Evaluating the Effectiveness of QUIC When Integrated Into GStreamer
Requirements and Plan

Matthew Walker
2310528W

22/08/2021

1 Objectives

The goal of this project is to evaluate the effectiveness of the QUIC protocol when integrated into GStreamer. In order to accomplish this, a new GStreamer source plugin that utilises an existing implementation of the QUIC protocol will be created. Using this new plugin, the performance of QUIC will be assessed when transmitting a range of common GStreamer media types within various network conditions. The results will be compared to the performance of the TCP protocol as well as the performance of TLS 1.3 over TCP. This will show whether or not QUIC-based GStreamer elements can function appropriately and demonstrate in which scenarios, if any, QUIC is the appropriate network protocol to use in a GStreamer pipeline.

The project can be broken down into five main stages:

- The addition of missing features to the tcpclientsrc GStreamer element
- The development of a GStreamer source element plugin that operates as a QUIC client.
- The development of a test framework for comparing the TCP and QUIC element plugins.
- The evaluation of the QUIC and TCP source elements using the test framework.
- The writing of a dissertation that discusses the results of the evaluation.

1.1 Changes to the tcpclientsrc Element

To allow for a fair comparison between TCP and QUIC within GStreamer, the GStreamer tcpclientsrc element will need to be updated to use encryption. Regarding the selection of the TLS implementation, the tcpclientsrc plugin already utilises the GIO library, which supports TLS 1.3

1.2 GStreamer-QUIC Element Plugin

We wish to provide GStreamer pipelines with the ability to read data from a QUIC server.

This will require the creation of a src element that can act as a QUIC client, connecting to a server and receiving data.

1.2.1 QUIC Implementation

The QUIC implementation used by the plugin will need to provide a C API as this is the language GStreamer plugins are written in. The following implementations will be considered for the plugin:

- ngtcp2
- lsquic
- quiche
- quant
- msquic

1.2.2 Minimum Functionality

At a minimum, the client src element will be able to:

- Set a GStreamer property value indicating the servers address on initialisation.
- Set GStreamer property values to use as transport parameters for the connection.
- Establish a QUIC connection with a server (this should be created when the element moves to the READY state).
- Accept at least one uni-directional stream for this connection (this should be accepted when the element moves to the PAUSED state).
- Set flow control limits to prevent the transfer of data (used before moving to the PLAYING state and after receiving overflow messages from downstream elements).
- Read data from this stream and send this data downstream as GstBuffers (This will occur during the PLAYING state).
- Send an End-of-Stream (EOS) event downstream when all data on the stream is received.
- Abort reading of a stream and request its closure if the pipeline requests a transition to NULL or an error occurs.
- Gracefully close a connection with a server if the pipeline requests a transition to NULL or an error occurs.
- Propagate QUIC protocol errors and unexpected connection closures to GStreamer.
- Utilise loss and spin bit extensions to allow a tool monitoring the network to determine connection RTT and QUIC packet loss.

1.2.3 Additional Functionality

If time allows, the client src should also be able to:

- Migrate to a new address.
- Utilise Data Packetization Layer Path MTU Discovery extension to determine maximum packet size.
- Utilise timestamps extension to improve performance of congestion control
- Utilise explicit congestion notification

1.3 Test Framework

A set-up similar to the one used by Calle Halme to analyse the performance of modern QUIC implementations[1] will be used to compare the QUIC GStreamer plugin to its TCP equivalent.

Mininet will be used to emulate a network for the purpose of testing. This project will utilise the Mininet Python API to create a test framework that will be used to evaluate the new QUIC plugin against the existing tcpclientsrc plugin. This test framework will be deployed to a Linux VM. The following aspects of the network will be configurable by the test framework:

- Delay
- Packet loss
- Bandwidth
- Path MTU

Time allowing, the effects of multiple concurrent implementations running on the same network will also be evaluated. This will allow us to determine if one protocol is the superior choice when it is expected that multiple transfers will be occurring on a network simultaneously.

1.3.1 Network Topology

The test framework will instantiate one or more server nodes connected to a switch node. This switch node will be connected to another switch node that connects to one or more client nodes. The framework will use the link between the switches to set the current test's delay, packet loss, path MTU, and bandwidth.

1.3.2 Client-Server applications

Simple server applications which can transmit a chosen media file to a client via QUIC, TCP and TLS 1.3 over TCP protocols will need to be created. Similarly, a client application will need to be created which can instantiate GStreamer pipelines that utilise the source elements for each protocol to download and playback the transferred media file.

As the chosen QUIC implementation will be written in C, the server applications will be written in C/C++. However, the client application could be written in Python, as there are GStreamer bindings for Python which allow the instantiation of pipelines.

1.3.3 Test Sequence

Before the tests begin, the test framework will create the network topology described above. Each test will follow this sequence:

- At the start of each test, the link between the switch nodes will be configured with the appropriate parameters.
- The appropriate server application will then be started on the server node. It will then listen for connection requests from a client.
- The client application will be started next. It will initialise the appropriate GStreamer pipeline and request it moves to PLAYING.

- As the client application is started, the test framework will also start tshark to monitor the packets passing between the client and the server.
- When data transfer is complete (signified by the client application receiving an EOS message from the pipeline) the server app, client app and tshark processes will be terminated.
- The tshark packet capture logs and GStreamer logs will then be collected.

