DASH experiments.

A. Experiment testbed setup

We built a test environment emulating standard Internet connections from home with TEACUP testbed. For illustrative purposes, we run experiment trials involving dash.js client (on top of FreeBSD NewReno) retrieving 10-sec video chunks from FreeBSD NewReno [24] lighttpd server. The path has 40ms base RTT delay, 12Mbps downstream / 1 Mbps upstream (12/1 Mbps) bottleneck bandwidth across FIFO, PIE and FQ-CoDel AQMs. We use the configuration file with the key parameters illustrated in Section III-E. Figure 11 shows the experimental setup of the testbed networks.

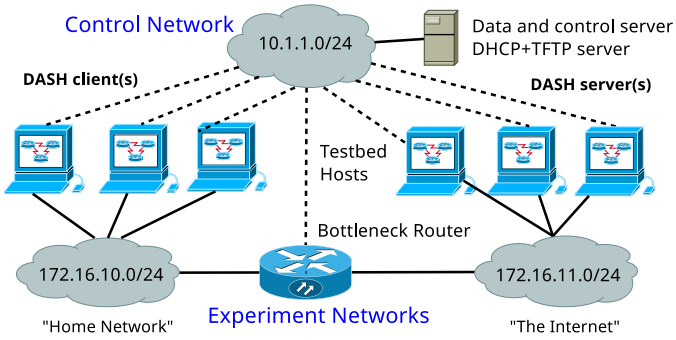


Figure 11: TEACUP testbed emulating DASH client(s) at home retrieving content from remote DASH server(s)

1) Hosts, router and bottleneck conditions: The router provides a configurable bottleneck between the client on network 172.16.10.0/24 and the server on network 172.16.11.0/24. Each host is a triple-boot machine that can run 64-bit Linux (openSUSE 12.3 with kernel 3.17.4 and Web10G patch), 64-bit FreeBSD (FreeBSD 10.1-RELEASE) or 64-bit Windows 7 (with Cygwin 1.7.25 for unix-like control of the host).

In our experiments, all end-hosts run FreeBSD NewReno and the bottleneck router runs 64-bit Linux (openSUSE 12.3 with kernel 3.10.18 patched to run at 10000Hz). The bottleneck router uses netem and tc [25] to concatenate an emulated path with specific one way delay (OWD) and an emulated bottleneck whose throughput is limited to a particular rate. Packets sit in a packet buffer while being delayed (to provide artificial OWD) and then sit in a separate "bottleneck buffer" of configurable size (in packets) while being rate-shaped to the bottleneck bandwidth.

We use Linux 3.17.4 kernel's implementations of FIFO, PIE and FQ-CoDel[6] for these experiments.

2) Traffic generators and logging: DASH flows were generated by the dash.js client requesting 10-sec video chunks from the DASH server. We use a standard lighttpd (version 1.4.35) [21] server with persistent HTTP connections enabled as our data source, serving the test dataset as described in Section III-A. TCP connection statistics were logged using SIFTR [26] under FreeBSD and Web10G [27] under Linux. Packets captured at both hosts with tcpdump were used to calculate non-smoothed end-to-end RTT estimates using CAIA's passive RTT estimator, Synthetic Packet Pairs (SPP) [28].

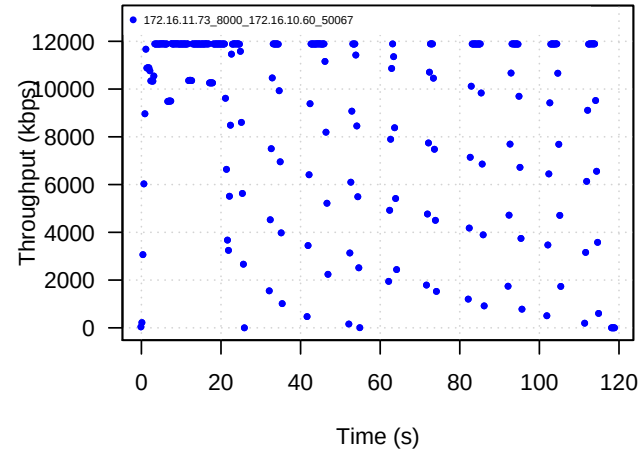
3) Measuring throughput: 'Instantaneous' throughput is an approximation derived from the actual bytes transferred during constant windows of time. Long windows smooth out the effect of transient bursts or gaps in packet arrivals. Short windows can result in calculated throughput that swings wildly from one measurement interval to the next. For these experiments we use a 0.8 sec wide window sliding forward in steps of 0.05 sec.

