Lecture 6

Chapter 2: Application Layer
2.2 Special section on HTTP 2.0
2.3 SMTP - Email
2.4 DNS
2.5 P2P
2.6 Video Streaming

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

- Special Section on HTTP/2.0

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

2.7 socket programming with UDP and TCP

Special section on HTTP, WebSocket, SPDY and HTTP/2.0

From Simon Bordet (sbordet@intalio.com)

Presented at Jfokus 2013

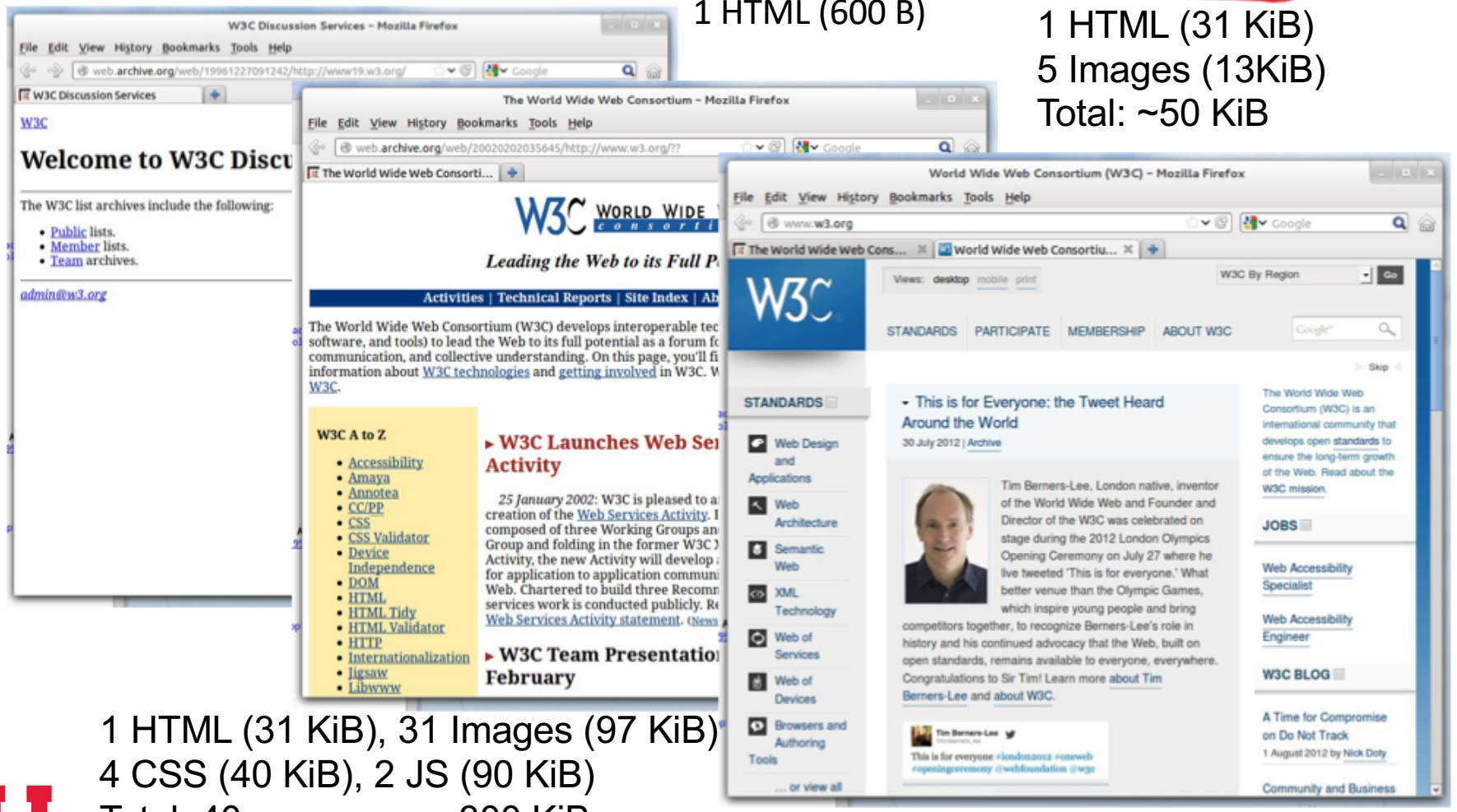
We'll quickly cover

- Some History of Web Protocols
- WebSocket – Bidirectional Web
- SPDY – A better Web
- HTTP 2.0 – the Future

w3c.org 1996, 2002, 2012

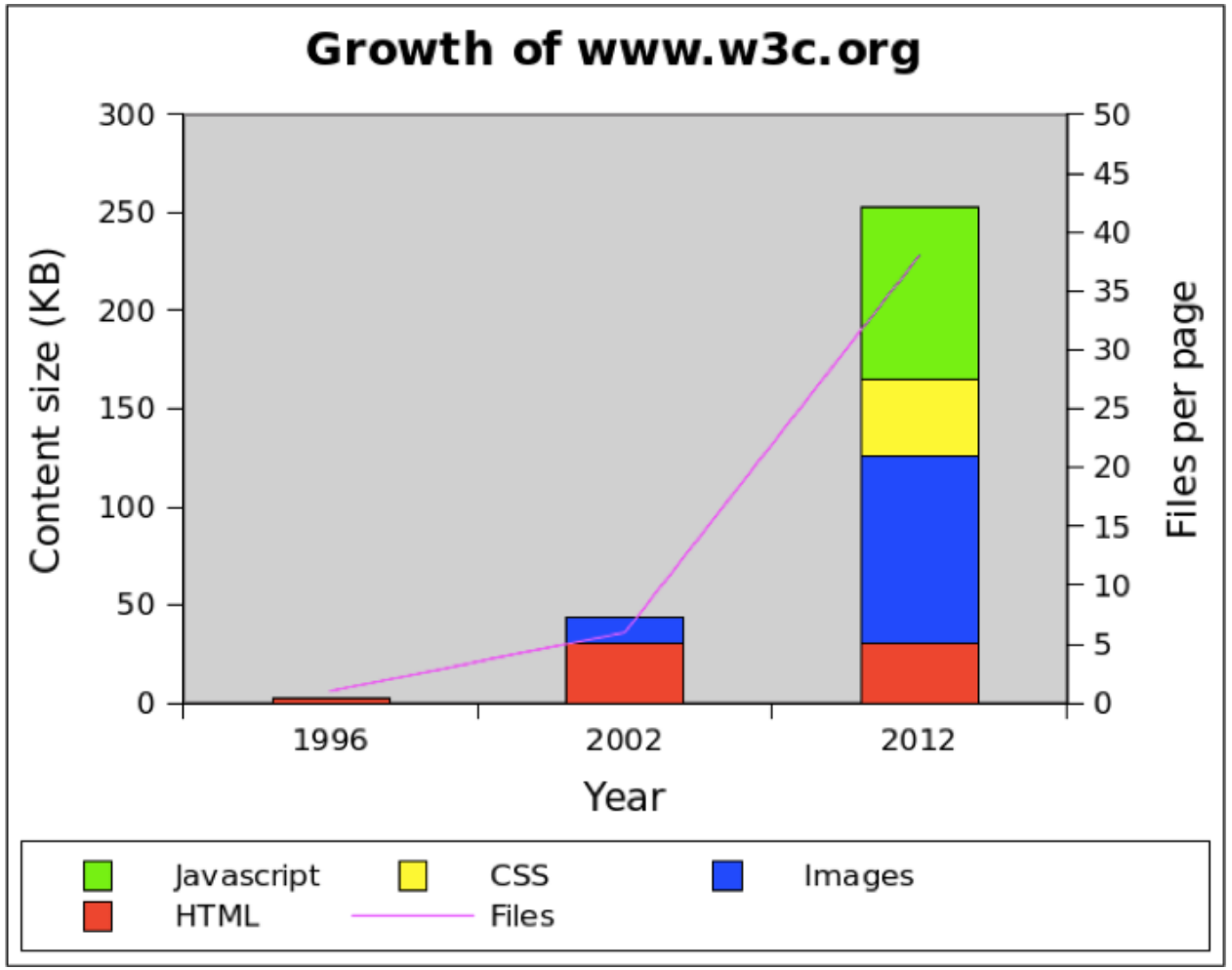
1 HTML (600 B)

