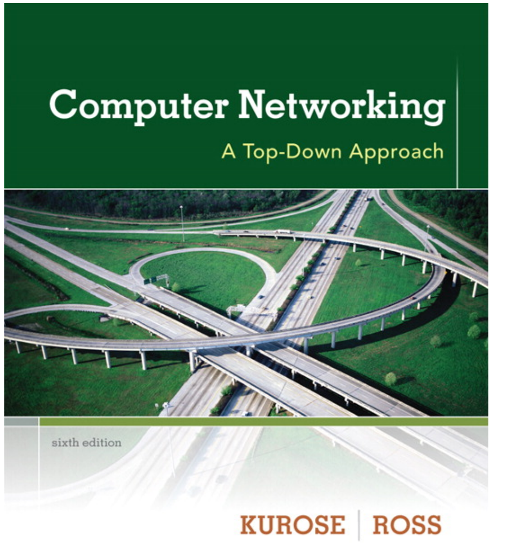
SCOM/ SRSI
Application Layer, Web, HTTP

Ana Aguiar
DEEC, FEUP

2020-21

Chapter 2
Application Layer



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

Chapter 2: outline

2.1 principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P applications

2.7 socket programming with UDP and TCP

Chapter 2: application layer

our goals:

- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- creating network applications
  - socket API

Some network apps

- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- …
- …

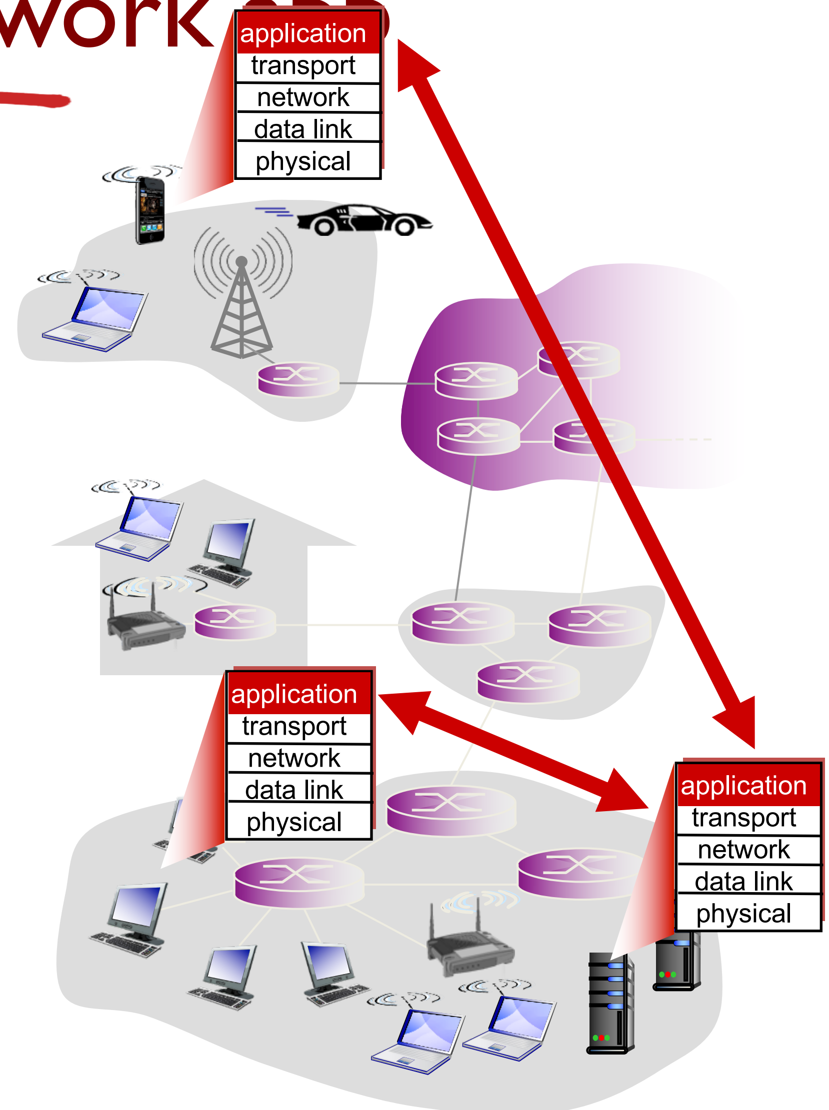
Creating a network app

write programs that:

