Investigating the Efficacy of QUIC for Low-Latency Streaming

Matthew Walker 2310528w

March 30, 2022

1 Status Report

1.1 Proposal

1.1.1 Motivation

IPTV allows for television programs to be streamed over IP networks. Typically, the TCP transport protocol is utilised and data is buffered to allow for smooth playback in the event of packet loss. This buffering introduces a delay. The delay is not an issue for VoD content however for live broadcasts this delay can be problematic. For example, neighbours using traditional broadcast services may react to events before IPTV users have a chance to see them, which could spoil the viewing experience.

Alternatively, UDP can be used as a transport protocol to ensure low-latency playback. However, UDP is unreliable and has no form of error correction, and as such playback quality may suffer on lossy connections.

QUIC is a new transport layer network protocol recently standardised by the IETF which has support for both reliable and unreliable streams. Utilising QUIC for video streaming may be able to reduce latency while maintaining playback quality.

1.1.2 Aims

This project aims to determine how fully and partially reliable QUIC implementations compare to UDP when used for video streaming. Both latency and the quality of playback of each QUIC implementation and UDP will be compared. The final results will show whether or not QUIC can perform more reliably than UDP without impacting the latency of the stream. To accomplish this, QUIC elements will be created within the Gstreamer framework, allowing the project to take advantages of built-in Gstreamer features such as frame parsing and time-stamping. The evaluation will be performed by using Mininet to simulate a network with one or more client-server pairs.

1.2 Progress

- QUIC implementation and test framework chosen: LSQUIC will be used as the QUIC library and Mininet will be used for network tests.
- Background research conducted on QUIC transport protocol, streaming techniques and potential QUIC libraries.

- Background research conducted on Gstreamer. Familiarised myself with other network elements.
- QUIC server and sink Gstreamer elements have been created. Currently they are capable of transporting all buffers on a single stream or creating a stream for each buffer. Initial support for unreliable datagrams has been added as well. These elements required a significant amount of time to develop and test.
- Gstreamer pipeline descriptions which utilise the QUIC server and client have been created. Similar pipelines descriptions which utilise existing Gstreamer UDP elements have also been written.
- Meson build files have been created to allow Gstreamer QUIC elements to be built and installed in a single command.
- Ubuntu was installed onto my personal PC to allow Gstreamer elements to be tested.
- Initial work has begun on the Mininet testing framework. So far this includes installiation, familiarisation with Mininet and its Python API, and familiarisation with tshark.

1.3 Problems and risks

1.3.1 Problems

The following issues were encountered in the project so far.

- Initially, an Ubuntu VM was used for development and testing. However, this was actually too slow for Gstreamer to run effectively. The VM was then replaced with a dual boot Ubuntu install.
- Gstreamer is not well documented, and the existing documentation is often incomplete/outdated. I found the best way to understand some aspects was to learn from existing element implementations. I also utilised the Gstreamer mailing list when necessary.
- There were a few errors in the Gstreamer QUIC elements which resulted in dropped/corrupted frames. These took some time to debug but have now been fixed.
- I fell quite ill during the final weeks of Semester 1. This impacted my ability to work on the project during these weeks but I plan to make up for lost time over the winter break.

1.3.2 Risks

- I have no experience with Mininet. **Mitigation**: I have begun reading through Mininet's API documentation and their example code and will continue to do so over the break. If I am struggling, Colin has offered to put me in touch with a PHD student who has a solid understanding of Mininet.
- In my initial experiments I have struggled to get packet capture via tshark to work in conjunction with Mininet. **Mitigation**: I know that this is possible and will perform further research and experiments to determine how to accomplish this. I can ask on the Mininet forums if I continue to have problems.
- The current Gstreamer QUIC elements have some multi-threading issues. This causes problems such as the pipeline not closing correctly when the QUIC connection is closed. However, its not clear if there are any performance impacts. **Mitigation**: The elements will need to be reworked slightly to mend this.

- I have a general idea of how to evaluate the various QUIC element versions and the UDP elements but I am not fully sure of which metrics I will need to analyse. **Mitigation**: Perform research into which metrics will be appropriate for evaluation and determine the best way to retrieve them.

1.4 Plan

1.4.1 Winter Break

- Use Mininet's Python API to create a dumbbell network topology which connects server and client nodes via switches. **Deliverable:** One or more Python files which can instantiate the above network topology, set parameters on the link between switch nodes, start the Gstreamer pipelines on the client and server host nodes, and save the output of the Gstreamer and LSQUIC logs.
- Add support for packet capture to existing Mininet test framework. **Deliverable:** As above but with tshark packet capture capabilities.
- Select possible parameter combinations for the switch node (e.g. drop rate = 0%,1%,10%). **Deliverable:** A list of possible parameter combinations.
- Add a test runner which will run through each parameter combination for each pair of QUIC elements and the existing pair of UDP elements. **Deliverable:** A test runner written in Python which can setup, run and teardown our Mininet network.

1.4.2 Semester 2

- Weeks 1-2: Determine appropriate metrics to extract during testing and make necessary modifications to test framework to allow this. Perform a dry run of the evaluation process to identify any issues. If issues are revealed, these will then be fixed. **Deliverable:** Fully functioning test framework and results from a dry run.
- Weeks 3-4: Add support for unreliable datagram extension to QUIC Gstreamer elements. Next, create a Gstreamer element pair which can send I-frames reliably and all other frames unreliably. **Deliverable:** QUIC Gstreamer element pair which can transport I-frames reliably and all other frames unreliably.
- Weeks 5-6: Make modifications to QUIC Gstreamer elements to fix multi-threading issues described in the risks section. Also clean up code to improve readability and maintainability. **Deliverable:** Polished QUIC Gstreamer elements which are ready for evaluation.
- Weeks 7-8: Perform evaluation and analyse results. **Deliverable:** Raw data from evaluation and results of analysis of this data.
- Weeks 9-10: Write up first draft of dissertation. **Deliverable:** First draft of dissertation written and submitted to supervisor for review.
- Weeks 11-12: Make final changes to dissertation, taking supervisor's feedback into account. Use final dissertation to create a presentation of the findings of the project. **Deliverable:** Final dissertation and source code are submitted. Ready to perform final presentation