1 HTML (31 KiB)
5 Images (13KiB)
Total: ~50 KiB

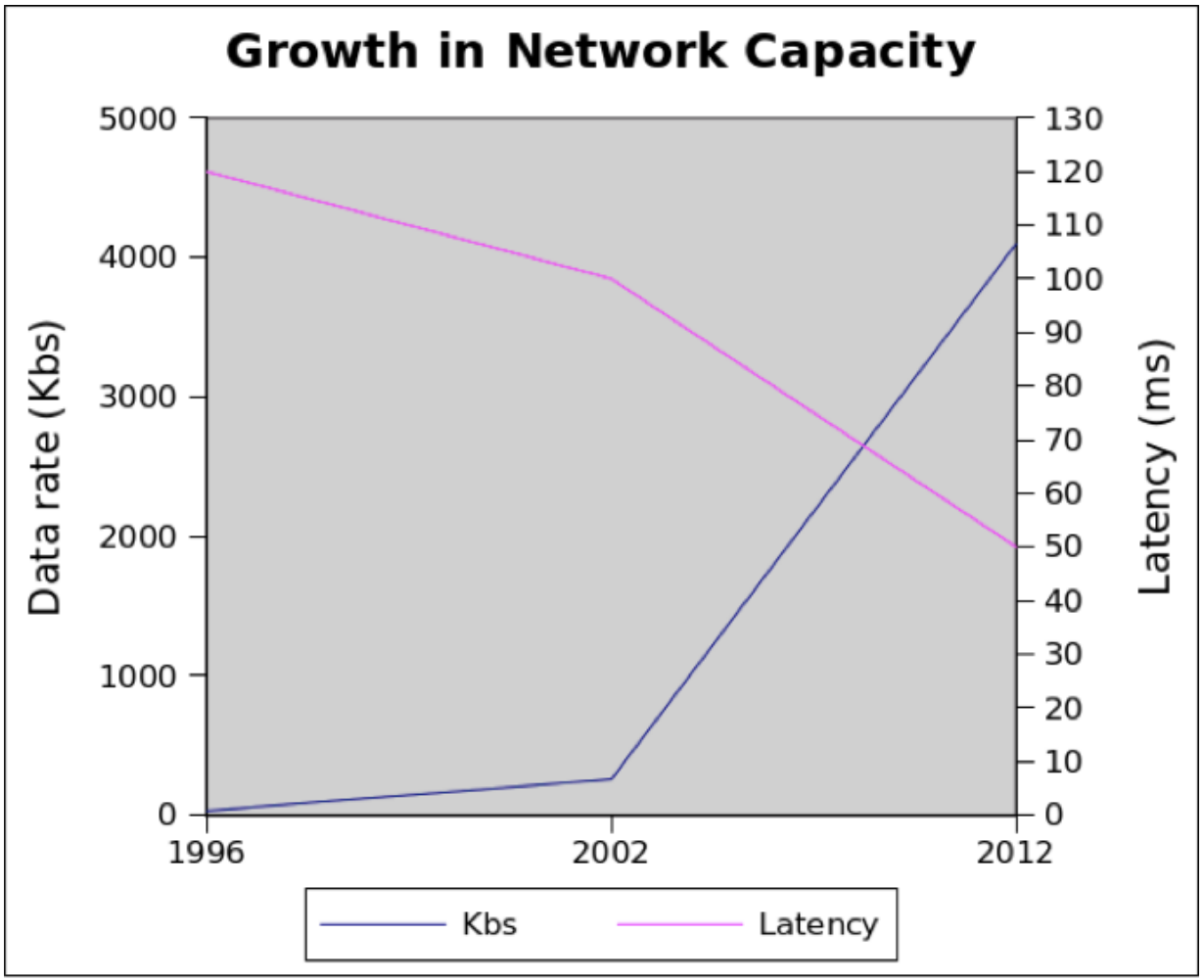


1 HTML (31 KiB), 31 Images (97 KiB)
4 CSS (40 KiB), 2 JS (90 KiB)
Total: 40 resources, ~300 KiB

Growth of w3c.org



Growth network capacity



Content Vs Network Growth with HTTP/1.0

Year	1996	2012
Protocol	HTTP/1.0	HTTP/1.0
Feature		
max Connections	2	2
Connections	1	38
Data/connection (KB)	2	7
Requests per connection	1	1
establish (ms)	*120*	*950*
requests (ms)	*120*	*950*
data (ms)	*556*	*495*
Slow start (ms)	*278*	*247*
Load time (ms)	1073	2642

Content Growth >> Network Growth

Improved HTTP

Year	1996	2012	2012	2012
Protocol	HTTP/1.0	HTTP/1.1	HTTP/1.1	HTTP/1.1
Feature			Pipeline	
max Connections	2	2	2	**6**
Connections	1	2	2	6
Data/connection (KB)	2	127	127	42
Requests per connection	1	19	19	6
establish (ms)	*120*	*50*	*50*	*50*
requests (ms)	*120*	*950*	*50*	*317*
data (ms)	*556*	*495*	*495*	*495*
Slow start (ms)	*278*	*16*	*16*	*47*
Load time (ms)	1073	1510	610	908

38/2 = 19 round trips

Semantic issues turned off by default

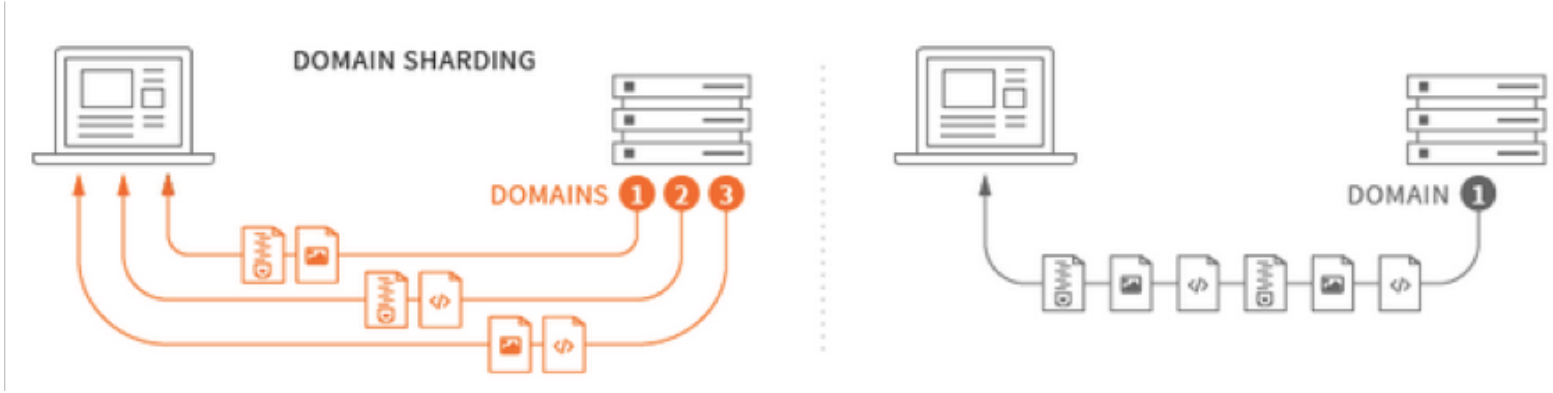
Connections have a cost

Multiple Connection Issues

- Server resources
  - Buffers and kernel data
  - Threads or selectors
- Load balancers must be sticky
  - Avoid distributed session
- Difficult concurrency issues
  - Multiple simultaneous accesses
  - Maybe out of order via different paths
- Starts an arms race
  - If 6 are good, surely 12 are better?
  - Already happening with domain sharding!

What is Domain Sharding?

- Technique for splitting resources across multiple domains
  - Improves page load time
  - Improves search engine visibility
  - Allows browser to download more resources



https://www.maxcdn.com/one/visual-glossary/domain-sharding-2/

New Protocols Needed

- Content Growth continues > Network Growth
  - HTTP has taken the easy wins
  - HTTP does not support the rich semantics needed
  - We've started to “bend the rules”
- The natives are getting restless
  - New & Experimental protocols have been deployed!
  - Websocket
    - New Semantics
  - SPDY
    - Better Efficiency

WebSocket

- Effort Initiated by Browser Vendors to
  - Bidirectional low latency communications
  - Replace Comet HTTP “hacks”
  - Implement rich web applications within the browser
    - JavaFX? Silverlight? Flash?
- Standardized
  - IETF - RFC6455 wire protocol
  - W3C - HTML5 Javascript API
  - JCP - JSR356 Java API (in progress)

WebSocket is a JavaScript API

- interface WebSocket : EventTarget
- {
- readonly attribute DOMString url;

- attribute EventHandler onopen;
- attribute EventHandler onerror;
- attribute EventHandler onclose
- attribute EventHandler onmessage;

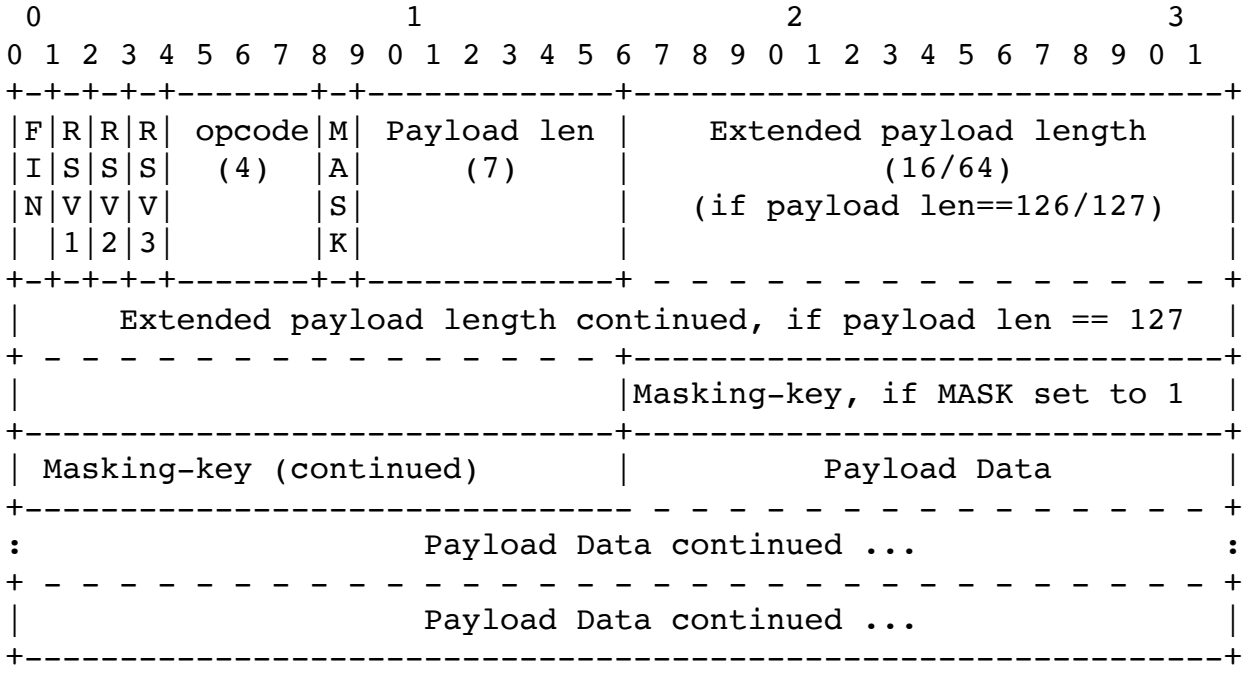
- void send(DOMString data);
- void send(Blob data);
- void close(...);
- };

WebSocket Upgrade

- Runs on port 80 or 443 for wss (secure)
- Uses HTTP/1.1 Upgrade Mechanism to explore if browser/server are capable of WebSocket
- REQUEST
- GET / HTTP/1.1
- Host: localhost:8080
- Origin: http://localhost:8080
- Connection: Upgrade
- Upgrade: websocket
- Sec-WebSocket-Key: SbdIETLKHQ1TNBLeZFZS0g==
- Sec-WebSocket-Version: 13
- RESPONSE
- HTTP/1.1 101 Switching Protocols
- Connection: Upgrade
- Upgrade: websocket
- Sec-WebSocket-Accept: y4yXRUolfnFfo3Jc5HFqRHNgx2A=