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation

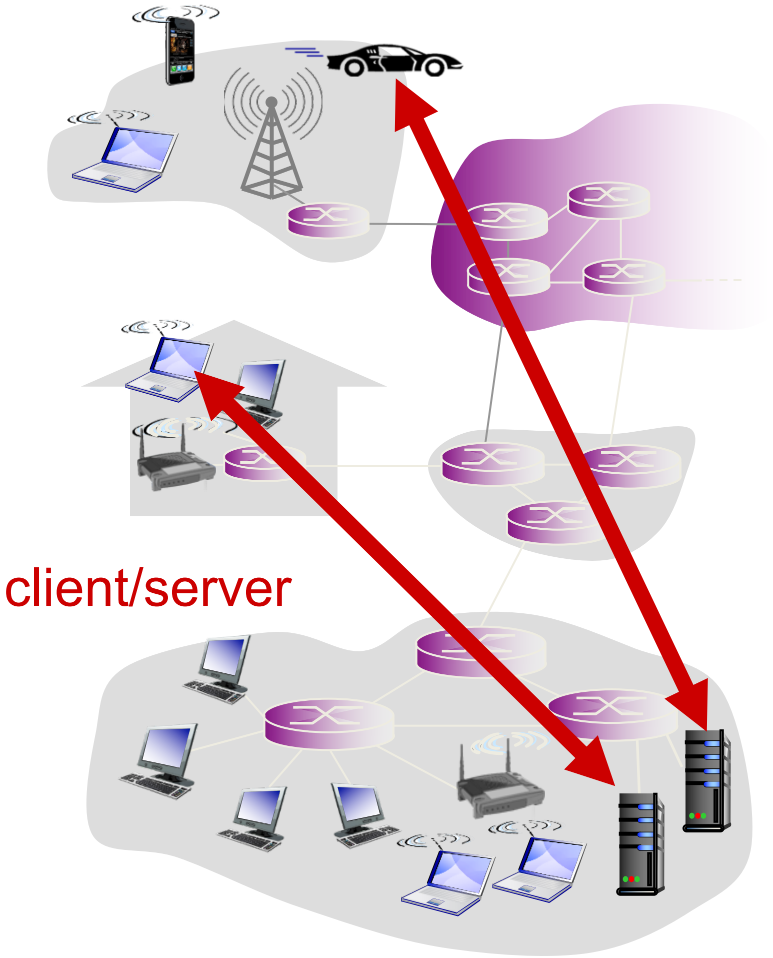


Application architectures

possible structure of applications:

- client-server
- peer-to-peer (P2P)

Client-server architecture



server:

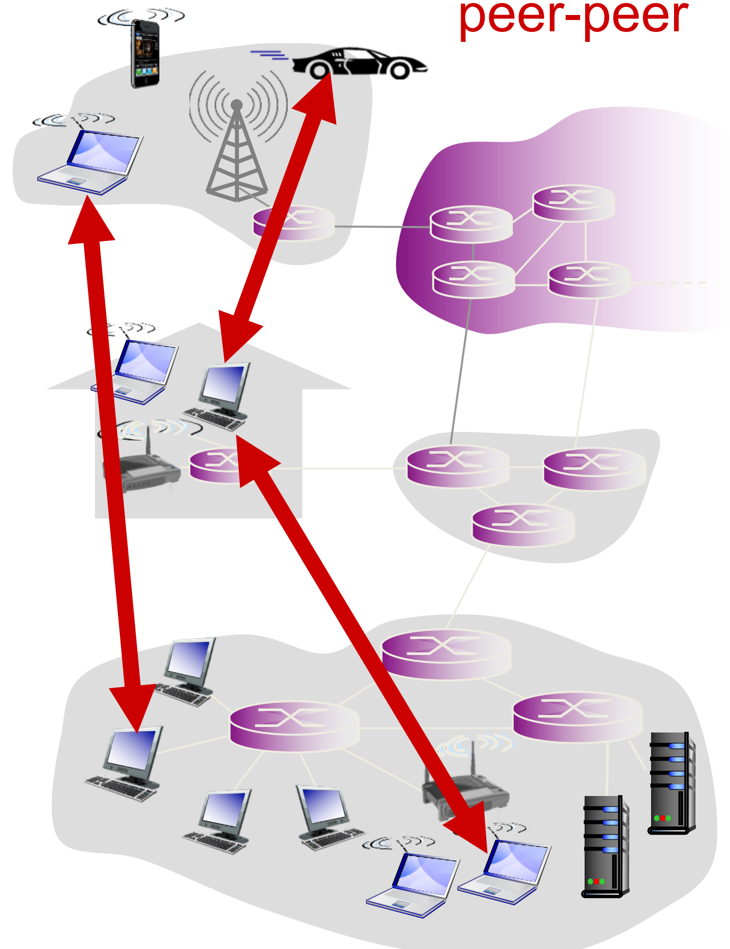
- always-on host
- permanent IP address
- data centers for scaling

clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



Processes communicating

process: program running within a host

- within same host, two processes communicate using inter-process communication (defined by OS)
- processes in different hosts communicate by exchanging messages

clients, servers

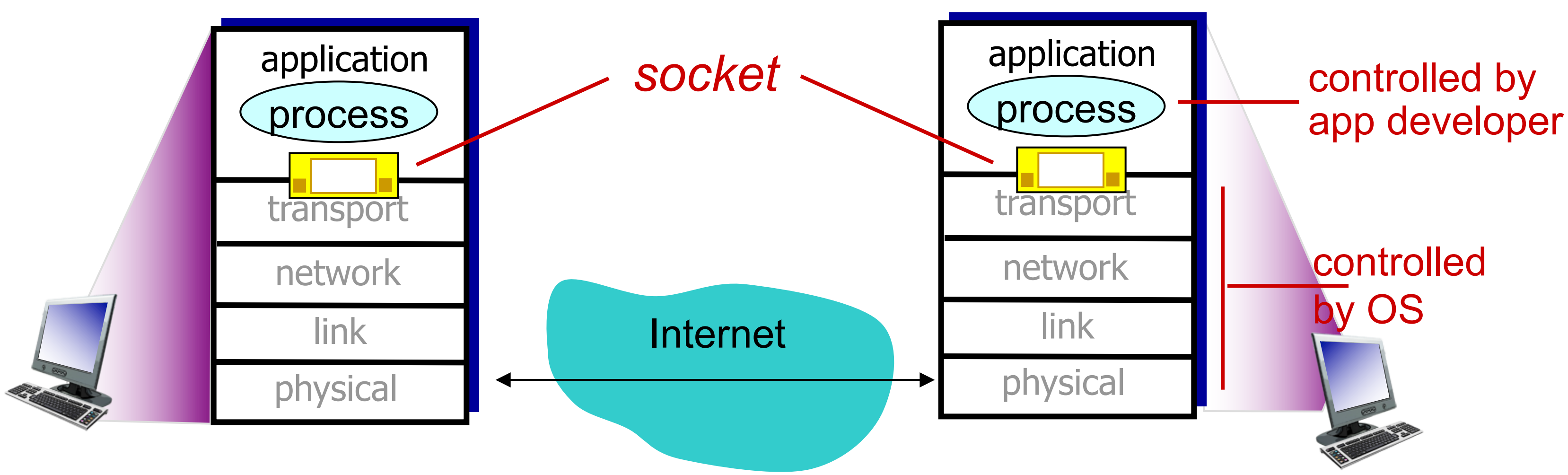
client process: process that initiates communication

server process: process that waits to be contacted

- aside: applications with P2P architectures have client processes & server processes

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* no, *many* processes can be running on same host

- *identifier* includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80
- more shortly…

App-layer protocol defines

- types of messages exchanged,
  - e.g., request, response
- message syntax:
  - what fields in messages & how fields are delineated
- message semantics
  - meaning of information in fields
- rules for when and how processes send & respond to messages

open protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype

What transport service does an app need?

data integrity

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get

security

- encryption, data integrity, …

Internet transport protocols services

TCP service:

- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security
- *connection-oriented:* setup required between client and server processes

UDP service:

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, orconnection setup,

Q: why bother? Why is there a UDP?

Internet apps: application, transport protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

WEB

Is the web the same as the Internet?

What makes a machine part of the web?

Is the web the same as the Internet?

What makes a machine part of the web?

Further reading

- HTTP is the New Narrow Waist
- HTTP as the narrow waist of the future internet

World Wide Web

- How does web browsing work?
- Which is the communication model?
- How do you find contents?
- Which protocols are involved?
- How could you improve the quality of experience?

