

CS 725/825 & IT 725

Lecture 9

# Application Layer

---

October 25, 2024

# Issues...

---

- ▶ **Typical HTTP transaction is too short for TCP**
  - (as you will see later in the course) TCP connection reaches full throughput after several RTTs
  - many HTTP request/responses are just a few packets
  - **solution: Persistent Connections**
- ▶ **Shared fate: Head of Line (HoL) blocking**
  - (as you will see later in the course) packet loss leads to reduced throughput
  - **solution: Parallel Connections**
- ▶ These two approaches are in conflict with each other...

# Server Architecture

---

▶ Infinite loop

– `while True do`

    Accept Connection

    Process the Connection

`done`

▶ Concurrent request processing

– `while True do`

    Accept Connection,

*have someone else to* Process the Connection

`done`

# Concurrent Requests

---

- ▶ Full-blown **processes**
  - costly in terms of resources
  - independent
- ▶ **Threads**
  - more lightweight
  - shared address space between requests
- ▶ **Event-driven** approach
  - do a quick bit of processing and schedule a callback when (typically) I/O is done
- ▶ **Virtual threads**

# Event-driven approach

---

## Node.js

```
const http = require('http');

const requestListener = function (req, res) {
  res.writeHead(200);
  res.end('Hello, World!');
}

const server = http.createServer(requestListener);
server.listen(8080);
```

# HTTP/2 design goals

---

- ▶ Improve utilization
- ▶ Reduce latency
- ▶ Improve security
- ▶ Enable fine-grained control over resources

# HTTP/2 approach (1)

---

## ▶ Multiplexed connections

- limits Head Of Line (HOL) blocking and eliminates the need for concurrent TCP connections

## ▶ Resource push

- reduces latency of waiting for page rendering and subsequent resource requests
- (met with resistance from service providers)

# HTTP/2 approach (2)

---

- ▶ Support for **low-latency secure connection establishment**
  - utilizes low-latency methods to open secure connections
  - while secure connection is not mandated, many current implementations do not support insecure communication
- ▶ Explicit **bandwidth allocation** for streams within a connection
  - information received concurrently on all streams with bandwidth shared according to the set ratios
  - (still needs some work, see *RFC 9218 Extensible Prioritization Scheme for HTTP* from June 2022)



# HTTP/2 steps

---

- ▶ Secure connection is **established**
- ▶ Individual **streams** are set up
- ▶ **Requests** dispatched
- ▶ Information **received concurrently** on all streams with **bandwidth shared** according to the set ratios

# QUIC motivation

---

- ▶ HTTP/2 is trying to match the **performance characteristics** of the underlying transport layer protocol (TCP) and the **needs of the application** protocol (HTTP)
  - for example, consider the interaction between TCP Slow Start and typically short HTTP data.
  - HTTP attempts to address this by various methods, such as persistent “Keep-Alive:” connections, reducing the number of RTTs required to open a secure connection, or opening multiple simultaneous connections. While these solutions improve performance, they do not address the core issues with TCP.
- ▶ **Better solution:** design an alternative transport protocol

# QUIC deployment

---

- ▶ Changing a widely-used protocol is a complex task!
  - Many lessons were learned from the transition to IPv6 that started more than 20 years ago and is still far from being done.
- ▶ At least, we do not have to worry about the network itself (network layer), only the end points...
- ▶ ... and, it turns out that **Google** (at least in the US) controls the most popular **browser** (Chrome) and provides some of the most significant **web applications** (search, maps, video, email, storage, web application infrastructure)

# QUIC - a silent revolution

---

- ▶ **Requirement 1**: a way to **negotiate an alternative protocol** that will not break existing protocols and allows a clean fallback on the traditional protocols.
  - alt-svc: HTTP response header
- ▶ **Requirement 2**: must be **based on an existing transport layer protocol** so that no changes to the protocol stack of the operating systems is required.
  - standard UDP
- ▶ These allow **seamless incremental deployment** that improves performance but does not disrupt

# HTTP/3

---

- ▶ A protocol formerly known as **Hypertext Transfer Protocol (HTTP) over QUIC**
- ▶ The latest major revision of HTTP
  - HTTP/1.1 → HTTP/2 → **HTTP/3**
- ▶ Standardization:
  - QUIC: [RFC 9000](#) (May 2021)
  - HTTP/3: [RFC 9114](#) (June 2022)

# HTTP/3 deployment

---

- ▶ First connection over TCP to port 443:

```
% curl -I https://google.com
HTTP/2 301 ← Moved Permanently
location: https://www.google.com/
content-type: text/html; charset=UTF-8
date: Tue, 16 Mar 2024 17:58:05 GMT
expires: Thu, 15 Apr 2024 17:58:05 GMT
cache-control: public, max-age=2592000
server: gws
content-length: 220
x-xss-protection: 0
x-frame-options: SAMEORIGIN
alt-svc: h3-29=":443"; ma=2592000,h3-T051=":443"; ma=2592000,h3-Q050=":443";
ma=2592000,h3-Q046=":443"; ma=2592000,h3-Q043=":443"; ma=2592000,quic=":443";
ma=2592000; v="46,43"
```

- ▶ Subsequent connections
  - UDP packets sent to port 443

# Electronic Mail - SMTP

---

- ▶ Asynchronous message delivery
  - delivers robustness and reliability
- ▶ Two types of agents:
  - User Agent (UA)
  - Message Transfer Agent (MTA)
- ▶ Two types of interactions (and protocols):
  - MTA to MTA
  - UA to MTA

# TRADITIONAL MAIL DELIVERY

(Users use UAs running on the same system as MTAs and share system authentication)