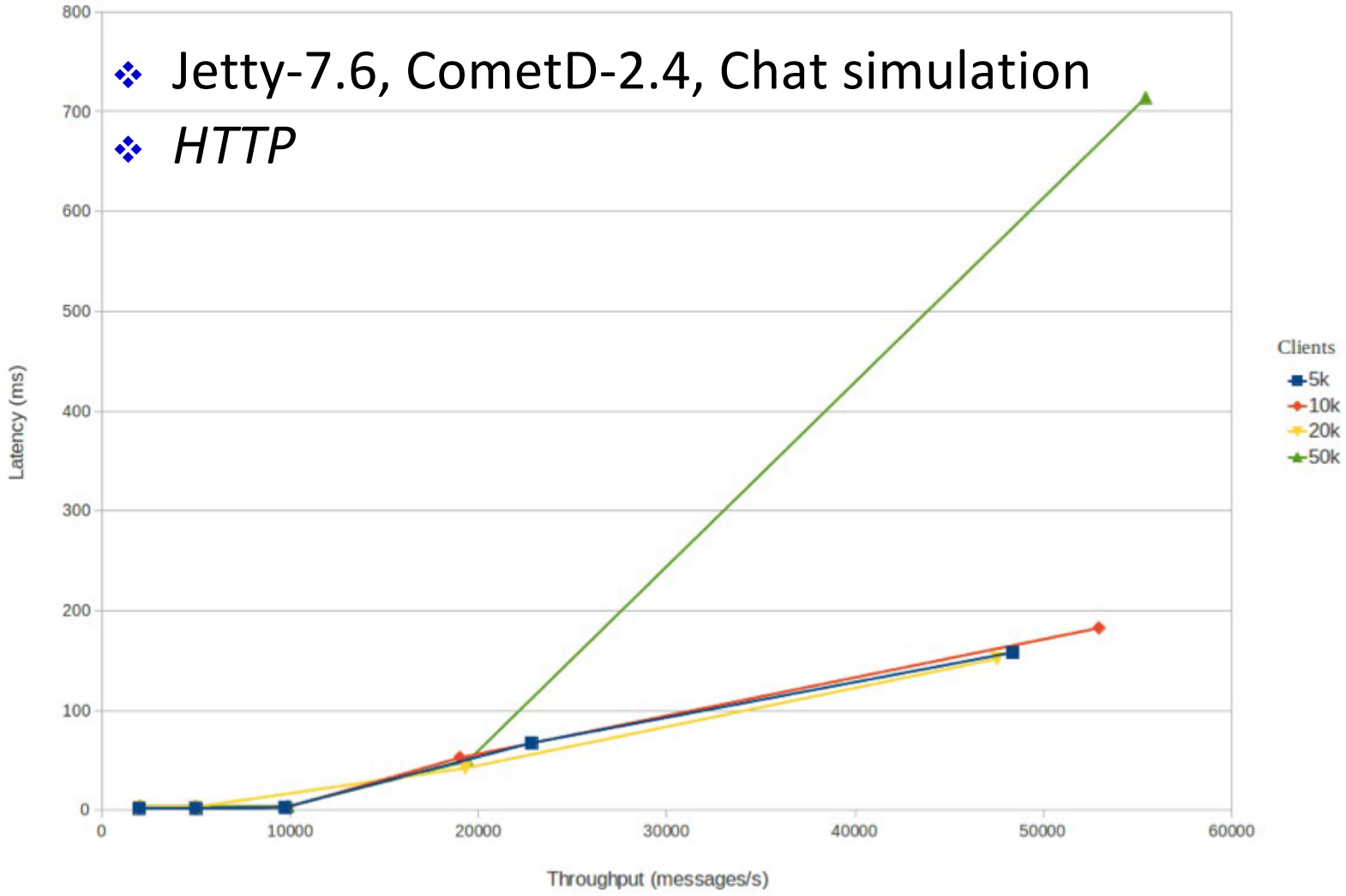
WebSocket Framing

- WebSocket frames of two types
  - Control frames (Close, Ping, Pong) & Data frames (UTF8 Text & Binary)
- Protocol is very compact
  - Smallest data frame has only 2 bytes overhead



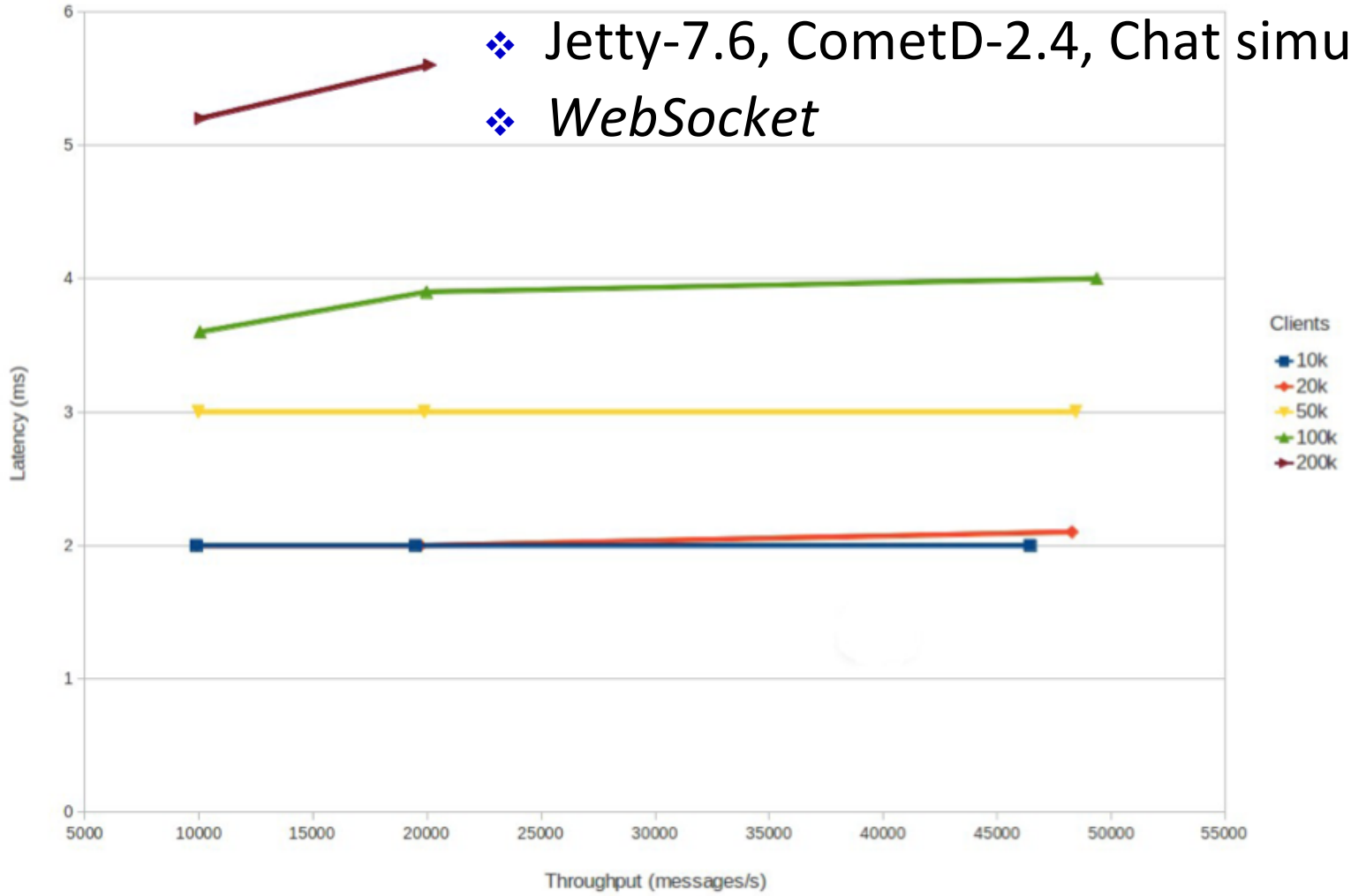
HTTP Performance

- Jetty-7.6, CometD-2.4, Chat simulation
- *HTTP*



HTTP Performance

- Jetty-7.6, CometD-2.4, Chat simulation
- *WebSocket*



WebSocket conclusions

- WebSocket is Here NOW!
  - In production
  - Big benefits
  - Use Jetty!
- WebSocket Limitations
  - Does not provide HTTP semantic
    - New applications need to be written
  - Intermediaries may need to be WebSocket aware
    - There are some problems
  - No connection limit
    - Multiplexing coming
  - Very Low level
    - Use CometD!

SPDY (Pronounced ”speedy”)

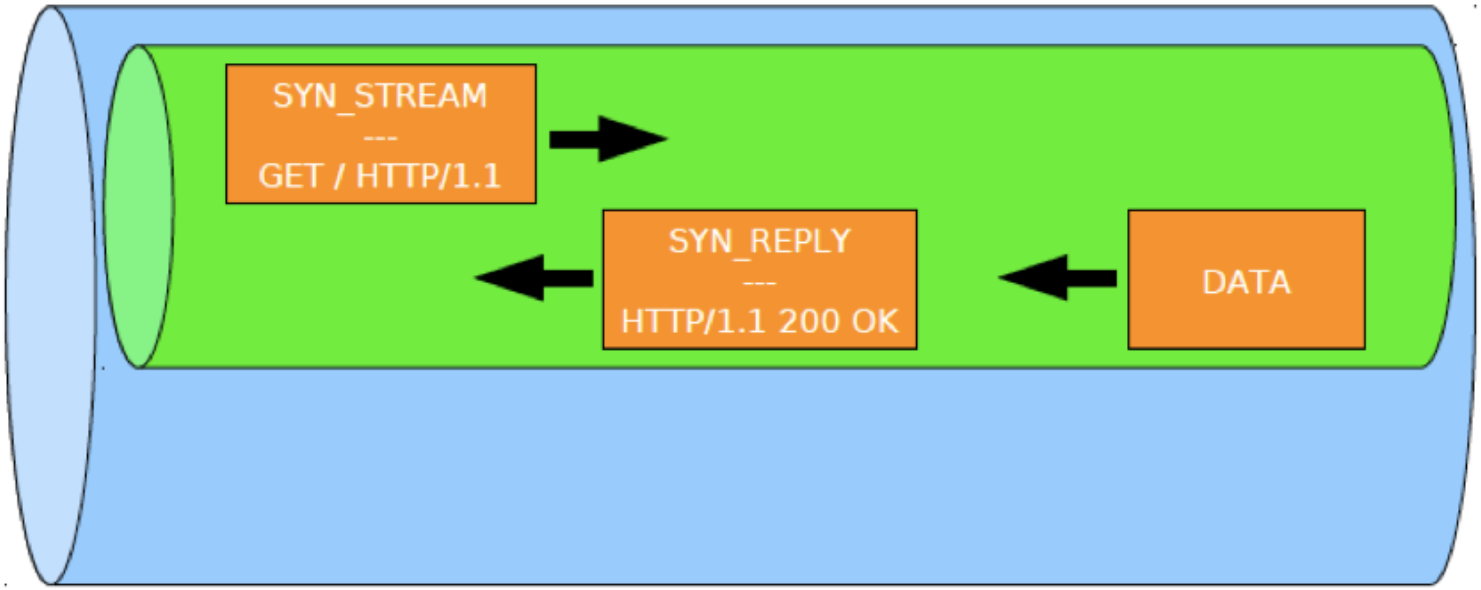
- SPDY is a live experiment to improve HTTP
  - Addresses HTTP 1.1 limits
  - Designed to be faster and better than HTTP
- Already widely deployed!
  - Google, Twitter, Webtide, etc.
  - Chrome & Firefox
- The SPDY protocol replaces HTTP on the wire
  - But it's transparent for applications

