香港城市大學
City University of Hong Kong

Department of Computer Science

BSCCS Final Year Project 2019-2020
Interim Report I

19CS038

Performance Analysis for Different Congestion Control Algorithms on Video Streaming over LTE

(Volume __ of __)

Student Name	:	**XUE Kaiwen**
Student No.	:	**55199704**
Programme Code	:	**BSCEGU4**
Supervisor	:	**Dr XU, Hong Henry**
Date	:	**November 10, 2019**

For Official Use Only

Introduction

Nowadays, video service on the internet takes over a half of all the internet bandwidth, with video streaming on Netflix accounting for 15% of global traffic (Wakefield, 2018). The effectiveness of video streaming not only improves the experience of audience watching live shows and other online videos, but also allows more bandwidth for other applications in the same network. The evolution of mobile data technology and mobile applications makes it more appealing for users to enjoy video streaming services via smart phones and mobile data. Verizon's 4G LTE wireless broadband's download speed can reach around 5-12Mbps (4G LTE Speeds Vs. Cable, 2013), enabling mobile data users to watch video as if they are use Wi-Fi network. It is essential to consider how to utilize the bandwidth assigned to the video streaming application.

Researchers in networking systems has done much significant work to optimize the performance of video streaming. Two approaches improve this performance: adaptive bitrate streaming to request chunks by estimating what quality of video the current network condition is able to support, and transport layer protocols to further utilize the bandwidth so that the actual throughput can be closer to the bandwidth capcacity.

This project aims at building a generic testing platform and providing sample results and conclusions for researchers in this field to conclude the performance of related works, test their own implementation on the same platform, produce results and perform analysis. In this project, we build a testbed that emulates LTE networks, automatically runs videos over different transport layer protocols and adaptive bitrate streaming algorithms. We also run the testbed and record the performance using some examples of LTE traces, transport layer protocols, and video streaming algorithms. We compare the results and analyze the performance.

In the following sections, we will discuss about the existing works in adaptive bitrate streaming and transport layer protocols in Literature Review, talk about the design of our system modules, and list the schedule of this project.

Literature Review

Adaptive Bitrate Streaming

Adaptive bitrate streaming (ABR) is a technology that the video quality of a video streaming application is dynamic and determined by actual network condition. This requires two parts, first is a improved video codec that encodes in separate chunks and in different qualities (bitrates). The second one is a client video player that can require a certain chunk in a video of certain bitrate from the server. MPEG-DASH (Information technology — Dynamic adaptive streaming over HTTP (DASH), 2014) is standardized by ISO that suggests a data model which encodes a video into several different quality sets. Each of the quality set contains chunks of same playback time. It also provides an example DASH client's behaviors to request chunks from video server.

The actual strategy of estimating current network condition and making the decision of which chunk to request is performed by different ABR algorithms. Some of the examples are Festive (Jiang, Sekar, & Zhang, 2012) that implements random scheduling and stateful bitrate selection of DASH chunks, buffer-based (Huang, Johari, Mckeown, Trunnell, & Watson, 2014) that uses video buffer to estimate current network condition, pensieve (Mao, Netravali, & Alizadeh, 2017) that applies reinforcement learning to decide the chunk size to get next, etc. Dash.js (Law, 2019) is a javascript implementation of MPEG-DASH client, users can use which to include a DASH player in front end websites.

Transport Layer Protocol

Underlying transport layer protocols also serve an important role in optimizing video streaming performance. Currently, TCP (Transmission Control Protocol) is used most as the transport layer protocol (Transmission Control Protocol, 1981). TCP regulates how to add headers to packets, how to transfer packets from sender to receiver, how to deal with timeout and retransmission, flow control and congestion control and so on. Changing components in TCP will produce different transmission patterns in the network. Among the above components, congestion control is often implemented independent of other parts. Congestion control algorithms are used to restrict the sending speed of sender so that traffic in the network will not be congested. There are several state-of-art congestion control algorithms that optimizes the sending speed without causing congestions, such as Cubic (Ha, Rhee, & Xu, 2008), BBR (Cardwell, Cheng, Gunn, Yeganeh, & Jacobson, 2017), Sprout (Winstein, Sivaraman, & Balakrishnan, Stochastic Forecasts Achieve

High Throughput and Low Delay over Cellular Networks, 2013), etc. These algorithms often behave differently in different applications.
Another work worth significant noticing is QUIC (Adam Langley, 2017), which pointed out the limitations of the design of TCP and proposes a transport layer protocol over UDP. The latest google chrome version supports QUIC. As the next generation application layer protocol HTTP/3 (Bishop & Akamai, 2019) will be over QUIC, it is expected to be deployed in a greater scale in the future and even replace TCP eventually. Congestion control algorithms discussed above are to be implemented on standardized QUIC.

