ELEC-E7320 Internet Protocols

Google QUIC and WebRTC

Yu Xiao
1.2.2018
Practical Arrangement

Please register at least one day earlier if you are coming to the exercise session. You can find the link from MyCourses

https://doodle.com/poll/pe3ys3sc6kbg4eys

Indicate your group number and topic, e.g. “Group X (HTTP2)”
Learning Outcomes

After this lecture, you will be able to

• Describe the key differences between HTTP1.1 and HTTP/2
• Describe the key features of SCTP, QUIC and WebRTC
Agenda

• HTTP2
• SCTP
• Google QUIC
• WebRTC
Go to http://presemo.aalto.fi/ip2

Please tick the boxes if you are not familiar with the concepts
• Pipelining, Head-of-line blocking
• Multiplexing
• Multi-homing
• Multi-streaming
Evolution of HTTP

1990: HTTP first used
1992: First image posted to web
1994: 1st secure retail transaction
1996: HTTP/1.0 specification published
1999: First viral video
2009: SPDY draft published
2012: HTTP/2 or RFC 7450 proposed standard published
2015: Httplib working group tasked with HTTP/2 creation

http://www.pewinternet.org/2014/03/11/world-wide-web-timeline/
SPDY and HTTP2

- SPDY is a networking protocol developed by Google for transporting web content
- The goal of SPDY is to reduce web page load time
- HTTP2 is built on SPDY

Source: HTTP2 explained
HTTP2 vs. HTTP1

Demo:

https://imagekit.io/demo/http2-vs-http1
How to make web faster?
HTTP/1.1 Pipelining

First-in-first-out

1 request per TCP connection

Multiple requests through one TCP connection
HTTP/2 Binary Framing Mechanism

All *communication* is performed over a single TCP connection that can carry any number of bidirectional streams.

Each stream may carry one or more messages.

Each *message* is a logical HTTP message, such as a request, or response, which consists of one or more frames.
HTTP/2 Multiplexing

- The *frame* is the smallest unit of communication that carries a specific type of data—e.g., HTTP headers, message payload.
- Frames from different streams may be interleaved and then reassembled via the embedded stream identifier in the header of each frame.
HTTP/2 Multiplexing

Source: https://cascadingmedia.com/insites/2015/03/http-2.html
Have we now solved all the Head-of-Line Blocking (HOLB) issues?

How about TCP HOLB?
TCP vs. UDP

TCP
- Slower but reliable transfers
- Typical applications:
  - Email
  - Web browsing

UDP
- Fast but non-guaranteed transfers ("best effort")
- Typical applications:
  - VoIP
  - Music streaming

Unicast

Unicast, Multicast, Broadcast
TCP HOLB

PDU 3 is blocking the head of the line.
SCTP (Stream Control Transmission Protocol)

- Can be considered as TCP++
- Two important features:
  - end-host multi-homing
  - multi-streaming
- Reliable but not necessarily in the desired order, full-duplex connections
- SCTP is used in WebRTC for the implementation and delivery of the data channel
Multi-homing

A multi-homed host has more than one network interface (more than 1 IP address)

connection

association

high availability increased reliability

Multi-streaming

- All the streams within an association are independent.
- A blocked stream does not affect the other streams in an association. (solving the head-of-line blocking issue)
- *How can multiple streams provide better responsiveness in transporting data?*
SCTP 4-way handshake

The server allocates the resource for the connection

INIT-ACK contains cookie

The server allocates the resource for the connection

SCTP permits data to be included in the COOKIE-ECHO and COOKIE-ACK packets
SCTP Connection Termination

Remove the half-closed state

No more data to be sent

TCP Connection Termination

SCTP Connection Termination

ACK

FIN

ACK

FIN

SHUTDOWN

SHUTDOWN-ACK

SHUTDOWN-COMPLETION
Process

Server

socket
bind
listen
accept

c SCTP_Sendmsg
stream 0

Data transfer

stream 1

c SCTP_Sendmsg

close

Connection termination

Client

socket
connect

c SCTP_Recvmsg

Connection establishment


c SCTP_Recvmsg

close
QUIC

• A reliable, multiplexed transport over UDP
• Open sourced in Chromium
• Always encrypted
• Runs in user-space
12 Design Goals of QUIC


1. **Widespread deployability** in today’s internet (i.e., makes it through middle-boxes; runs on common user client machines without kernel changes, or elevated privileges) **User space, over UDP**

2. **Reduced head-of-line blocking** due to packet loss **multiplexing**

3. **Low latency** (minimal round-trip costs, both during setup/resumption, and in response to packet loss) **0-RTT, FEC**

4. Improved support for mobile, in terms of latency and efficiency

5. Congestion avoidance support comparable to, and friendly to, **TCP**
12 Design Goals of QUIC

6. Privacy assurances comparable to TLS (without requiring in-order transport or in-order decryption)

7. Reliable and safe resource requirements scaling, both server-side and client-side (including reasonable buffer management and aids to avoid facilitating DoS magnification attacks)

8. Reduced bandwidth consumption and increased channel status responsiveness (via unified signaling of channel status across all multiplexed streams)

9. Reduced packet-count, if not in conflict with other goals.

10. Support reliable transport for multiplexed streams (can simulate TCP on the multiplexed streams)

11. Efficient demux-mux properties for proxies, if not in conflict with other goals.

12. Reuse, or evolve, existing protocols at any point where it is plausible to do so, without sacrificing our stated goals (e.g., consider uTP(Ledbat), DCCP, TCP minion)
QUIC: Quick UDP Internet Connections