1.3.4 Test Data

tshark (WireShark's cli) will be used to monitor the transmission of packets through the network. Packet size, the quantity of packets, throughput and overall connection time can be used to compare the performance of the each protocol in different network conditions. Additionally, GStreamer logs will be gathered to determine if any underflows occurred (due to poor data throughput) during the playback of the media file being downloaded.

2 Deliverables

- A time-log of work on the project.
- Data gathered during the evaluation.
- Source code for the new QUIC client src GStreamer element.
- Source code for the updated tcpclientsrc element.
- A manual for installing these elements.
- Descriptions of pipelines utilising QUIC, TCP and TLS over TCP element plugins which can playback or store common GStreamer media types.
- A simple Python program which can instantiate the pipelines from these descriptions.
- Simple server applications which utilise QUIC, TCP and TLS 1.3 over TCP.
- Source code for the server applications.
- Source code for the test framework.
- A UML diagram describing how the components of the framework interact.
- A diagram of the mininet network topology used in testing and evaluation.
- tshark and GStreamer logs from the evaluation (Evaluation data).
- Documents providing analysis and visulisation of the evaluation data.
- A dissertation describing the results of the project.

3 Work Plan

What follows is a plan detailing when work required for the project will be carried out:

3.1 Week 1 (Start Date: 27/SEP/2021)

- Set up source control repository for project.
- Add project template to source control and insert details about the project.
- Set up Linux VM for working with GStreamer.

3.2 Week 2 (Start Date: 04/OCT/21)

- Define requirements for the project.
- Lay out plan for the project.

3.3 Week 3 (Start Date: 11/OCT/2021)

- Implement interoperability tests for potential QUIC implementations.
- Select a QUIC implementation for the project.

3.4 Week 4 (Start Date: 18/OCT/2021)

- Select the appropriate base class for GStreamer element plugin.
- Create a GStreamer element plugin that inherits from the chosen class.
- Add necessary GStreamer properties to store server address and transport parameters.
- Add an include to the plugin for the QUIC implementation to ensure the plugin can access the implementation.
- Ensure created plugin can be built and installed successfully.

3.5 Week 5 (Start Date: 25/OCT/2021)

- Add necessary functionality to allow QUIC client src element to initialise (enter NULL state).
- Add necessary functionality to allow QUIC client src element to establish a connection (enter READY state).
- Add necessary functionality to allow QUIC client src element to accept a QUIC stream from a connected server and set flow control to prevent data transfer (enter PAUSED state).

3.6 Week 6 (Start Date: 01/NOV/2021)

- Add necessary functionality to allow QUIC client src element to add data from a QUIC stream to GstBuffers and push these downstream (enter PLAYING state).
- Add necessary functionality to allow QUIC client src element to handle QUIC protocol errors and unexpected stream and connection closures.
- Add necessary functionality to allow QUIC client src element to close streams and connections early if a NULL transition is requested by the pipeline.

3.7 Week 7 (Start Date: 08/NOV/2021)

- Add extensions to the QUIC client src element (ECN, timestamps, DPLPMTUD)
- Add ability to migrate to a new address to QUIC client src element.

3.8 Week 8 (Start Date: 15/NOV/2021)

- Add TLS 1.3 support to tcpclientsrc.

3.9 Week 9 (Start Date: 22/NOV/2021)

- Create Python program to instantiate and run pipelines for each protocol.
- Create server applications for each protocol. (These can use existing example servers as a base to save time).

3.10 Week 10 (Start Date: 29/NOV/2021)

- Test that the pipelines created by the Python program can connect to the server applications.
- Fix any issues that arise during testing.

3.11 Week 11 (Start Date: 06/DEC/2021)

- Begin development of Python Mininet testing framework.
- (Time may be taken here to study for exams)

3.12 Week 12 (Start Date: 13/DEC/2021)

- Complete and hand in Status Report.

3.13 Week 13 (Start Date: 03/JAN/2022)

- Continue work on Mininet testing framework.

3.14 Week 14 (Start Date: 10/JAN/2022)

- Complete work on Mininet testing framework.

3.15 Week 15 (Start Date: 17/JAN/2022)

- Conduct a small test evaluation of the plugins for each protocol.

3.16 Week 16 (Start Date: 24/JAN/2022)

- Analyse results of initial test run and identify any issues that will require adjustments to the test framework.
- Fix any issues with the test framework.

3.17 Week 17 (Start Date: 31/JAN/2022)

- Perform full evaluation of QUIC, TCP and TLS 1.3 over TCP plugins.

3.18 Week 18 (Start Date: 07/FEB/2022)

- Analyse results, creating graphs and providing explanations.

3.19 Week 19 (Start Date: 14/FEB/2022)

- Begin writing dissertation.

3.20 Week 20 (Start Date: 21/FEB/2022)

- Continue work on dissertation

3.21 Week 21 (Start Date: 28/FEB/2022)

- Continue work on dissertation

3.22 Week 22 (Start Date: 07/MAR/2022)

- Polish dissertation.

3.23 Week 23 (Start Date: 14/MAR/2022)

- Hand in dissertation and source code.

3.24 Week 24 (Start Date: 21/MAR/2022)

- Create a presentation describing work completed and results.

References

[1] Calle Halme. Performance analysis of modern quic implementations. 2021. URL 10.1109/ICGCCEE.2014.6922233.