This project aims at using the elements in the above two sections as variants and compare the performance in different conditions. It is useful to design a testing platform to let users switch ABR algorithms, congestion control algorithms, LTE traces, and even transport layer protocol as a whole. The results produced by this platform will be a reference for researchers to grasp the performance of different transport layer protocols efficiently.

Preliminary Design, Solution, and System

The design of this project contains two parts:

1. Platform of testing video streaming performance over different transport protocols (TR), different congestion control algorithms (CC), different ABR algorithms (ABR), and different LTE network conditions (LTE).
2. Running test on combinations of a set of choices of above elements and analyze the results.

Note that the choices of TR, CC, ABR, and LTE combination sets will effect the design of the platform. Currently, our choices are:

- TR: TCP, implemented in Linux kernel; QUIC, version 46, implemented by google
- CC: Cubic; BBR, both have implementation on TCP and QUIC
- ABR: MPC; robust MPC; Reinforcement Learning, implemented in Dash.js
- LTE: Traces provided by Mahimahi (Winstein, Mahimahi, n.d.)

Modules

LTE Network Emulation

We use Mahimahi (Winstein, Mahimahi, n.d.) traces to emulate LTE network. Traces in Mahimahi contain the actual capacity of bandwidth provided by telecom companies such as

Verizon and AT&T. We will enter Mahimahi shells, run applications, and process the logs. In this module, we need to keep Mahimahi configuration (delay, queue length, loss) same while changing underlying components such as traces.

The purpose of emulating LTE network is to compare the gap of capacity and actual throughput to see how well congestion control and ABR algorithms do in terms of estimating bandwidth. The fluctuating nature of bandwidth in LTE links creates more challenges for good estimations. Moreoever, Mahimahi traces can be used to give the upper bound of ABR algorithms' performance by replacing the estimators in those algorithms with real bandwidth capacity. This provides a baseline for comparison.

Web Client and Server

After entering the emulated network, we will need to set up the client applies DASH logic that requests video chunks of different bitrates, whose decision is made by ABR algorithms. The server can be just a normal web server that serves all files, including base HTML files, style sheets, scripts, and videos chunks.

We choose google chrome as client, accompanied with an agent implemented in pensieve (Mao, Netravali, & Alizadeh, 2017) that helps to request and record information. Google chrome is able to invoke Javascript functions so that both Dash.js and agent can be integrated into one part. Necessary changes are made in Dash.js to log more information or change the implementations of ABR algorithm a little bit. For TCP, we will use Apache web server.

For QUIC, we still use google chrome, agent, and Dash.js client as discussed above. However, we choose a standalone QUIC server developed by google chrome (Playing with QUIC, n.d.) as web server. This server requires configuration on certificates and appropriate file headers for served files. When opening google chrome to connect to QUIC server, we need to perform mapping and include some flags to adapt to server's requirements.

Data Visualization

Data Visualization for this project is non-trivial. It needs to process two kinds of data files. First is Mahimahi's downlink log. From the downlink log, we can find out the actual throughput as well as bandwidth capacity in one emulation. The other one is the log given pensieve's agent, which includes bitrate (quality) choices and reward of reinforcement learning. We can also add more elements, such as bandwidth estimation, to the agent so that the log will also be able to show these information.

With these information, we can plot on different kinds of comparisons. The idea is to keep at least two of the components in TR, CC, ABR, LTE unchanged while comparing the rest. To satisfy this requirement, our design is to place log processing in a Python package and let different plotting scripts use the classes and methods defined in the package. This method can decouple data processing and actual visualization.

User Interface

We will develop a user interface, either in command line or in GUI, to let users select the variants. The underlying logic flow will be: entering Mahimahi trace, starting up web client and server, run video application, process logs, plot graphs.

Design Requirements

Automated Testing

The most essential design requirement for this project is that all tests performed on the platform must be automated. The reason is that when comparing playbacks of a video using different variants, the time must be accurated mapped to corresponding capacity or throughput. If different playbacks have different mapping, the comparison will be meaningless. Therefore, we must use a framework to run the logic follow discussed in the last section in the same time interval. We also need to let a program sleep for a few seconds to synchronize the time mapping.

Verification of Testing Accuracy

To verify if our platform provides accurate testing, we use the following metrics to ensure the authenticity of results.

1. Web clients and video applications has to be run inside Mahimahi-based emulation;
2. A connection has to be established in intended CC and TR;
3. Mapping of time and throughput must be uniform for different playbacks;
4. Log processing and plotting scripts should restrict the results in certain error bounds;

These metrics can be achieved by:

1. Run a simplest CC and TR both in and out the emulation as a baseline, then observe these data patterns. Once the new results are of the same pattern, they are accurate.

