15-441 15-641 Computer Networking

Lecture 28: The Web Today

Justine Sherry

Peter Steenkiste

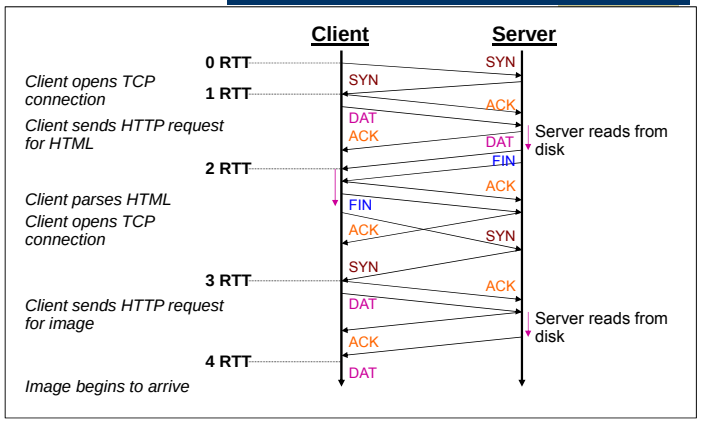
Fall 2017

www.cs.cmu.edu/~prs/15-441-F17

Outline

- HTTP 2 – what is new?
- QUIC – a new transport
- SDN – a centralized approach to network control

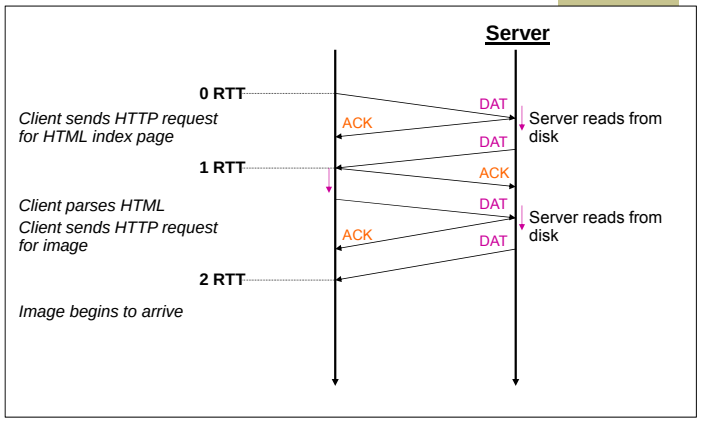
HTTP 1: Single Transfer Example



HTTP 1.1

- Multiple GETs over one TCP connection
  - Avoid handshake and slow start when getting multiple objects from same server
- Transfers can be pipelined, i.e., multiple outstanding requests
  - Requests are handled in FIFO order by server
- How to identify requests/responses?
  - Delimiter → Server must examine response for delimiter string
  - Content-length and delimiter → Must know size of transfer in advance
  - Block-based transmission → send in multiple length delimited blocks

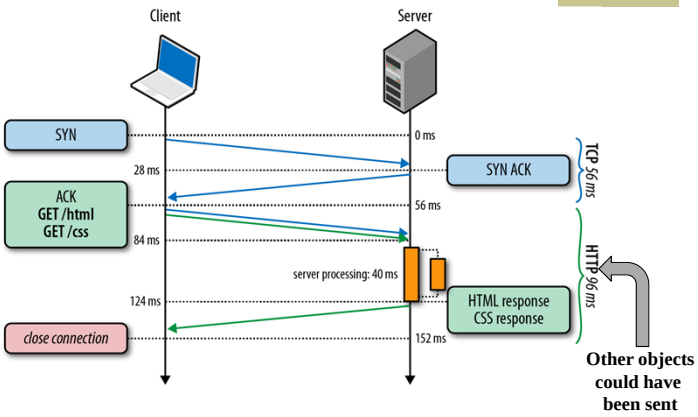
Persistent Connection Solution



Some Challenges with HTTP 1.1

- Head of line blocking: "slow" objects delay later requests
  - E.g., objects from disk versus objects in cache
- Browsers open multiple TCP connections to achieve parallel transfers
  - Increases throughput and reduces impact HOL blocking
  - Increases load on servers and network
- HTTP headers are big
  - Cost higher for small objects
- Objects have dependencies, different priorities
  - Javascript versus images
  - Extra RTTs for "dependent" objects