Exercise: HTTP Protocol

- Start a wireshark capture
- Open the browser
- Access www.cloud.futurecities.up.pt, www.w3c.org, …
- Stop the wireshark capture
- Analyse the log
  - Find the DNS query packets
  - How much time did it take to resolve the name?
  - How many TCP connections were opened?
  - Which information is inside the query and response headers?
  - What is the meaning of that information?

Resource, Locator, Address

- Information or function: what the user wants

Searches on a search engine

- Resource (document or service)

Fixed binding in current web, and URL also returned by search engine

- Locator (URL) (name of where it can be found)

Resolve name

- Address: IP address of the server from where to fetch the resource

Web

Typical Web Page

- Base HTML page + linked objects
  - Scripts
    - Local
    - External

  <script src="//s3-us-west-2.amazonaws.com/ieeeshutpages/gdpr/settings.js"></script>
  - Images

  <img src="img_girl.jpg" alt="Girl in a jacket" width="500" height="600">
  - Videos

  <video width="320" height="240" controls>
   <source src="movie.mp4" type="video/mp4">
   <source src="movie.ogg" type="video/ogg">
   Your browser does not support the video tag.
  </video>
  - Audio
  - …

HTTP

WWW Architectural Components

- Client, e.g. browser
- Server
- HTTP protocol
- HTML
- URL
- Proxies

HyperText Transfer Protocol

- HyperText Transfer Protocol (HTTP): RFC 2616
  - Client-server protocol
  - Request-response
  - Evoked by the browser on the client machine and by the server process on the server machine
  - Uses TCP
  - Stateless
    - Server and client do not keep state (information) about connection

HTTP Message

```
START LINE <CRLF>
MESSAGE_HEADER <CRLF>
<CRLF>
MESSAGE BODY <CRLF>
```

- Start Line says whether it is a request or response
- Message Header specifies options and parameters
- `<CRLF>` acts as delimiter
- Message Body contains the data
  - Request or response contents
- More information: https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

HTTP Requests

- GET: retrieve resource contents
- POST: write to resource
- PUT: update resource with contents
- DELETE: remove resource contents
- OPTIONS: retrieve information about communication capabilities and/ or content options
- HEAD: retrieve header only
- TRACE: application-level loopback (for debug)
- CONNECT: for use with proxy

HTTP Responses

- Start line contains
  - HTTP version
  - Response code
    - 1xx Informational
    - 2xx Success
    - 3xx Redirection
    - 4xx Client Error
    - 5xx Server Error
  - String explaining response (optional)
- How are non-text contents exchanged?
  - MIME

Cookies

- Why are cookies needed?
  - Recall that HTTP is a stateless protocol
  - How to keep session information, e.g. shopping cart?
- What are cookies?
  - Pieces of data sent as an HTTP response
  - Stored by client (browser) and sent in subsequent interactions
  - Used by the server to keep session state
- IETF Standard RFC 6265
  - http://tools.ietf.org/html/rfc6265
- Are cookies dangerous?
  - https://www.addedbytes.com/blog/are-cookies-dangerous/