B. Experiment Results

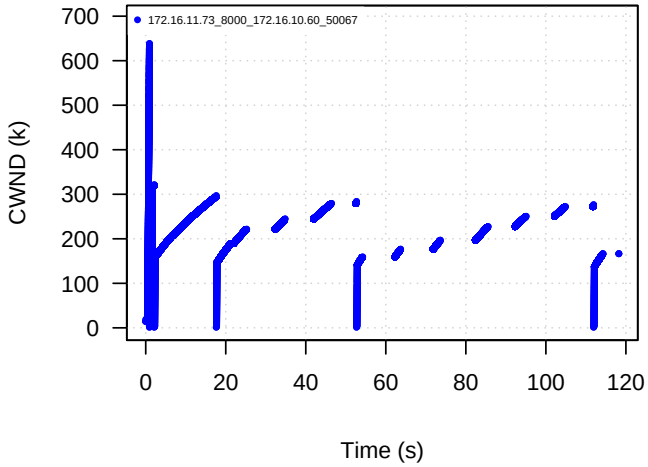
There are no modifications to TEACUP data analysis fabric tasks. Default analysis functions [10] such as analyse_throughput, analyse_rtt, analyse_ackseq, can be applied to DASH experiments. TEAPLOT [11] can also be used to animate the playback of TCP's behaviour during DASH experiments.

Figures 12 and 13 show throughput, `cwnd`, RTT and ACK sequence number over time when a DASH stream is being bottlenecked by FIFO, PIE and FQ-CoDel AQM schemes across a 12/1Mbps, 40ms base RTT path. While all FIFO, PIE and FQ-CoDel achieve similar throughput (Figures 12a, 13a, 13b) and download rate (Figures 12d, 13g, 13h), PIE and FQ-CoDel induces much less queuing delays (Figures 13e and 13f) than traditional FIFO (Figure 12c). FQ-CoDel has a tighter burst tolerance and emulates a smaller buffer by inducing packet losses more aggressively than PIE (as shown in the congestion window graphs in Figures 13d and 13c) hence it induces lower queuing delays than PIE. In these scenarios where the path delay is relatively small, FQ-CoDel and PIE allows DASH to achieve similar performance. The performance of DASH over FQ-CoDel degrades when the path delay increases, i.e when the ratio of the effective queue buffer to the path's Bandwidth Delay Product (BDP) gets smaller [29].

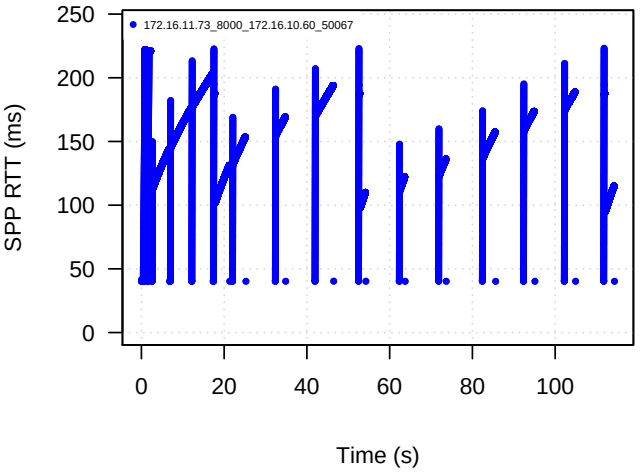
[6] CoDel results are not shown because they are similar to those of FQ-CoDel when there is only a single flow.