Example of Head of Line Blocking

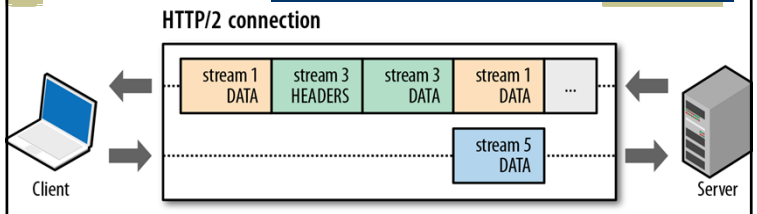


Source: http://chimera.labs.oreilly.com/books/1230000000545/ch11.html

HTTP 2.0 to the Rescue

- Responses are multiplexed over single TCP connection
  - Server can send response data whenever it is ready
  - "Fast" objects can bypass slow objects – avoids HOL blocking
  - Fewer handshakes, more traffic (help cong. ctl., e.g., drop tail)
- Multiplexing uses prioritized flow controlled streams
  - Urgent responses can bypasses non-critical responses
  - ≈ multiple parallel prioritized TCP connections
- HTTP headers are compressed
- A PUSH features allows server to push embedded objects to the client without waiting for a client request
  - Avoids an RTT
- Default is to use TLS – fall back on 1.1 otherwise

HTTP/2 Multi-Streams Multiplexing



Bit	+0..7	+8..15	+16..23	+24..31
0	Length			Type
32	Flags			
40	R	Stream Identifier		
...	*Frame Payload*			

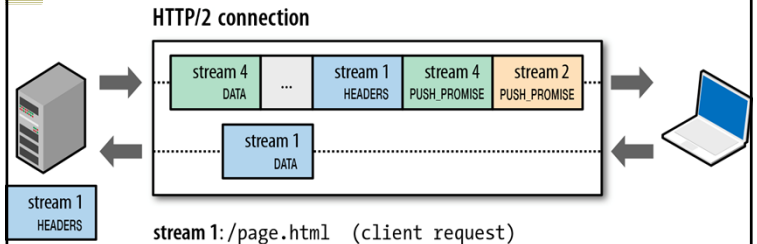
HTTP/2 Binary Framing

https://tools.ietf.org/html/rfc7540

Multiplexing

- Traffic sent as frames over prioritized streams
  - Frames types: headers, data, settings, window updates and push promise
- Sender sends high priority frames first
  - Frames are pulled from a queue when TCP is ready to accept more data
  - Reduces queueing delay
- Each stream is flow controlled
  - Receiver opens window faster for high priority streams
  - Replicates TCP function but at finer granularity
- Clearly adds complexity to HTTP library

HTTP/2 Server Push




```
stream 1: /page.html   (client request)
stream 2: /script.js   (push promise)
stream 4: /style.css   (push promise)
```


HTTP 2 PUSH Features

- Server can "push" objects that it knows (or thinks) the client will need
  - Avoids delay of having client parse the page and requesting the objects (> RTT)
- But what happens if object is in the client cache – Oops!
  - Server sends PUSH_PROMISE before the PUSH
  - Client can cancel/abort the PUSH
- How does server know what to PUSH?
  - Very difficult problem with dynamic content
  - Javascripts can rewrite web page – changes URLs
- Also: benefits limited to objects from the origin server

Outline

- HTTP 2 – what is new?
- QUIC – a new transport
- SDN – a centralized approach to network control

QUIC: A User-Level Transport

- QUIC is a user-level, UDP-based reliable transport
  - SPDY is a Google HTTP 1.1 replacement
  - Supports HTTP 2, which is based on Google's SPDY

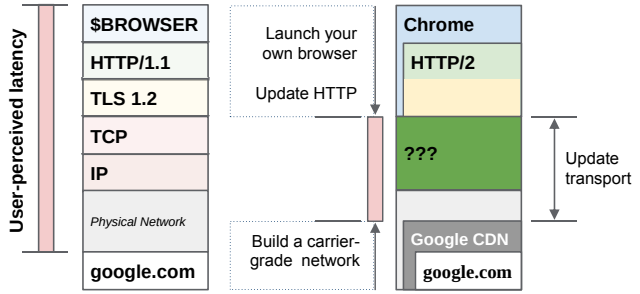


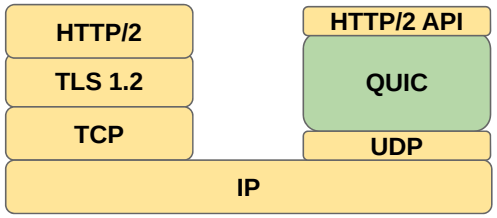
Figure and some slides based on https://www.chromium.org/quic

What is QUIC?