2. For TR, simply use google chrome's developer's tool to check. For CC, we should use command line tools and cross validate among results;
3. We record timestamps in different logs, and subtract the bias. If the bias is too big, drop the result;
4. We calculate the average throughput, estimation, and QoE from logs and compare them with Mahimahi's native plotting tools, which can give these information.

Plan and Schedule for Implemenation and Testing

Detailed Implementation

The whole system will be implemented on Linux Ubuntu 18.04. Shell, Pearl, and Python will be used to write testing scripts. For user interface, either Java or Python could be used. Verification will be implemented as discussed in last section.

Project Timeline

As milestones are described in design, implementation, and testing of the system, we include a gannt diagram here to illustrate a rough project timeline. Note that this part only deal with workload until the end of December. The remaining tasks (user interface and testing) will be scheduled in the end of December. All milestones and task deals are subject to change and will be included in the following reports.

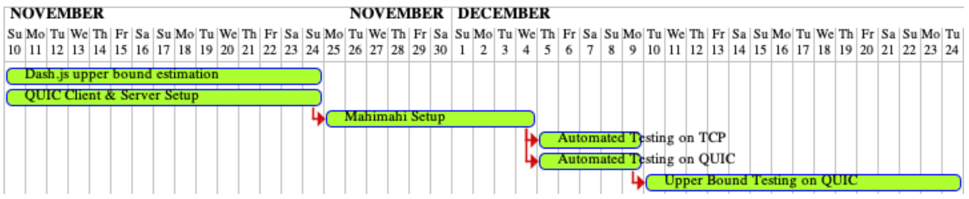

Figure 1 -- Gannt Diagram of This Project

References

4G LTE Speeds Vs. Cable. (2013, May 9). Retrieved from Verizon Wireless: https://www.verizonwireless.com/articles/4g-lte-speeds-vs-your-home-network/

Wakefield, J. (2018, October 4). *Netflix viewing eats up world's data.* Retrieved from BBC News: https://www.bbc.com/news/technology-45745362

Information technology — Dynamic adaptive streaming over HTTP (DASH). (2014, May 15). Retrieved from International Organization for Standardization: https://standards.iso.org/ittf/PubliclyAvailableStandards/c065274_ISO_IEC_23009-1_2014.zip

Jiang, J., Sekar, V., & Zhang, H. (2012). Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. *CoNEXT' 12* (pp. 97-108). Nice: ACM.

Huang, T.-Y., Johari, R., Mckeown, N., Trunnell, M., & Watson, M. (2014). A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. *SIGCOMM' 14* (pp. 187-198). Chicago: ACM.

Mao, H., Netravali, R., & Alizadeh, M. (2017). Neural Adaptive Video Streaming with Pensieve. *SIGCOMM' 17* (pp. 197-210). Log Angeles: ACM.

Law, W. (2019, 3 20). *Dash.js*. Retrieved from Github: https://github.com/Dash-Industry-Forum/dash.js/wiki

Transmission Control Protocol. (1981, September). Retrieved from IETF Standards: https://tools.ietf.org/html/rfc793

Ha, S., Rhee, I., & Xu, L. (2008). CUBIC: A New TCP-Friendly High-Speed TCP Variant. *SIGOPS' 08* (pp. 64-74). New York: ACM.

Cardwell, N., Cheng, Y., Gunn, S., Yeganeh, H. S., & Jacobson, V. (2017). BBR: Congestion-Based Congestion Control. *ACM Communication*, 58-66.

Winstein, K., Sivaraman, A., & Balakrishnan, H. (2013). Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. *NSDI' 13* (pp. 459-471). Lombard: ACM.

Adam Langley, A. R. (2017). The QUIC Transport Protocol: Design and Internet-Scale Deployment. *Sigcomm' 17* (pp. 183-196). Los Angeles: ACM.

Bishop, M., & Akamai, E. (2019, November 8). *Hypertext Transfer Protocol Version 3 (HTTP/3)*. Retrieved from QUIC Internet Draft: https://quicwg.org/base-drafts/draft-ietf-quic-http.html

Winstein, K. (n.d.). *Mahimahi*. Retrieved from Mahimahi: http://mahimahi.mit.edu/

Playing with QUIC. (n.d.). Retrieved from Chromium Project: https://www.chromium.org/quic/playing-with-quic

Appendix

Monthly logs (copied from FYPMS):

October, 2019

In this month, I have the following progress:

- Finish the experiments on streaming over TCP
- Establish a test bed where a client and server that can stream MPEG-DASH videos over QUIC

Deliverable:

- GitHub repository that contains all testing scripts and results, including images and data
- QUIC streaming test files
- Fully documented procedure of every progress so that all experiments are reproducible