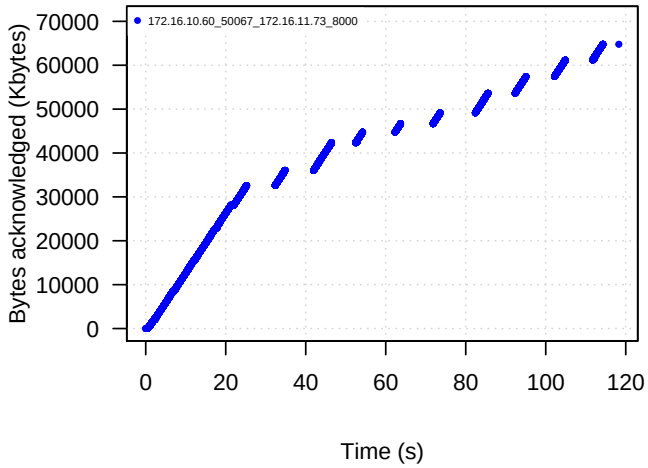
(a) Throughput vs time



(b) `cwnd` vs time

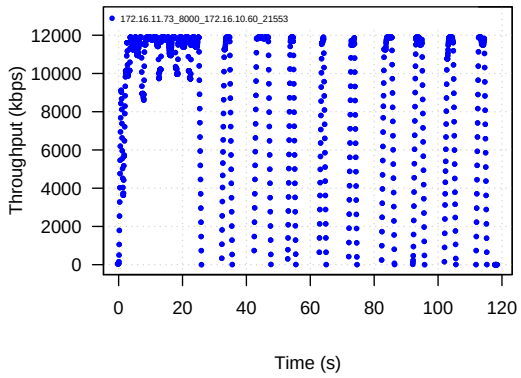


(c) SPP RTT vs time

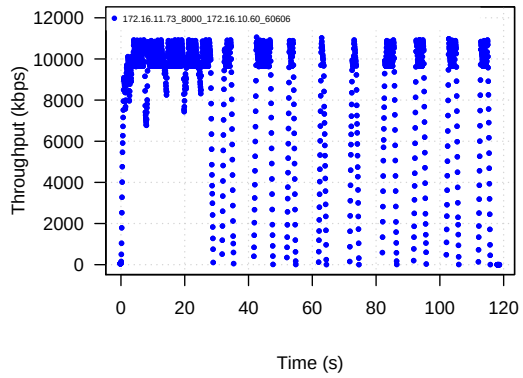


(d) ACK Sequence Number vs time

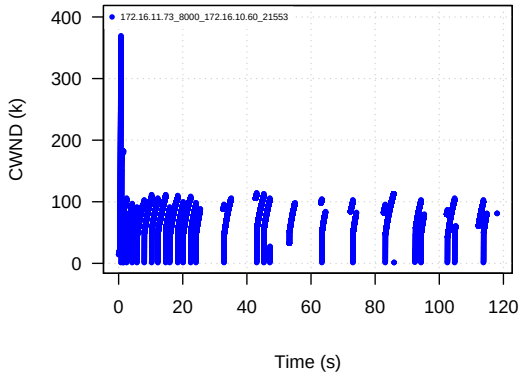
Figure 12: Single DASH stream across a 12/1Mbps, 40ms base RTT path, with a 180-packet Linux FIFO bottleneck