Protocol initiation in SPDY

- SPDY uses TLS (aka SSL) connection on port 443
  - TLS extended with Next Protocol Negotiation
- New framing protocol over TLS
  - Intermediaries don't know it is not HTTP wrapped in TLS
- Transports existing HTTP Semantics (GET, POST, HEAD etc)
  - Client and Server applications don't know it is not HTTP

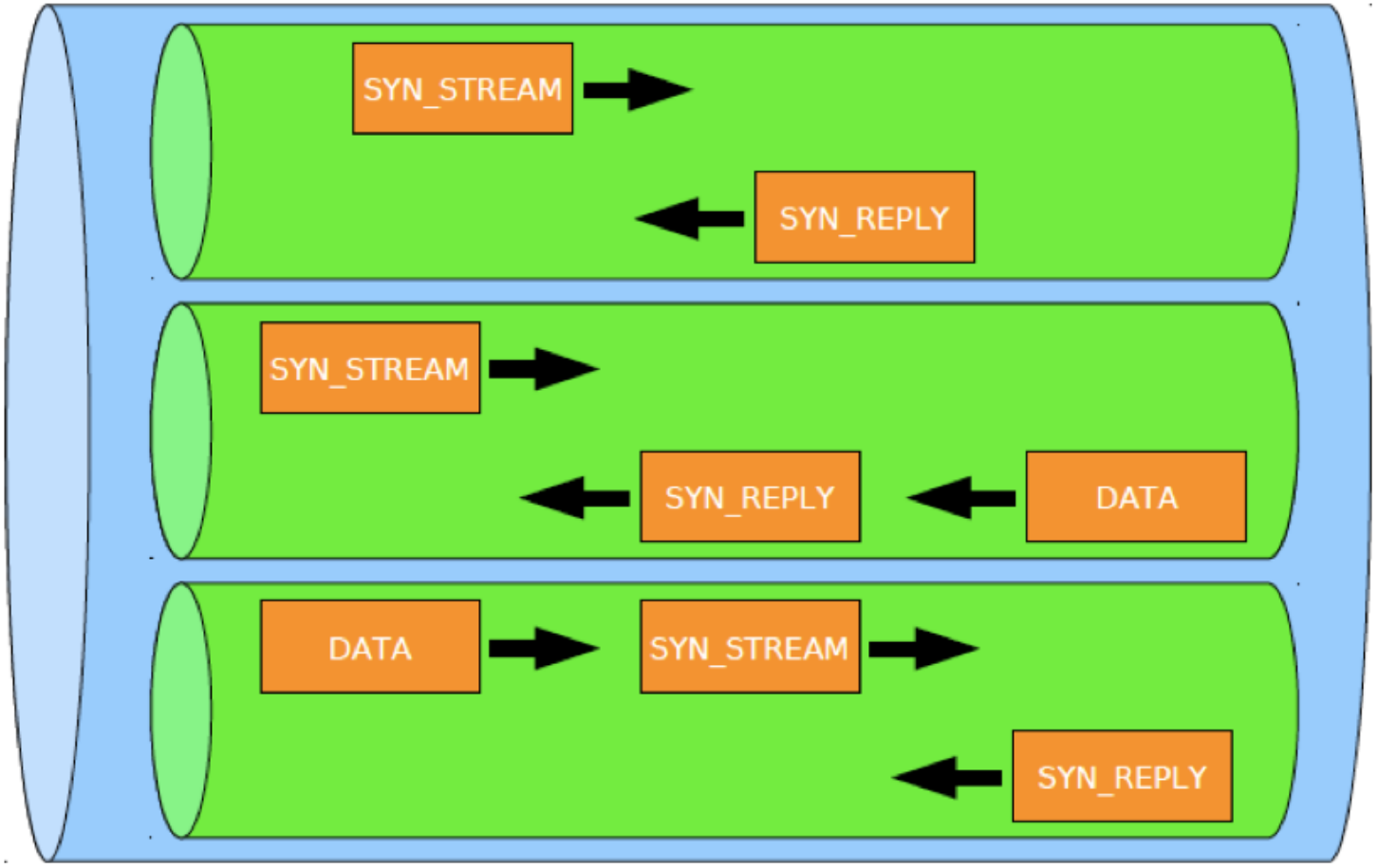
SPDY Framing

- SPDY defines a new framing layer
  - Binary, slightly more expensive than WebSocket's
  - Faster than HTTP to parse
  - Built to carry HTTP, but can carry other protocols



SPDY Multiplexing

- Multiplexing is built-in



SPDY Multiplexing

- Multiplexing
  - Allows to make better use of TCP connections
    - Reduces TCP slow start
  - Uses less resources on the server
  - Responses may be sent out of order
  - Without waiting for slow responses
- Key point: reducing round trip waits
  - Caused by limited connections and lack of multiplexing

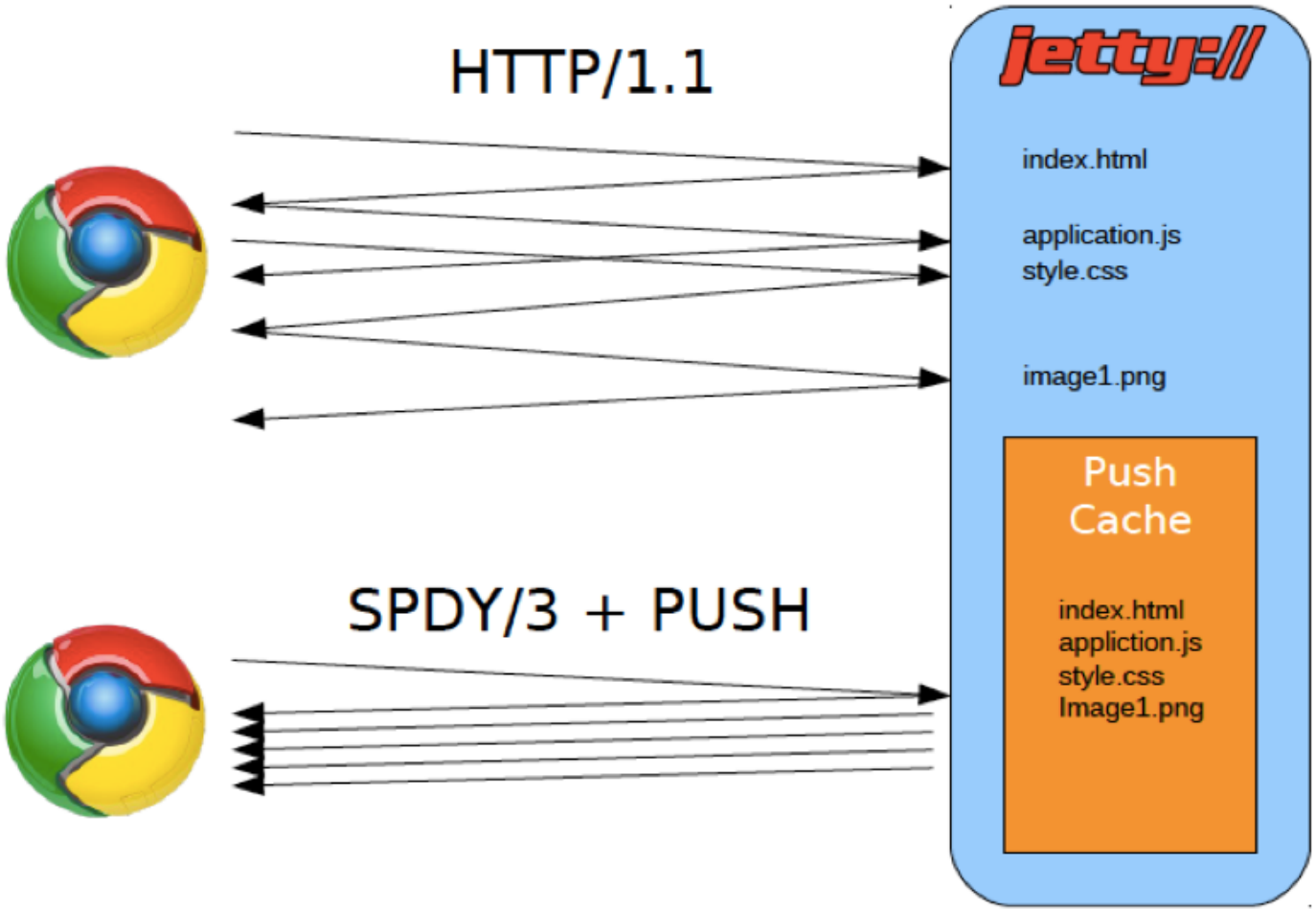
SPDY Semantics

- SPDY is built for HTTP
  - But allows WebSocket to be carried too
- HTTP header compression
  - Typical HTTP requests contains ~1 KiB of headers
    - Most of them are constant
  - Saves ~25% on first request, 80% and more on subsequent requests
- Headers for typical requests to webtide.com:
  - 568 bytes → 389 bytes → 26 bytes
  - 100% → 68% → 4.5%

SPDY Push

- SPDY Push
  - Server pushes secondary resources that are associated to a primary resource
  - Works in collaboration with browser's cache for a better user experience
- Jetty is one of the first SPDY servers that provides automated push functionalities
  - Totally transparent for applications
  - Based on the “Referrer” header
    - To associate primary and secondary resources
  - Using “If-Modified-Since” header to avoid unnecessary Pushes

SPDY Push



SPDY Speed

- How much faster than HTTP can SPDY be ?
  - Early to say, but initial figures up to 25%
  - SPDY optimizations limited by HTTP “hacks”
    - Domain sharding
- Can my JEE web application leverage SPDY ?
  - xYes ! And without changes !

