Week 3

QUIC – Quick UDP Internet Connections

QUIC initially created by Google – Chrome uses QUIC, Firefox has also enabled QUIC

Facebook has transferred to QUIC, gmail and youtube – all fairly recent

Currently

* Web is built with TCP
* 3-way handshake to start connection
* Adds significant delay to each new connection
* In TLS (transport layer security), even more packets have to be connected

3 way handshake much longer with security

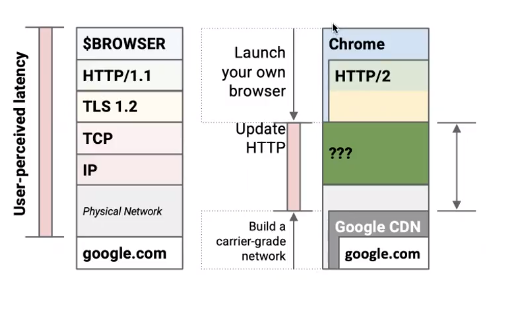
How do you make the web faster

HTTP 1 had limited resources – can only fetch one resource at a time

HTTP2 was developed by google to make things faster

SPDY & HTTP/2

* SPDY + HTTP/2: One single TCP connection instead of multiple
  + Big move in improving speed
* Downside: Head of line blocking
  + By using TCP there were some problems with SPDY
  + If a message is lost, the other messages would go through, but because it needs to be passed in the correct order all the other messages will not go through they will wait in the queue.
* In TCP, packets need to be processed in correct order



Improve this middle green section

If we don’t use TCP, we probably want to use UDP

UDP

Fire and Forget

* Less time spent to validate packets
* Downside – no reliability

Why QUIC is needed?

* Combine speed of UDP protocol with TCP’s reliability
* Very hard to make changes to TCP
* Faster to implement new protocol on top of UDP
* Roll features in TCP if they prove theory

Reducing packets is essential

* Especially with high latency links and wireless devices (LTE, 4G, etc) with RTTs of over 100ms
* Sending/receiving 4 packets (TCP/TLS can add up to 300ms)
* QUIC will try do this

QUIC

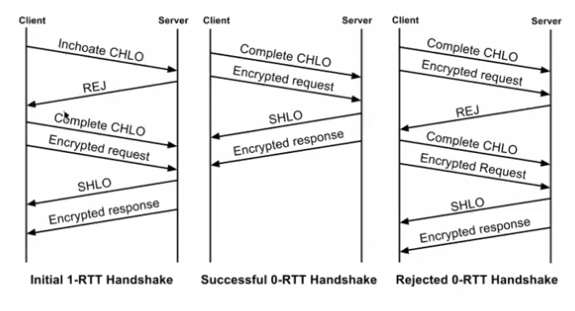
* UDP does NOT depend on order of arriving packets
* Lost packets will only impact an individual resource e.g. CSS or JS file
* QUIC is combining best parts of HTTP/2 and SPDY
  + Multiplexing on top of non-blocking transport protocol

Where does QUIC fit in?

* Replaces TLS stack and part of HTTP/2
* Implements its own crypto layer and congestion control
* Replaces TCP
* Multiplexing/connection management is handled in QUIC

Establishing a QUIC Connection

* HTTP response header
  + Alternate-Protocol: 443:quic
* Client establishes QUIC connection in the background
* Client’s can cache if server supports QUIC



Inchoate - just begun and so not fully formed or developed; rudimentary.

CHLO

Successful 0-RTT handshake – first message is complete

Successful 0-RTT handshake – the server does not recognize, or the token has expired

RTT – real time text

CHLO – Client hello

SHLO – server hello

When it REJ – it sends through all the information that the client needs

**Why would you do 1-RTT message? What are the benefits? You do this to establish the connection initially**

QUIC features

HTTP/2 Features in QUIC

* Multiplexed streams
  + Multiple streams bundled into one
* Sharing connection across domains
* HPACK header compression
  + Each request used to have excess header information
  + This meant excess bandwidth used
  + HPACK complaced the headers by removing redundant info
* Stream prioritization
* Flow control
* Server initiated streams

QUIC Features

* 0-RTT connection handshake
* 0-RTT encryption handshake
* Connections survive IP address change
* Enhanced packet loss recovery
* Always encrypted
* Mostly fixes head of line blocking
* FEC (Forward Error Correction) data recovery

**QUESTION: What sort of encryption does QUIC use?**

**Same as TLS**

Session resumption & parallel downloads

* No longer dependent on source IP of connection
* Own identifier: Connection UUID <- token from the server
* Go from WiFi to LTE and keep UUID
* Connection not dependent on IP so can easily move networks

Forward Error Correction (FEC)

* Every packet includes enough data of other packets such that missing packets can be reconstructed (similar to data drives)
* RAID 5 on network level (this is used in disc – datablock has a parity bit to check whether the data is correct) (block level parity – if one disc fails you can recover it)
* Trade off: **more payload vs. less retransmission** (look at the channel – asses this)
* Currently 10% overhead
  + For every 10 packets sent one can be reconstructed

Multiplexed Streams

* HTTP/1.1
  + 4-8 outstanding requests on 4-8 connections
  + Resource intensive on the server
* HTTP/2

Connection Sharing

* Multiple domains over one TCP connection
  + Domain must be in certificate and resolve the same IP

Prioritization

* Ability for clients to set a priority of a stream
* Dependency tree for streams
* Higher weights get more resources
* Resources proportional to the weighting

**Question: How is this dependency tree constructed**

**You look at how many objects are required by that object so that would correspond to the weighting**

Server-Initiated Streams

* Should send push before referencing resource – race condition
* Response must be cacheable
* Depending on the use case can be faster or slower
  + Browser already has the resource cached
  + Client cancelling the stream using RST\_STREAM frame

Loss Recovery

* Each QUIC carries a new packet number
  + Lost packet gets new **sequence number** and sent again
  + No retransmission-ambiguity-problem like in TCP
* **Stream offsets** in stream frames are used for delivery ordering
  + This is a departure from TCP approach where sequence number are used for both **reliability and ordering**

Flow Control

* Credit-based flow-control. A QUIC receiver advertises the window size (using byte offset)
  + Receiver periodically sends window update frames that increase the advertised offset limit for that stream
* Connection-level flow control, which limits the aggregate buffer that a sender can consume at the receiver across all streams
* Stream-level flow control, which limits the buffer that a sender can consumer on any given stream

Performance benefits

* New paradigm
  + Transport + Security in user space
  + 0-RTT
* UDP as transport layer
* Benefits
  + Faster Deployability
  + No head-of-line-blocking
* Standardization process by IETF

Future: IPv6