HTTP/2
TLS
TCP

HTTP over QUIC
QUIC
TLS 1.3
TCP-like congestion control, loss recovery
UDP

IP

Source: callstats.io
QUIC vs. TCP+TLS+HTTP2

Key advantages of QUIC over TCP+TLS+HTTP2 include:

- Connection establishment latency
- Improved congestion control
- Multiplexing without head-of-line blocking
- Forward error correction
- Connection migration
1-ever connection – 1 RTT

No cached information available

First CHLO is inchoate (empty)
• Simply includes version and server name

Server responds with REJ
• Includes server config, certs, etc
• Allows client to make forward progress

Second CHLO is complete
• Followed by initially encrypted request data

Server responds with SHLO
• Followed immediately by forward-secure encrypted response data
Subsequent Connection – 0 RTT

First CHLO is complete
• Based on information from previous connection
• Followed by initially encrypted data.

Server responds with SHLO
• Followed immediately by forward-secure encrypted data
Example CHLO

CHLO
PAD : (607 bytes of padding)
SNI : 0x717569632D64566D6F2E69696A706C75732E6A70 Server Name Indicator
STK : 0x95ACC8F704D31697A396019034133971COC658688BB8323062B929930C12FA0BF5889191CD1CAC1B7AE2B7FF17AF1C2F4880BC11 Source Address Token
VER : ‘Q023’ Version
CCS : 0x399FF95340F7FEC97B26E9E7E45C71FF Common Certificate Set
NONC : 0x5452D757288EE7AC8D6709DD289D0C6BA0723C7496DE2F4F30814F64C48F436A Nonce
MSPC : 100 Max streams per connection
AEAD : ‘AESG’ AEAD algorithm
UAID : canary Chrome/40.0.2205.0 User Agent ID
SCID : 0x83F4EA5D3EDBD8CC218829BA2BFDF304E Server Config ID
SWND : 20 Server’s Initial Congestion Window
ICSL : 30 Idle connection state lifetime
KATO : 0 Keep alive Timeout
PUBS : 0x65F638326982B96A6B7251AEF0A61B568A6EE5D34FD7B2E3C46D9453B809F36 Public Key
KEXS : ‘C255’ Key Exchange methods
COPT : Connection Option
CGST : ‘QBIC’ Congestion Control Feedback Type
IRTT : 957 Estimated initial RTT in use
CFCW : 10485760 Initial session/connection flow control receive window
IFCW : 10485760 Initial flow control receive window
SFCW : 10485760 Initial stream flow control receive window
**Features/Benefits of QUIC**

**Zero roundtrip (0-RTT) connection time for returning connections.**

Comparison between TCP, TCP + TLS, and QUIC

![Diagram comparing TCP, TCP + TLS, and QUIC](https://www.callstats.io/2017/06/02/quic/)

Source modified from: Chromium blog, 2015
Congestion Control

- All packets carry new sequence numbers (easier to tell which packets are originals vs. retransmissions)
Multiplexing

Forward Error Correction (FEC)

No need to resend a missing frame!

Source: https://www.slideshare.net/shigeki_ohtsu/quic-overview
Connection Migration

• Instead of (source address, source port, destination address, destination port) a 64bit connection id (CID) is used.

• You can change IP address or ports but still keep your connection alive—that means zero connection time even if your mobile phone switches from Wi-Fi to Cellular!
WebRTC: Web Real-time Communications
WebRTC: Web Real-time Communications

- A media engine with JavaScript APIs
- Allow **direct** connection between browsers in order to exchange any type of data
- Connections between peers can be made without requiring any special drivers or plug-ins

![Diagram of WebRTC network architecture](image)
Signaling

• How do the clients know about each other?

No standard signaling protocol!

https://www.youtube.com/watch?v=2Z2PDsqgJP8
JSEP (Javascript Session Establishment Protocol)
Connect to Wi-Fi: aalto open
Use Chrome or Firefox to open https://10.100.4.191:8443
Components

- **Client**
  - Web page
- **Applications server**
  - User & room management
- **Media server**
  - Video streaming
Demo

Two phases
Media negotiation
Media exchange

Source code: https://github.com/Kurento/kurento-room
WebRTC

[XHR, SSE, WebSocket]

HTTP 1.x/2.0

Session (TLS) - optional

Transport (TCP)

Network (IP)

[RTCPeerConnection, DataChannel]

SRTP

Session (DTLS) - mandatory

ICE, STUN, TURN

Transport (UDP)

SCTP

Aalto University
3 Main APIs

- MediaStream (aka GetUserMedia)
- RTCPeerConnection
- RTCDataChannel
Main APIs

- **MediaStream** (aka GetUserMedia): get access to camera, microphone, or screen of the device

  - Can contain multiple tracks
RTCPeerConnection

- Create a connection between peers and communicate audio and video

Aalto University
ICE: Interactive Connectivity Establishment

- ICE tries to find the best path to connect peers.
- A STUN server is used to get an external network address.
- TURN servers are used to relay traffic if direct (peer to peer) connection fails.
STUN: Session Traversal Utilities for NAT

- Check the IP:port address of an incoming request (from an application running behind a NAT) and send that address back as a response.

Source: https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/
TURN: Traversal Using Relay NAT

Source: https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/
RTCDatChannel

- Mechanism for bidirectional exchange of data between peers
- Two modes of operation: reliable and unreliable
- Low latency
- Secure communication
Thank You!