SPDY & Jetty

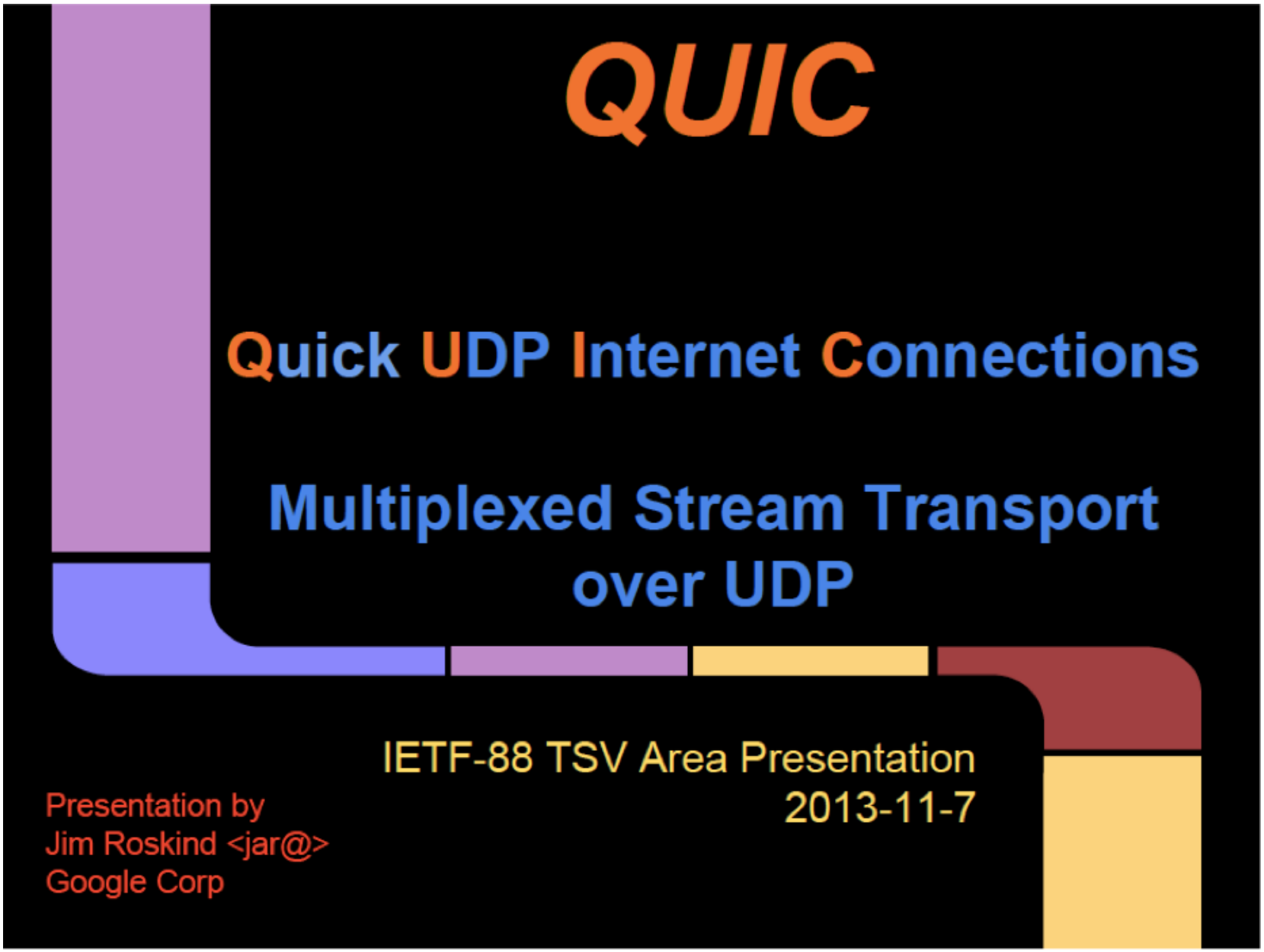
- Jetty 7 & Jetty 8
  - SPDY added by mocking HTTP wire protocol
  - Multiplexing requires mock protocol per channel
- Jetty 9
  - Re architected to separate wire protocol from semantics
  - HTTP semantics shared between HTTP, SPDY
  - Supports 1:n Protocol:Semantics
  - Same separation of Websocket protocol from semantics
  - HTTP, SPDY, WebSocket, TLS 1st class citizens

HTTP/2.0

- HTTP 2.0 work has started
  - IETF HTTPbis working group rechartered
- Several Proposals received:
  - Google - SPDY as the basis
  - Microsoft – websocket framing with HTTP semantics
- SPDY Adopted as the starting point
  - Unlikely to require TLS or NPN
  - Will define precise interactions with proxies
- Update: SPDY adopted as HTTP/2.0

The Big Picture

- The Web is evolving
  - Web Protocols must keep the pace new loads
  - Applications must keep pace with new semantics



What is QUIC?

Effectively replaces TLS and TCP out from under SPDY (predecessor of HTTP/2.0)

Provides multiplexed in-order reliable stream transport (especially HTTP) over UDP

Protocol is pushed into application space (unlike TCP which is handled in kernel)

Overview

- Why aren't current protocols enough?
  - e.g., SPDY multiplexes streams, doesn't it?
- What could make QUIC valuable?
  - What could make it better(?) than SPDY?
- Status of efforts?

Why is SPDY fast?

- It is all about *latency (time till response)*
- SPDY multiplexes requests over one TCP connection
- SPDY compresses headers
- What is the problem?

Why isn't SPDY Enough?

- SPDY runs over TCP
  - Lose one SPDY packet: all the streams wait
    - HOL blocking
  - Lose one SPDY packet, bandwidth shrinks
    - Sharded connections have an advantage!!!
- SPDY may be slow to connect
  - TCP connect may cost 1-Round-Trip-Time (RTT)
  - TLS connect costs at least another RTT
- TLS and TCP are slow to evolve
  - More importantly, they are very slow to deploy
    - (at both ends, and in middle boxes!)

QUIC Goals

1. Deploy in today's internet
2. **Low latency** (connect, and responses)
   a. It is *ALL* about the latency
3. Reliable-stream support (like SPDY)
   a. Reduce Head Of Line (HOL) blocking due to packet loss
4. Better congestion avoidance than TCP
   a. Iterate and experiment
5. Privacy and Security comparable to TLS
6. Mobile interface migration
7. Improve on quality of sliced bread

QUIC Success Criteria

The Internet is faster and more pleasant to use

Two paths:

a) QUIC makes headway reducing latency

b) TCP and TLS steam ahead, and perhaps use techniques advocated for QUIC

Either way: The users will win.

Field Data, Plus Application Needs Drive Development

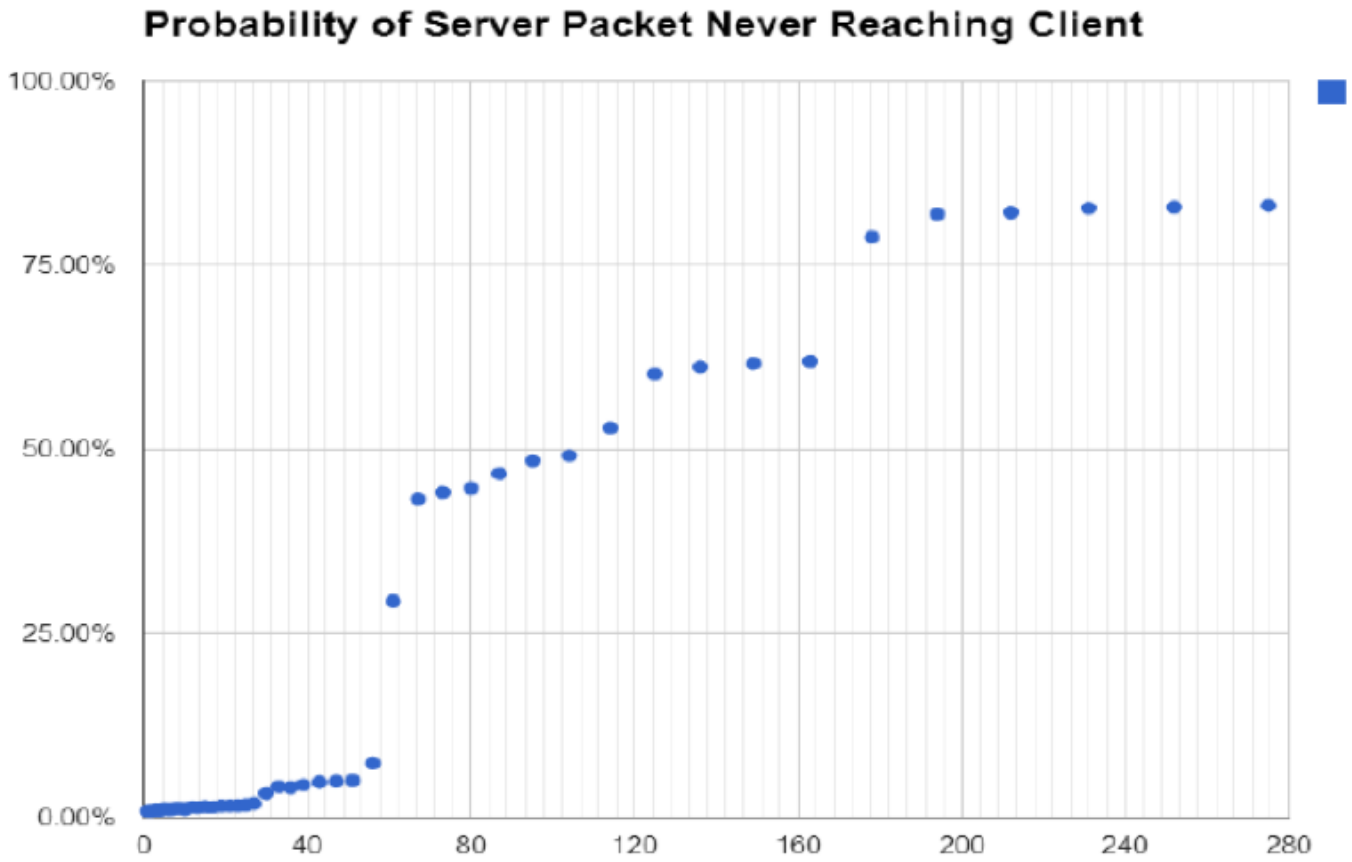
- Client side data from Chrome
  - Real users; Real user machines; Real cross traffic; Real ISPs
  - Aggregate data, and perform A/B experiments
- Server side instrumentation evaluates application impact
  - User happiness drives everything

Can we really Deploy a UDP Protocol in Today's Internet?

- UDP works for gamers and VOIP
  - They really care about latency
- 91-94% of users can make outbound UDP connections to Google
  - Tested for users that had TCP connectivity to Google
- UDP is plausible to build a transport in today's internet
  - See NAT Unbinding data in supporting slides

NAT Unbinding:

How much idle until unbinding?



How does QUIC achieve 0-RTT Connection Cost?