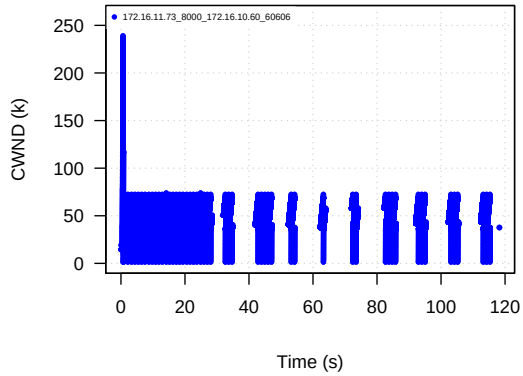
(a) PIE Throughput vs time



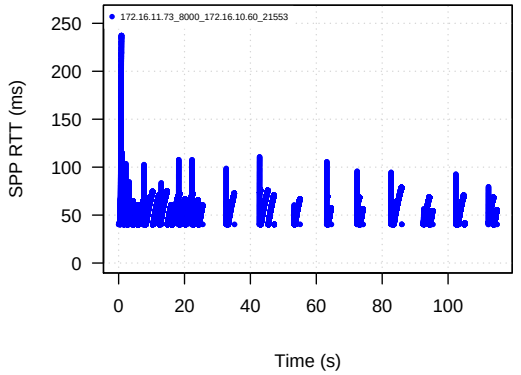
(b) FQ-CoDel Throughput vs time



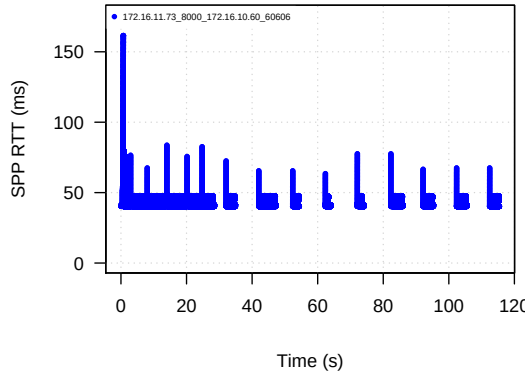
(c) PIE `cwnd` vs time



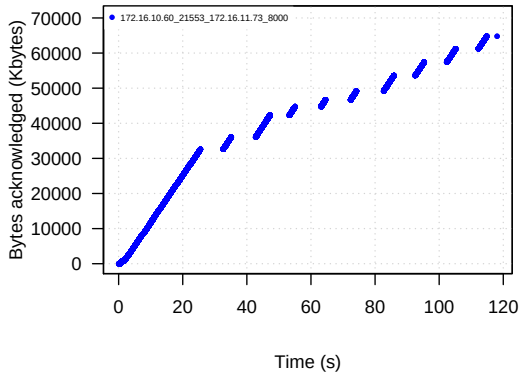
(d) FQ-CoDel `cwnd` vs time



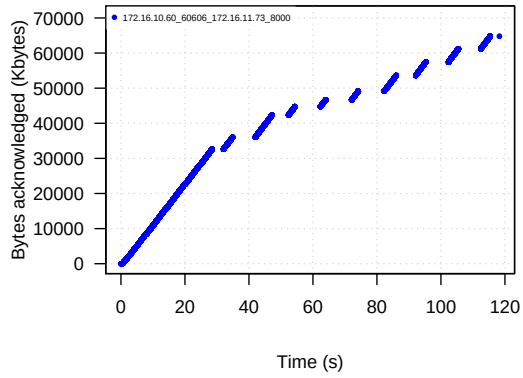
(e) PIE SPP RTT vs time



(f) FQ-CoDel SPP RTT vs time



(g) PIE ACK Sequence Number vs time



(h) FQ-CoDel ACK Sequence Number vs time

Figure 13: Single DASH stream across a 12/1Mbps, 40ms base RTT path, with a 1000-packet Linux PIE and FQ-CoDel bottleneck

V. Conclusions and Future Work

The DASH traffic generator enhancements to our TEACUP software provides a platform to explore the interactions between DASH, TCP and AQMs under various network path characteristics and system settings using either dash.js HTML5 player or VLC media player. This report provides patching and DASH client integration instructions onto FreeBSD and Linux end-hosts. We also provide experiment results generated with TEACUP testbed as to illustrate the consequences of single DASH flow over traditional FIFO, PIE and FQ-CoDel.

Future work will include extending TEACUP's data analysis functions to support per-chunk throughput, representation rates and other QoE metrics for DASH performance analysis.

Appendix

These appendices describes how to build VLC source code from development branch in TEACUP testbed hosts as setup according to [9].

A. Building VLC under FreeBSD 10.1-RELEASE

Prepare the build environment

FreeBSD default compiler `clang` is recommended for building VLC. The testbed image already has the required autotools installed, so no additional tools are required.

Get the source via git

1) Install git

```
> pkg install git
```

2) Get the source code from the tip of the development branch

```
> git clone git://git.videolan.org/vlc.git
```

3) The following instructions have been tested against `3.0.0-git Vetinari (revision 2.2.0-git-4692-g560eedb)`. To checkout this specific version:

```
> git checkout g560eedb
```

4) Bootstrapping the source tree (Note: bootstrapping requires autotools)

```
> cd vlc
> ./bootstrap
```

Tweaking VLC source code for FreeBSD

Some minor changes need to be made to build VLC under FreeBSD.

1) Edit `./vlc/src/posix/error.c`

```
./vlc/src/posix/error.c line 51
---*buf = sterror_l(num,loc)
++++*buf = strerror(num)
```

2) Patch VLC source code with the `patch-include-vlc_common.h` from FreeBSD ports tree. As root do:

```
> patch < /usr/ports/multimedia/vlc/files/
          patch-include-vlc_common.h
```

Install necessary packages / libraries

These are the minimum set of packages/libraries required to build VLC for our experiments. Install them with FreeBSD package manager.

```
> pkg install xorg
> pkg install xcb
> pkg install qt4
> pkg install sdl
> pkg install ffmpeg
> pkg install liba52
```

Configuration

`./configure` is used to check whether our system is able to compile VLC, we can also specify features in our VLC build.