New transport designed to reduce web latency

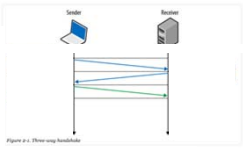
- TCP + TLS + HTTP 2 over UDP
- Faster connection establishment than TLS/TCP
  - 0-RTT usually, 1-RTT sometimes
- Deals better with packet loss than TCP
- Has Stream-level and Connection-level Flow Control
  - I.e., TCP plus HTTP 2 level flow control
- Always encrypted
- User-level implemention allows faster evolution
  - E.g., new congestion control modules
- FEC recovery
- Multipath

Where does it fit?

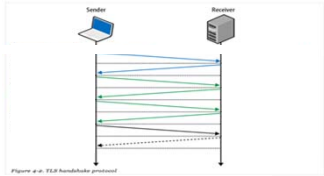


0-RTT connection establishment

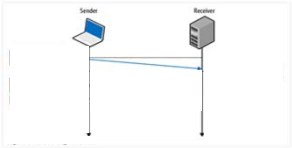
TCP



TCP + TLS

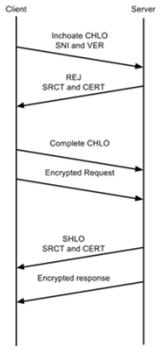


QUIC
(equivalent to TCP + TLS)



First-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



Incomplete Client Hello + Tags

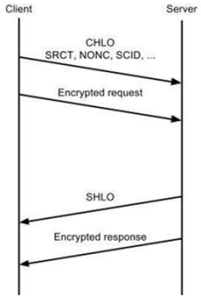
Reject + Missing Info

New Handshake + Request

Complete Handshake + Response

Subsequent connections - 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



Congestion control & reliability

- QUIC builds on decades of experience with TCP
- Incorporates TCP best practices
  - TCP Cubic - fair with TCP
  - FACK, TLP, F-RTO, Early Retransmit...
- More flexibility going forward
  - User level implementation, modular desgn
  - Improved congestion feedback, control over acking, FEC, …
- Uses packet sequence numbers to simplify signaling
  - Packets carry frames with Detailed “TCP like” information
- Better signaling than TCP

Better signaling than TCP

- Retransmitted packets consume new sequence number
  - No retransmission ambiguity
  - Prevents loss of retransmission from causing RTO
- More verbose ACK
  - TCP supports up to 3 SACK ranges
  - QUIC supports up to 256 NACK ranges
  - Per-packet receive times, even with delayed ACKs
- ACK packets consume a sequence number
- Connections are associated with 64 bit connection ids
  - TCP uses 5-tuples
  - Helps with mobility, MP-TCP, ..

Deployment Issues: Client-side protection

- What if UDP is blocked?
  - Chrome seamlessly falls back to HTTP/TCP
- What if the path MTU is too small?
  - QUIC handshake fails, Chrome falls back to TCP
- What if a client doesn't want to use QUIC?
  - Chrome flag / administrative policy to disable QUIC

When client-side protection is not enough...

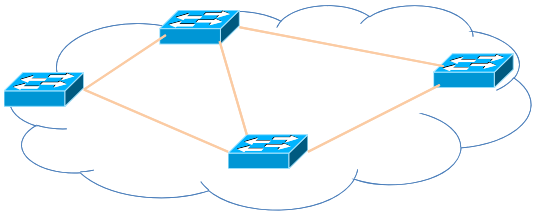
- As a last resort, Google disables QUIC to specific ASNs
  - This is used as a fallback to protocol features
- Why do we disable QUIC delivery?
  - Degraded quality of experience measured
  - Indications of UDP rate limiting at peak times of day
  - End user reports (via chromium.org)

Outline

- HTTP 2 – what is new?
- QUIC – a new transport
- SDN – a centralized approach to network control
  - Based on slides by Jennifer Rexford, Princeton