- Speculate that the server's public key is unchanged since last contact
  - Propose session encryption key in first packet
- Include GET request(s) immediately after
  - Upgrade to Perfect Forward Secrecy ASAP
- Similar speculative techniques tried/developed in TLS and TCP
  - See crypto doc for fancy details
  - See supporting slides for some highlights

Congestion Avoidance via Packet Pacing

- QUIC has pluggable congestion avoidance
  - TCP Cubic is baseline
  - Working on Pacing *plus* TCP Cubic
  - Working on bandwidth estimation to drive pacing
- QUIC monitors inter-packet spacing
  - Monitors one-way packet transit times
  - Spacing can be used to estimate bandwidth
  - Pacing reduces packet loss

Does Packet Pacing really reduce Packet Loss?

- Yes!!! Pacing seems to help a lot
- Experiments show notable loss when rapidly sending (unpaced) packets
- Example: Look at 21st rapidly sent packet
  - 8-13% lost when unpaced
  - 1% lost with pacing
- See supporting slides on “Relative Packet ACK probabilities” for some details

How might QUIC connection survive a Mobile Network Change?

- TCP relies on src/dest IP/port pairs
  - Mobile client (changing network/IP) means broken TCP connection :-(
  - Broken TCP connection means big reconnect latency
- QUIC relies on a 64 bit GUID in all packets
  - Client source IP is used only to respond to the mobile client
- ...and of course with QUIC, if we lose....
  - ...then fast 0-RTT reconnect is a fallback

How can a Forward Error Correction (FEC) Packet help?

- Trade increased bandwidth for decreased latency
- QUIC has packet level Error Correction
  - Keep a running-XOR of (some) packets
    - Send XOR as an Error Correction packet
- ...but is packet loss bursty?
  - XOR Error Correction won't work if we have several consecutive losses!

Will Error Correction Coding really help?

- Packet loss is not that bursty :-)

Example:

- 20 packets with about 1200 Bytes each
  - Retransmit needed 18% of the time
- 20 packets plus FEC Packet
  - Retransmit needed 10% of the time

- 5% extra bandwidth ==> -8% retransmits

See support slide on Retransmit Probabilities for 1200B payloads for experimental data

Status of Efforts (11/2013)

- Currently landing, limping, and evolving
  - In Chrome and in some Google servers
  - Trying to work as well as TCP Cubic
  - FEC built in… but not turned on
  - 0-RTT works when same server is hit
- Try prototype Chrome canary
  - about:settings Enable QUIC :-) (must restart)
  - about:net-internals to look at activity
- Try test-server in chromium codebase
- Longer road to Crypto PCI compliance for handling credit cards :-(

Backup Slide areas: Other Nice Stuff in QUIC Design

1. Defend against "Optimistic ACK Attack"
   a. Defend against Amplification Attacks
2. Handle header compression (like SPDY)
   a. ...despite out-of-order arrival of packet (context)!
3. Support TCP Congestion Avoidance
   a. Baseline: prevents Internet Congestion Collapse
4. 0-RTT Server-side redirects
   a. Hand off service to other server without an RTT, while getting good crypto in that new server's stream

More Nice Crypto Things In QUIC

1. Encryption used for both UDP port 80 (HTTP) and for port 443 (HTTPS)
   a. … but port 80 does not authenticate server
2. Connections upgrade to Perfect Forward Secrecy asap
   a. After about 1 RTT
3. Packet padding to make traffic analysis a tad harder
4. FIN (like) and ACK packets authenticated
   a. No 3rd party teardown

How can I contribute?

News group: proto-quic@chromium.org

https://groups.google.com/a/chromium.org/d/forum/proto-quic

Contribute to Chromium source tree!

Evolving wire spec tries to record state-of-the-Chromium-tree for landed code

...but debugging often drives changes

Design document has motivations and justifications

FAQ For Geeks addresses some questions

How is HOL Blocking Reduced?

- UDP is not in-order, like TCP
  - QUIC adds packet sequence numbers
- SSL crypto block depend(ed) on the previous block's decryption
  - QUIC uses packet sequence numbers as crypto-block Initialization Vector (IV) source
  - QUIC collapses and reuses protocol layers!
- SSL encrypted blocks don't match IP packet boundaries :-(
  - QUIC aligns encryption blocks with IP packets
  - One lost QUIC packet won't stop the next packet from being decrypted :-)

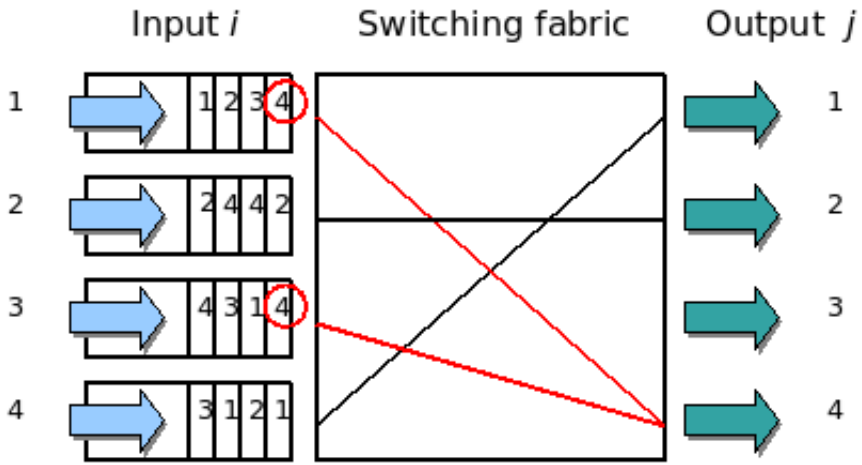
What is HOL Blocking?

- **Head-of-line blocking** (**HOL blocking**) in computer networking is a performance-limiting phenomenon that occurs when a line of packets is held-up by the first packet, for example in input buffered network switches, out-of-order delivery, and multiple requests in HTTP pipelining.

https://en.wikipedia.org/wiki/Head-of-line_blocking

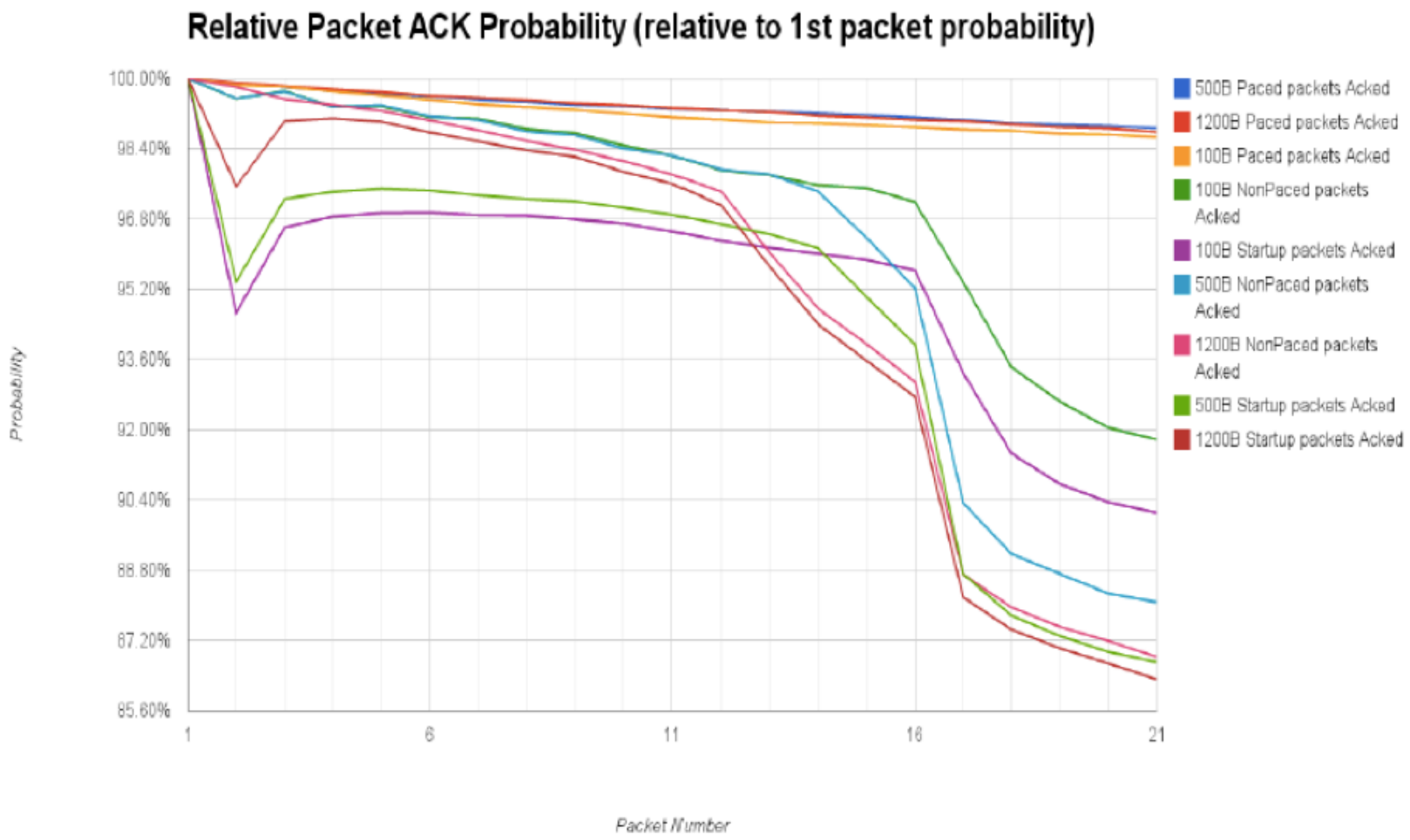
HOL Blocking Example

- Inputs 1+3 want Output 4
- Fabric chooses Input 3
- 1st input flow cannot be processed in the same clock cycle
- Further,1st input flow is blocking a packet for output 3, which is idle