1. Show various available options

```
> ./configure --help
```

2. For our experiment purposes, we configure VLC with this command:

```
> ./configure --disable-lua --disable-freerdp
              --disable-vcd --disable-taglib
              --enable-run-as-root
```

Compilation

Compile VLC with GNU make

```
> gmake
```

or to install VLC to the system, run as root:

```
> gmake install
```

For our experiments, we simply run VLC from the build directory

```
> ./vlc
```

Testing

1) A simple test after successful compilation of VLC:

```
> startx
> setenv DISPLAY :0
> cd {vlc build directory}
> dbus-run-session ./vlc
```

The VLC graphical user interface should show up.

2) A simple test of VLC DASH streaming:

```
> startx
> setenv DISPLAY :0
> cd {vlc build directory}
> dbus-run-session ./vlc -v
  http://www-itec.uni-klu.ac.at/ftp/
  datasets/DASHDataset2014/BigBuckBunny/
  10sec/BigBuckBunny_10s_simple_2014_05_09.mpd
```

Note: These instructions have been tested against:

```
1. FreeBSD clang version 3.4.1
   (tags/RELEASE_34/dot1-final 208032)
   20140512
2. GNU Make 4.1
```

B. Building VLC under Linux OpenSUSE 3.17.4-vanilla

Prepare the build environment

The testbed image already has the required `autotools` installed, so no additional tools are required.

Get the source via git

1) Install git

```
> zypper install git
```

2) Get the source code from the tip of the development branch

```
> git clone git://git.videolan.org/vlc.git
```

3) The following instructions have been tested against `3.0.0-git Vetinari (revision 2.2.0-git-4692-g560eedb)`. To checkout this specific version:

```
> git checkout g560eedb
```

4) Bootstrapping the source tree (Note: bootstrapping requires autotools)

```
> cd vlc
> ./bootstrap
```

Install necessary packages / libraries

These are the minimum set of packages/libraries required to build VLC for experiments. Install them with Linux OpenSUSE package manager.

1) Add OpenSUSE VLC repositories to the system repositories. As root do:

```
> zypper ar http://
  download.videolan.org/pub/vlc/SuSE/13.1
  VLC
```

2) Install VLC libraries dependencies from newly enabled repositories.

```
> zypper si -d vlc
```

3) Install additional packages.

```
> zypper install xorg-x11
> zypper install xorg-x11-server
```

Configuration

`./configure` is used to check whether our system is able to compile VLC, we can also specify features in our VLC build.

1) Show various available options

```
> ./configure --help
```

2) For our experiment purposes, we configure VLC with this command:

```
> BUILDCC="/usr/bin/gcc -m64 -std=c99"
  ./configure --disable-lua
    --disable-freerdp --disable-vcd
    --disable-taglib --enable-run-as-root
```

Compilation

Compile VLC with GNU make

```
> gmake
```

or to install VLC to the system, run as root:

```
> gmake install
```

For our experiments, we simply run VLC from the build directory

```
> ./vlc
```

Testing

1) A simple test after successful compilation of VLC:

```
> startx
> cd {vlc build directory}
> DISPLAY=:0 dbus-run-session ./vlc
```

The VLC graphical user interface should show up.

2) A simple test of VLC DASH streaming:

```
> startx
> cd {vlc build directory}
> DISPLAY=:0 dbus-run-session ./vlc -v
  http://www-itec.uni-klu.ac.at/ftp/
  datasets/DASHDataset2014/BigBuckBunny/
  10sec/BigBuckBunny_10s_simple_2014_05_09.mpd
```

Note: These instructions have been tested against:

```
1. gcc (SUSE Linux) 4.8.3 20140627
   [gcc-4_8-branch revision 212064]
2. GNU Make 4.0
```

References

[1] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP: Standards and Design Principles," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011, pp. 133–144. [Online]. Available: http://doi.acm.org/10.1145/1943552.1943572

[2] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous, "Network Characteristics of Video Streaming Traffic," in *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '11. New York, NY, USA: ACM, 2011, pp. 25:1–25:12. [Online]. Available: http://doi.acm.org/10.1145/2079296.2079321

[3] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao, "YouTube Everywhere: Impact of Device and Infrastructure Synergies on User Experience," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. New York, NY, USA: ACM, 2011, pp. 345–360. [Online]. Available: http://doi.acm.org/10.1145/2068816.2068849

[4] Sandvine, "Sandvine Global Internet Phenomena Report 2015," https://www.sandvine.com/downloads/general/global-internet-phenomena/2015/global-internet-phenomena-report-latin-america-and-north-america.pdf, 2015 (accessed May 2016).