Traditional Computer Networks

Data plane:
Packet streaming

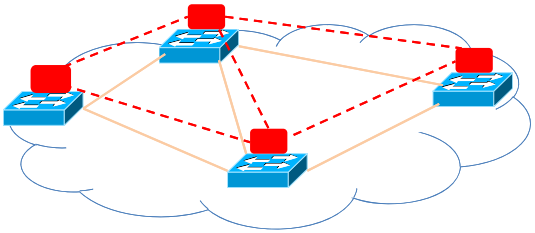


Forward, filter, buffer, mark, rate-limit, and measure packets

Traditional Computer Networks

Control plane:
Distributed algorithms

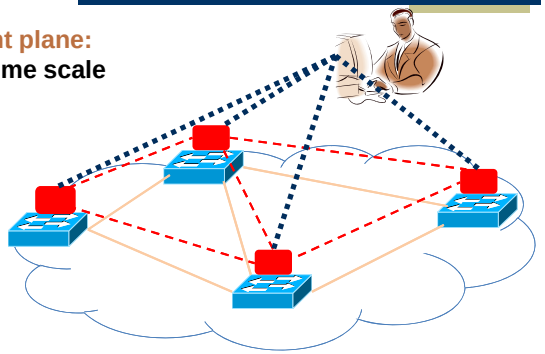
Good scalability
No single point of failure



Track topology changes, compute routes, install forwarding rules

Traditional Computer Networks

Management plane:
Human time scale

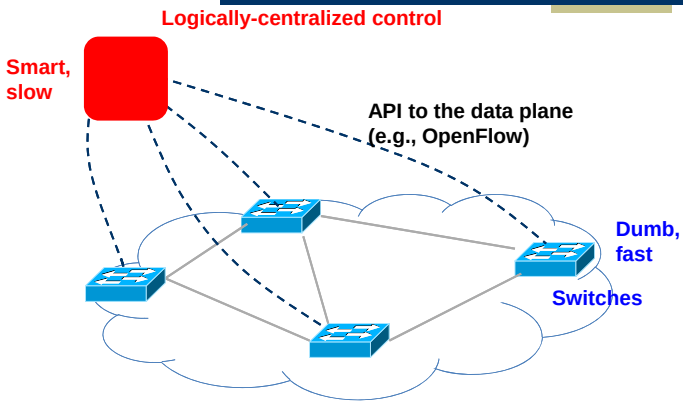


Collect measurements and configure the equipment

Death to the Control Plane!!!

- Simpler management
  - No need to "invert" control-plane operations
- Faster pace of innovation
  - Less dependence on vendors and standards
- Easier interoperability
  - Compatibility only in "wire" protocols!!
- Simpler, cheaper equipment
  - Minimal software

Software Defined Networking (SDN)



Data-Plane in OpenFlow: Simple Packet Handling

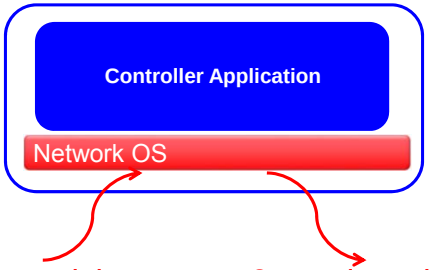
- Simple packet-handling rules
  - Pattern: match packet header bits
  - Actions: drop, forward, modify, send to controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets

1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward(2)
3. src=10.1.2.3, dest=*.*.*.* → send to controller

Unifies Different Kinds of Boxes

- Router
  - Match: longest destination IP prefix
  - Action: forward out a link
- Switch
  - Match: destination MAC address
  - Action: forward or flood
- Firewall
  - Match: IP addresses and TCP/UDP port numbers
  - Action: permit or deny
- NAT
  - Match: IP address and port
  - Action: rewrite address and port