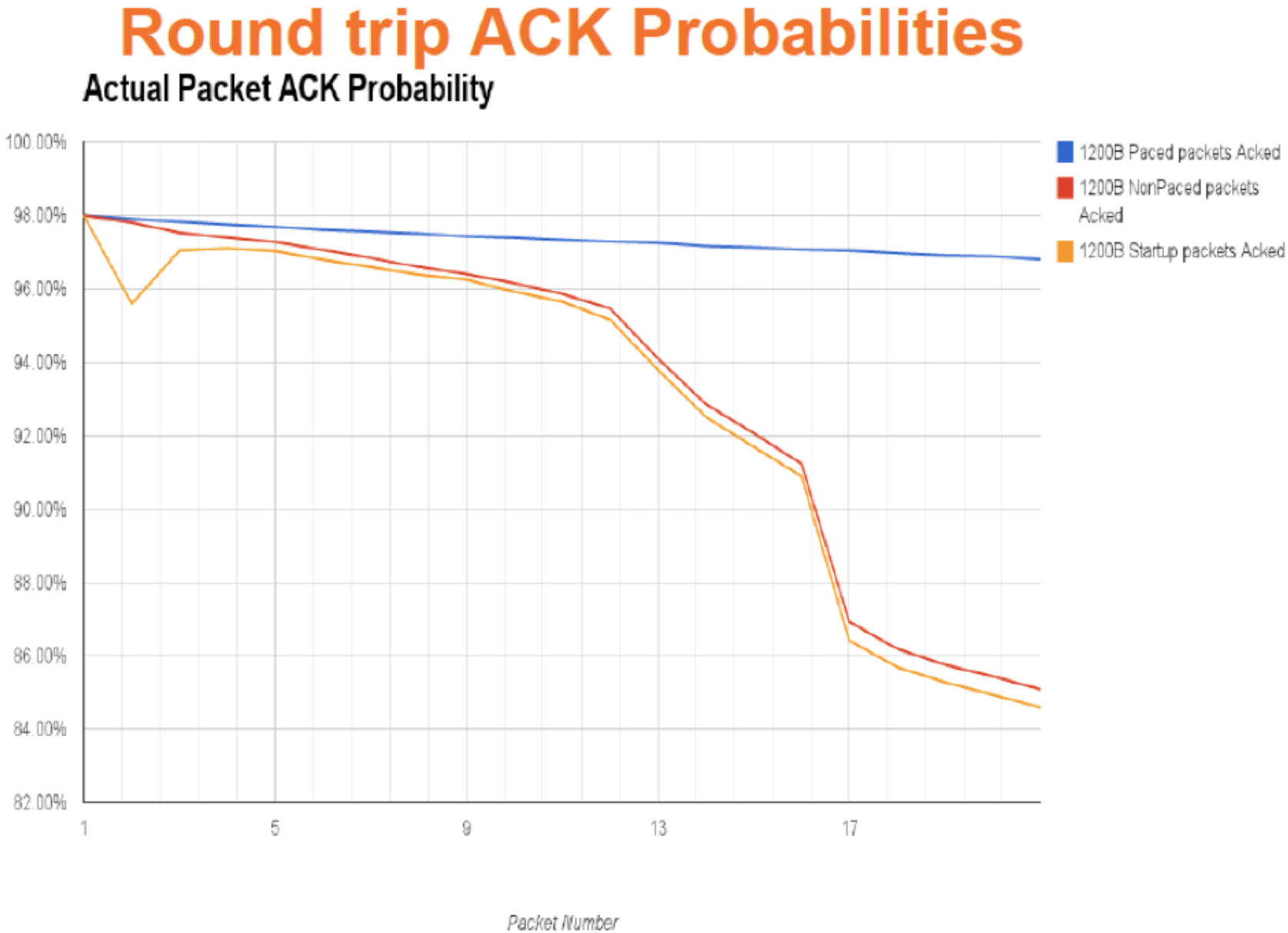


Client->Server->Client round trip:

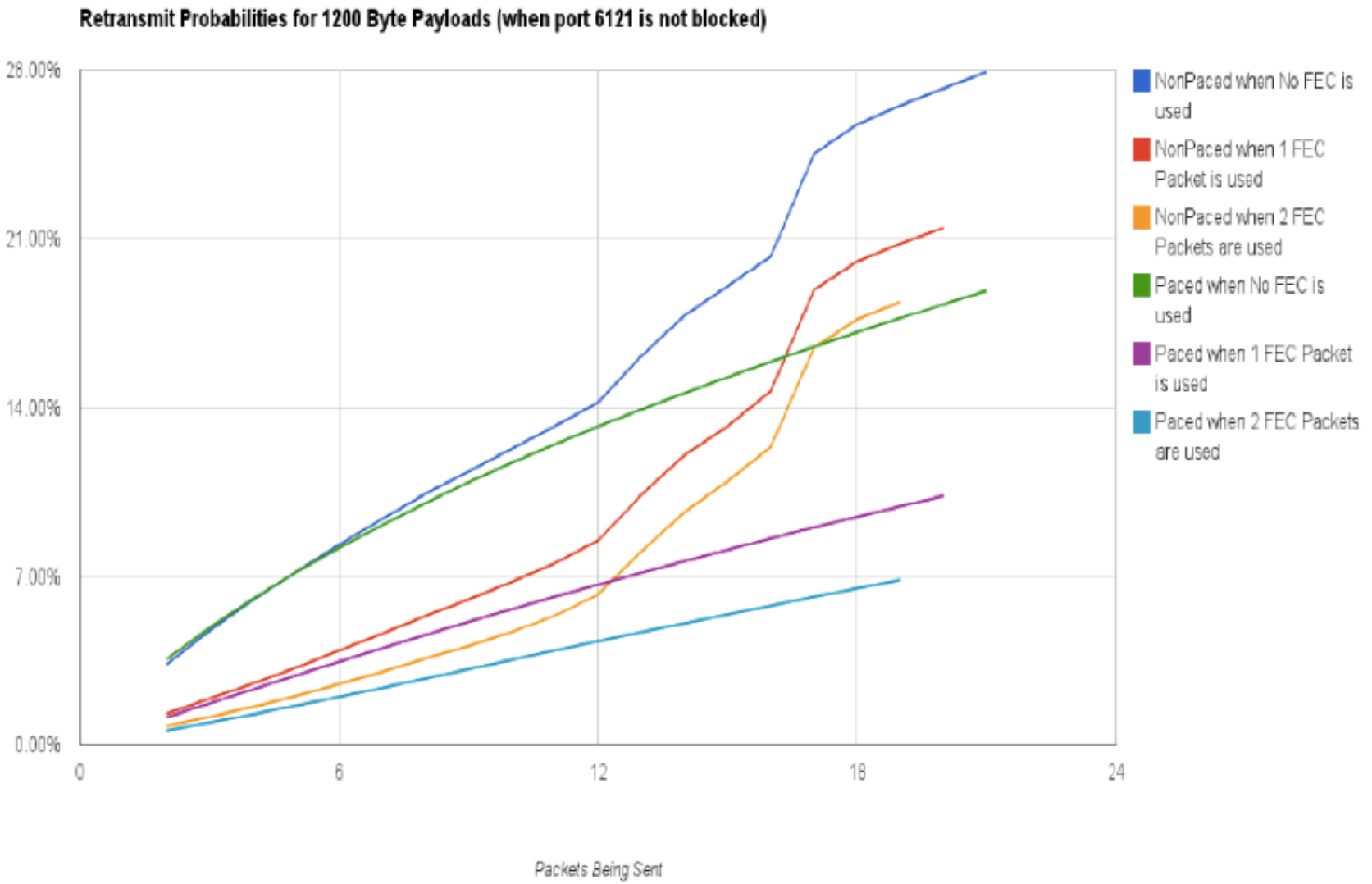
21 packets: 1200B vs 500B vs 200B



Round trip ACK Probabilities



Retransmit-needed Probabilities



NAT Unbinding Results Caveats

1. Did not correct for 1% ambient packet loss
   a. Could have sent N packets after pause
2. Did not validate internet connectivity
   a. Users may have disconnected… so there is some disconnection conflation
3. Did not test to see if NAT was being used
   a. IPv6 *often* avoids NAT
   b. This could explain the 20% tail that “never”(?) unbinds

NAT Unbinding

- From RFC 2663, 3.2.3:

NAT will perform address unbinding when it believes that the last session using an address binding has terminated

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 electronic mail

- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks

2.7 socket programming with UDP and TCP

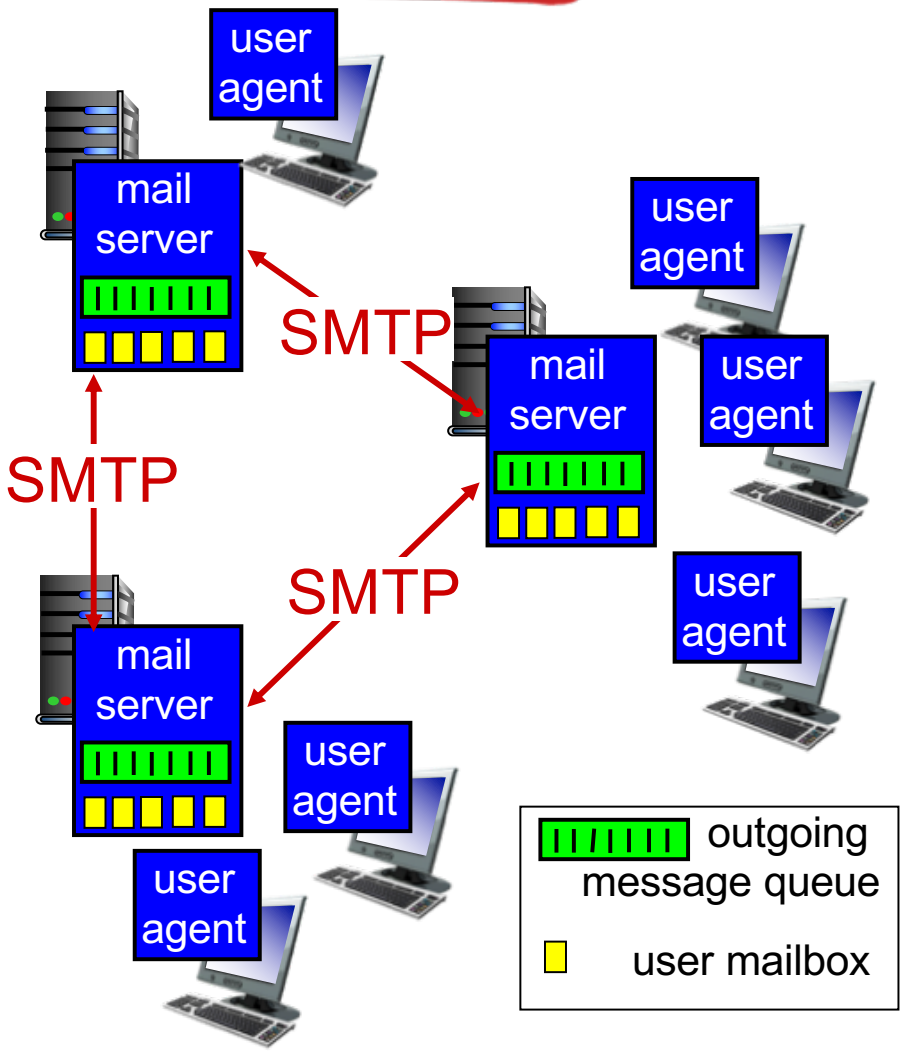
Electronic mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