[5] R. Pan, P. Natarajan, F. Baker, G. White, B. VerSteeg, M. Prabhu, C. Piglione, and V. Subramanian, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem," IETF Draft, https://tools.ietf.org/html/draft-ietf-aqm-pie-03, November 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-aqm-pie-03

[6] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management," IETF Draft, https://tools.ietf.org/html/draft-ietf-aqm-codel-02, December 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-aqm-codel-02

[7] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "FlowQueue-Codel," IETF Draft, https://tools.ietf.org/html/draft-ietf-aqm-fq-codel-03, November 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-aqm-fq-codel-03

[8] Python, "Fabric: Python Remote Execution," http://www.microsoft.com/silverlight/smoothstreaming/, 2016 (accessed October 1, 2016).

[9] S. Zander and G. Armitage, "CAIA Testbed for TEACUP Experiments Version 2," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 150210C, 10 February 2015. [Online]. Available: http://caia.swin.edu.au/reports/150210C/CAIA-TR-150210C.pdf

[10] S. Zander, G. Armitage, "TEACUP v1.0 – Data Analysis Functions," Centre for Advanced Internet Architectures, Swinburne University of Technology, Tech. Rep. 150529B, 2015. [Online]. Available: http://caia.swin.edu.au/reports/150529B/CAIA-TR-150529B.pdf

[11] I. True, G. Armitage, and P. Branch, "Teaplot v0.1: A browser-based 3D engine for animating TEACUP experiment data," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 150828A, 28 August 2015. [Online]. Available: http://caia.swin.edu.au/reports/150828A/CAIA-TR-150828A.pdf

[12] R. Castagno and D. Singer, "MIME Type Registrations for 3rd Generation Partnership Project (3GPP) Multimedia files," RFC 3839, IETF, 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3839.txt

[13] H. Garudadri, "MIME Type Registrations for 3GPP2 Multimedia Files," RFC 4393, IETF, 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4393.txt

[14] G. Tian and Y. Liu, "Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 109–120. [Online]. Available: http://doi.acm.org/10.1145/2413176.2413190

[15] ISO/IEC, "ISO/IEC 2309-1:2012 Information Technology: Dynamic Adaptive Streaming over HTTP (DASH) Part 1: Media presentation description and segment formats," http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57623, 2012.

[16] D. I. Forum, "DASH Industry Forum," http://dashif.org/.

[17] dash.js, "dash.js Player," https://github.com/Dash-Industry-Forum/dash.js/wiki.

[18] Chromium, "The Chromium Projects," https://www.chromium.org/, 2016.

[19] Mozilla, "Mozilla Firefox," https://www.mozilla.org/, 2016.

[20] Xorg, "X.org Foundation," https://www.x.org/, 2016.

[21] Lighttpd, "Lighttpd," http://www.lighttpd.net/, 2014.

[22] S. Lederer, C. Müller, and C. Timmerer, "Dynamic Adaptive Streaming over HTTP Dataset," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12. New York, NY, USA: ACM, 2012, pp. 89–94. [Online]. Available: http://doi.acm.org/10.1145/2155555.2155570

[23] C. Müller and C. Timmerer, "A vlc media player plugin enabling dynamic adaptive streaming over http," in *Proceedings of the 19th ACM International Conference on Multimedia*, ser. MM '11. New York, NY, USA: ACM, 2011, pp. 723–726. [Online]. Available: http://doi.acm.org/10.1145/2072298.2072429

[24] J. Healy, L. Stewart, and D. Hayes, "NewReno Congestion Control Algorithm," https://www.freebsd.org/cgi/man.cgi?query=cc_newreno.

[25] Linux, "The Linux Foundation - netem," http://www.linuxfoundation.org/collaborate/workgroups/networking/netem, Nov. 2009.

[26] L. Stewart, "SIFTR - Statistical Information for TCP Research," http://caia.swin.edu.au/urp/newtcp/tools.html.

[27] Web10G, "The Web10G Project," http://web10g.org/.

[28] S. Zander and G. Armitage, "Minimally-Intrusive Frequent Round Trip Time Measurements Using Synthetic Packet Pairs," in *The 38th IEEE Conference on Local Computer Networks (LCN 2013)*, October 2013.

[29] J. Kua, G. Armitage, and P. Branch, "The Impact of Active Queue Management on DASH-based Content Delivery," in *41st Annual IEEE Conference on Local Computer Networks (LCN 2016)*, Dubai, United Arab Emirates (UAE), Nov. 2016.