Controller: Programmability



Events from switches
Topology changes,
Traffic statistics,
Arriving packets

Commands to switches
(Un)install rules,
Query statistics,
Send packets

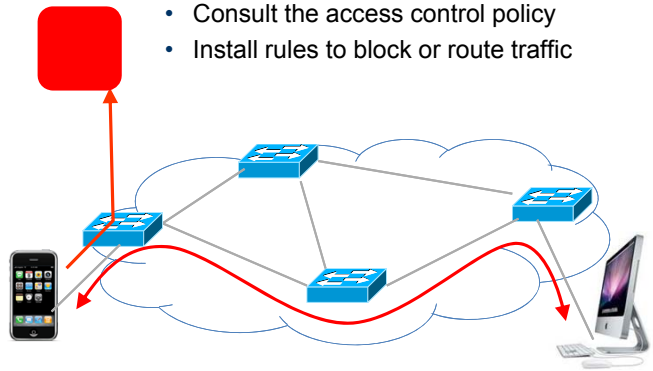
Example OpenFlow Applications

- **Dynamic access control**
- **Seamless mobility/migration**
- **Server load balancing**
- **Network virtualization**
- Using multiple wireless access points
- Energy-efficient networking
- Adaptive traffic monitoring
- Denial-of-Service attack detection

See http://www.openflow.org/videos/

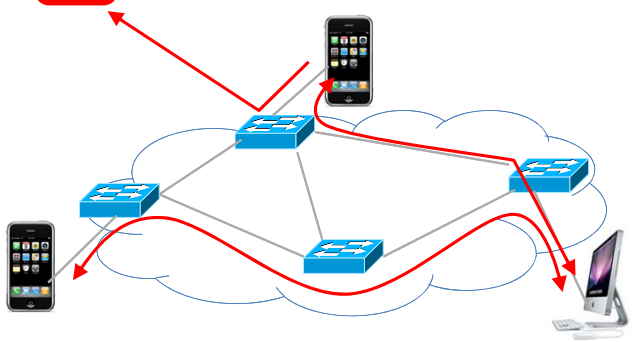
E.g.: Dynamic Access Control

- Inspect first packet of a connection
- Consult the access control policy
- Install rules to block or route traffic



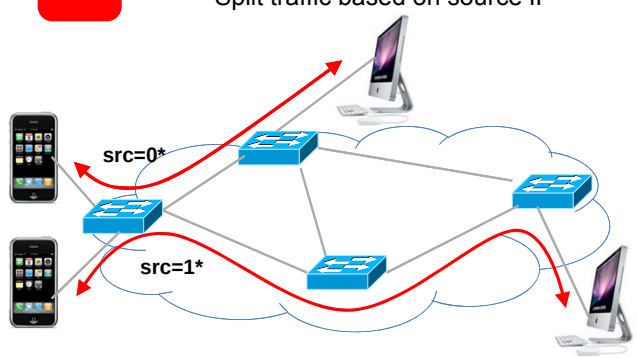
E.g.: Seamless Mobility/Migration

- See host send traffic at new location
- Modify rules to reroute the traffic



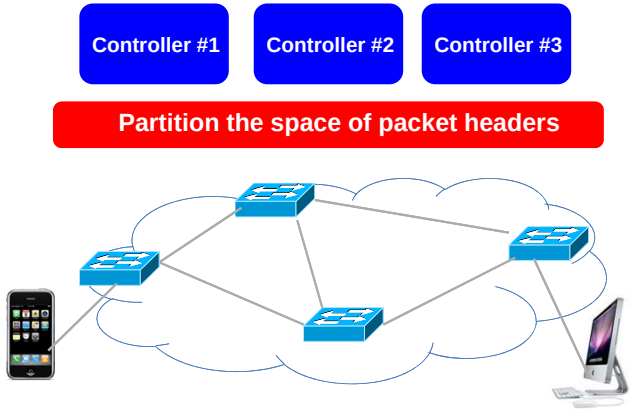
E.g.: Server Load Balancing

- Pre-install load-balancing policy
- Split traffic based on source IP



36

E.g.: Network Virtualization



37

OpenFlow in the Wild

- Open Networking Foundation
  - Google, Facebook, Microsoft, Yahoo, Verizon, Deutsche Telekom, and many other companies
- Commercial OpenFlow switches
  - HP, NEC, Quanta, Dell, IBM, Juniper, …
- Network operating systems
  - NOX, Beacon, Floodlight, Nettle, ONIX, POX, Frenetic
- Network deployments
  - Eight campuses, and two research backbone networks
  - Commercial deployments (e.g., Google backbone)

The End!

- Reminder: review session tomorrow

39