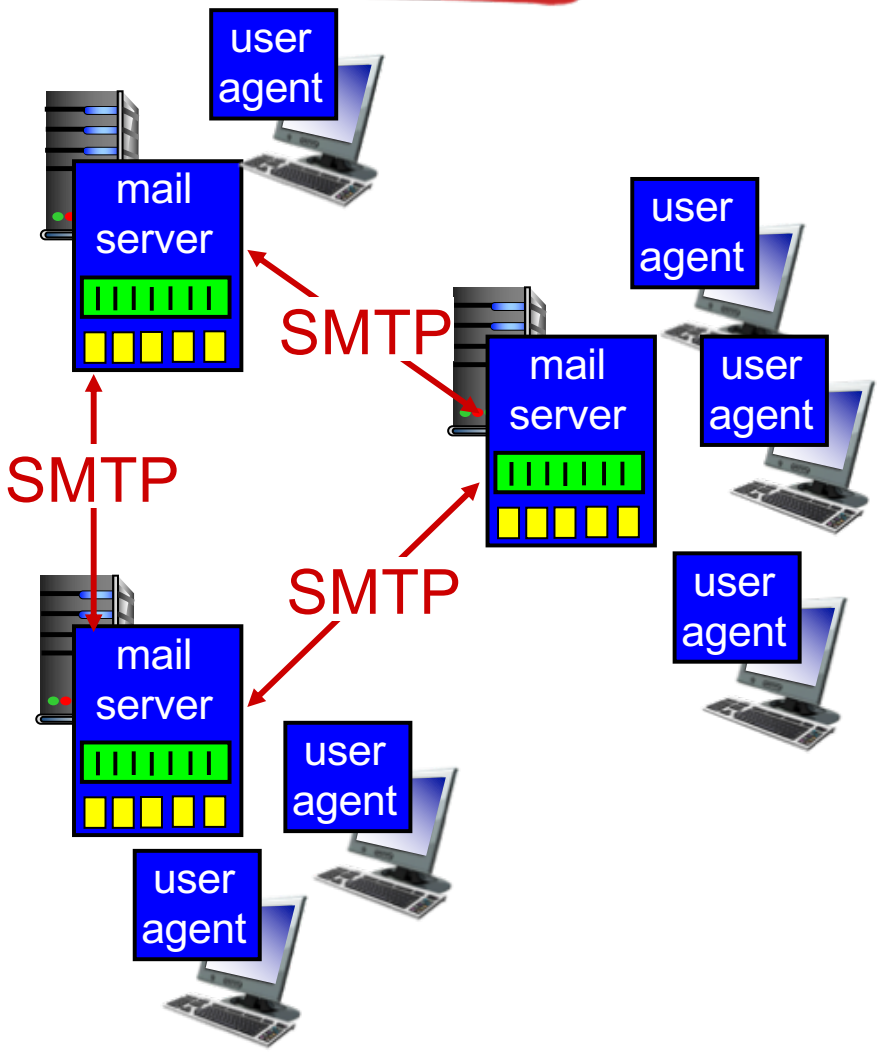
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



Electronic mail: mail servers

mail servers:

- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server

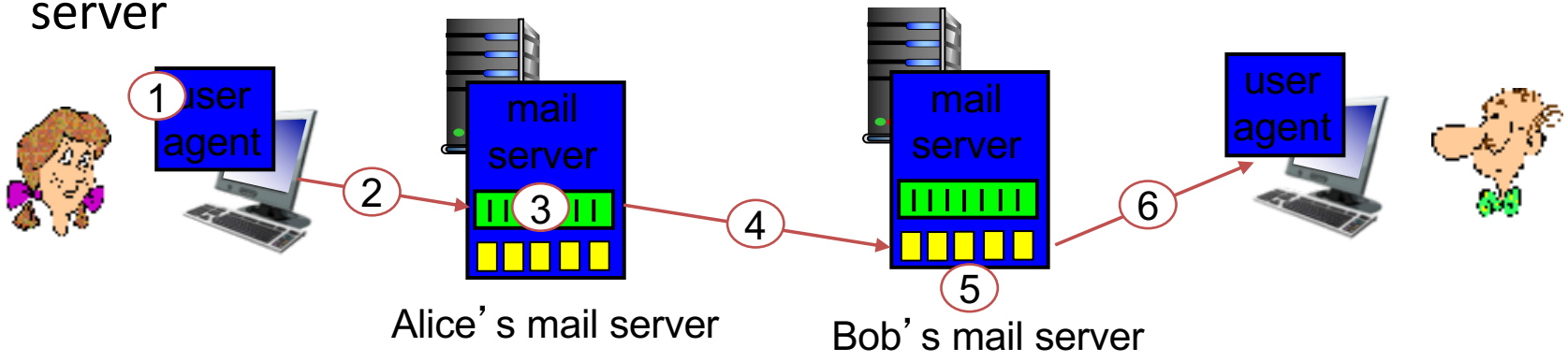


Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction (like HTTP)
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCI

Scenario: Alice sends message to Bob

- 1) Alice uses UA to compose message "to" bob@someschool.edu
- 2) Alice's UA sends message to her mail server; message placed in message queue
- 3) client side of SMTP opens TCP connection with Bob's mail server
- 4) SMTP client sends Alice's message over the TCP connection
- 5) Bob's mail server places the message in Bob's mailbox
- 6) Bob invokes his user agent to read message



Sample SMTP interaction

```
S: 220 hamburger.edu
    C: HELO crepes.fr
    S: 250  Hello crepes.fr, pleased to meet you
    C: MAIL FROM: <alice@crepes.fr>
    S: 250 alice@crepes.fr... Sender ok
    C: RCPT TO: <bob@hamburger.edu>
    S: 250 bob@hamburger.edu ... Recipient ok
    C: DATA
    S: 354 Enter mail, end with "." on a line by
 itself
    C: Do you like ketchup?
    C: How about pickles?
    C: .
    S: 250 Message accepted for delivery
    C: QUIT
    S: 221 hamburger.edu closing connection
```

Try SMTP interaction for yourself:

- **`telnet dijkstra.cs.uh.edu 25`**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

DO NOT use this server to send email to anyone but each other. The systems administrator is being very kind to me letting us use this.

There's also a chance we'll get stuck at the border.

The above lets you send email without using email client (reader)

SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses `CRLF.CRLF` to determine end of message

comparison with HTTP:

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response message
- SMTP: multiple objects sent in multipart message

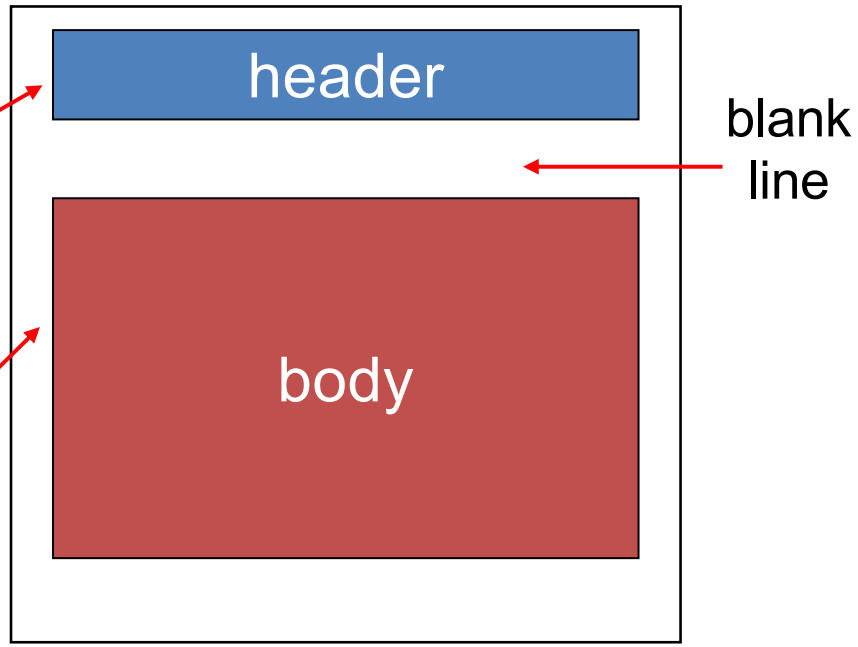
Mail message format

SMTP: protocol for exchanging email messages

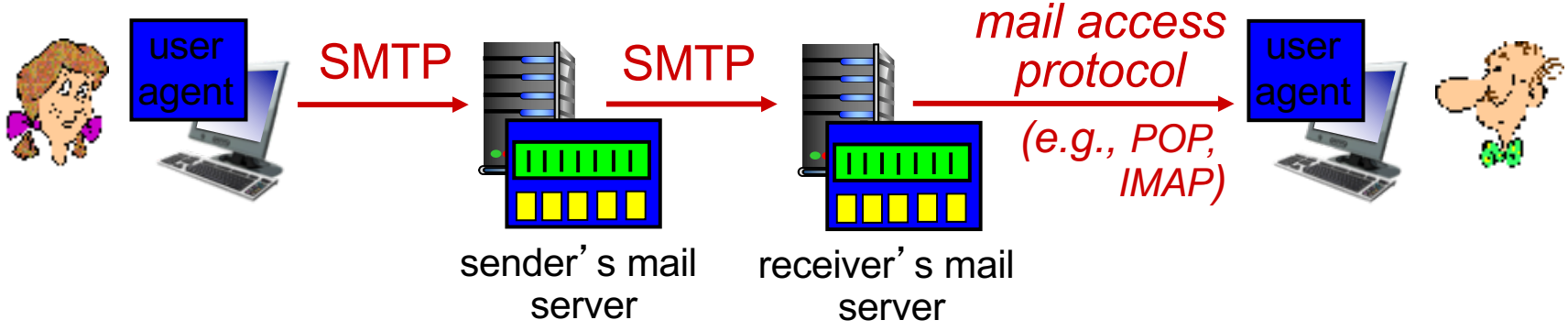
RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP MAIL FROM, RCPT TO: commands!
- Body: the "message"
  - ASCII characters only



Mail access protocols



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]: authorization, download
  - IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored messages on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

POP3 protocol

authorization phase

- client commands:
  - **`user`**: declare username
  - **`pass`**: password
- server responses
  - **`+OK`**
  - **`-ERR`**

transaction phase, client:

- **`list`**: list message numbers
- **`retr`**: retrieve message by number
- **`dele`**: delete
- **`quit`**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 (more) and IMAP

more about POP3

- previous example uses POP3 "download and delete" mode
  - Bob cannot re-read e-mail if he changes client
- POP3 "download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

IMAP

- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name