Telekom Quic Challenge

Science Hack 2018

Challenge

- Setup of a webserver using quic
- Compare its performance to TCP based webservers

Research

Quic is a new network protocol on the transport layer initially developed by Google and is based on UDP. It aims to improve performance of web applications compared to TCP. Its main advantages are:

- Always using encryption/TLS (even parts of the headers are encrypted)
- All handshakes are done in parallel
- Better congestion avoidance (done in user space which enables faster improvements)

The big challenge when writing quic-based applications is that there are not many working and well documented implementations. We looked at the following implementations:

Chromium

The chromium/chrome browser is already capable of handling quic as client. Furthermore, the chromium source code provides quic functionalities for both client and server side which one is supposed to play around with. There is some documentation, but it doesn't go into the details of the API but rather explains how the quic protocol works and what its strengths are. At first, it was unclear to us, which parts of chromium we had to build to use the quic features. After managing to extract the quic server and client, we set them up using the suggested settings. They both ran but couldn't connect to each other and we couldn't figure out, what the problem was.

Libquic and Protoquic

These libraries extract the quic related parts of the chromium project. The issue with them is that they don't provide any documentation themselves but refer to chromium's, which is not applicable to them. They also are no longer under active development.

Quic-go

Quic-go is an implementation of quic in Go (Google-developed language). It is under active development and provides an example client and server .One problem we faced with quic-go was that it needs Go 1.9 or higher which wasn't yet supplied in the Debian repositories, but after installing the latest version and overcoming some challenges with the way the Go workspace needs to be set up we

managed to connect the client and server to each other. Unfortunately, we had almost no knowledge of the Go language.

Caddy Server

Caddy is an open source server which has experimental quic-support provided by quic-go. In the end we used Caddy to set up a server on a virtual machine in the Open Telekom Cloud. The last problem we faced there was the certificate required for encryption which is required by quic, but by using a free domain name from freenom.com we were able to generate a certificate using Let’s Encrypt and were finally able to load a page in Chrome served via quic.

Performance Testing

To compare the performance of quic compared to TCP-based webservers we created several html files which were then served by three different caddy servers configured to use TCP/HTTP2, TCP/HTTP1.1 and QUIC. The servers were all running on the same VM and served the same files. The files contained different types of data: Some were different sized simple html files containing only text, some included a few other CSS or JavaScript-files and some required a large number of data and images to be fetched. We then set up some automated tests measuring the loading time of the servers for the different files (see separate .csv files for results). One problem we faced there was the large difference in load time when certain files were cached by the browser. Because there currently still aren’t any command line utilities supporting quic so we had to use Chrome for the tests. In the end we managed to find a Chrome extension which suppressed all the caching done by the browser.

The tests showed that quic outperforms HTTP/1.1 in almost every case but there was hardly any difference in loading time compared to HTTP/2 on TCP. The cases in which it was a bit better were when transferring many small files and when using a weak or partially blocked internet connection.