User-server state: cookies

many Web sites use cookies

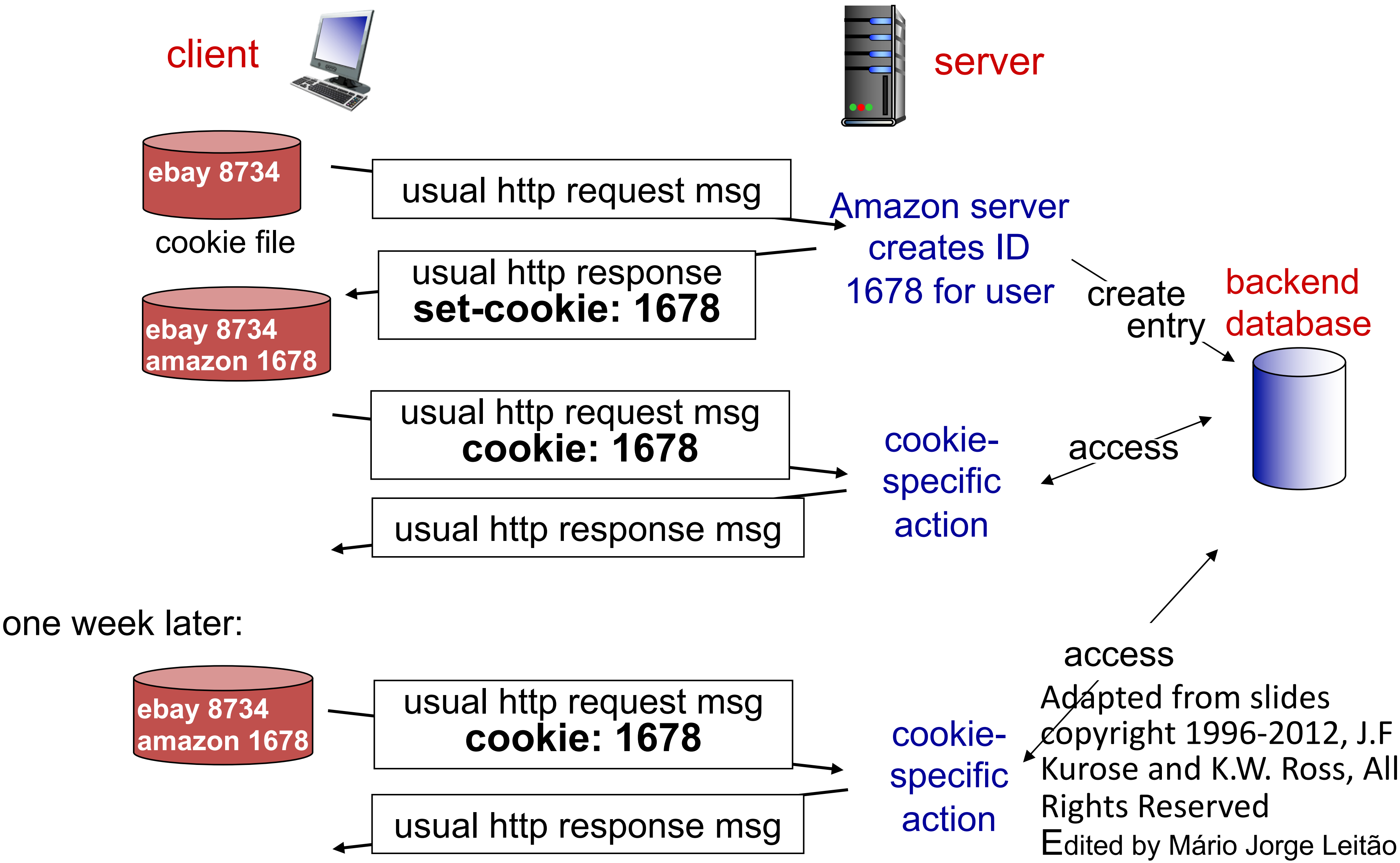
four components:

1) cookie header line of HTTP *response* message
2) cookie header line in next HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

example:

- Susan always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping “state” (cont.)



Cookies (continued)

what cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

aside

cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

how to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

Cookies

- How can you be tracked using cookies?

HTTP Performance

- Which metric would you use to evaluate HTTP?
- Go to the wireshark log you captured earlier today:
  - What was the page loading delay, i.e. the delay perceived by the user?
    - Assume that the browser did not add any delay
  - How many objects did the webpage have?

Web Performance

- What does it depend on?
  - Network performance: bandwidth, latency
  - Server performance
  - Contents
    - Amount and size of embedded objects (images, video, etc), external fonts, scripts, etc
    - Further reading on website performance: Patrick Meenan. How fast is your website?. Communications of the ACM 56, 4 (April 2013), 49-55. DOI=10.1145/2436256.2436270
  - For the user: also on processing and rendering time

Typical Web Page

- Base HTML page + linked objects
  - Scripts, fonts, style sheets
    - Local
    - External

<script src="//s3-us-west-2.amazonaws.com/ieeeshutpages/gdpr/settings.js"></script>
<link rel="stylesheet" type="text/css" href=”https://www.freefont.org/browserref.css">

  - Images

<img src="img_girl.jpg" alt="Girl in a jacket" width="500" height="600">

  - Videos

```
<video width="320" height="240" controls>
 <source src="movie.mp4" type="video/mp4">
 <source src="movie.ogg" type="video/ogg">
 Your browser does not support the video tag.
</video>
```

  - Audio
  - …

Accessing an HTML Document

- Access base HTML document
- Parse document
- Request all objects in the document
  - Requires resolving names and accessing other servers, if objects not all in the same server

HTTP connections

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

Non-persistent HTTP

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Non-persistent HTTP (cont.)

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

time

6. Steps 1-5 repeated for each of 10 jpeg objects

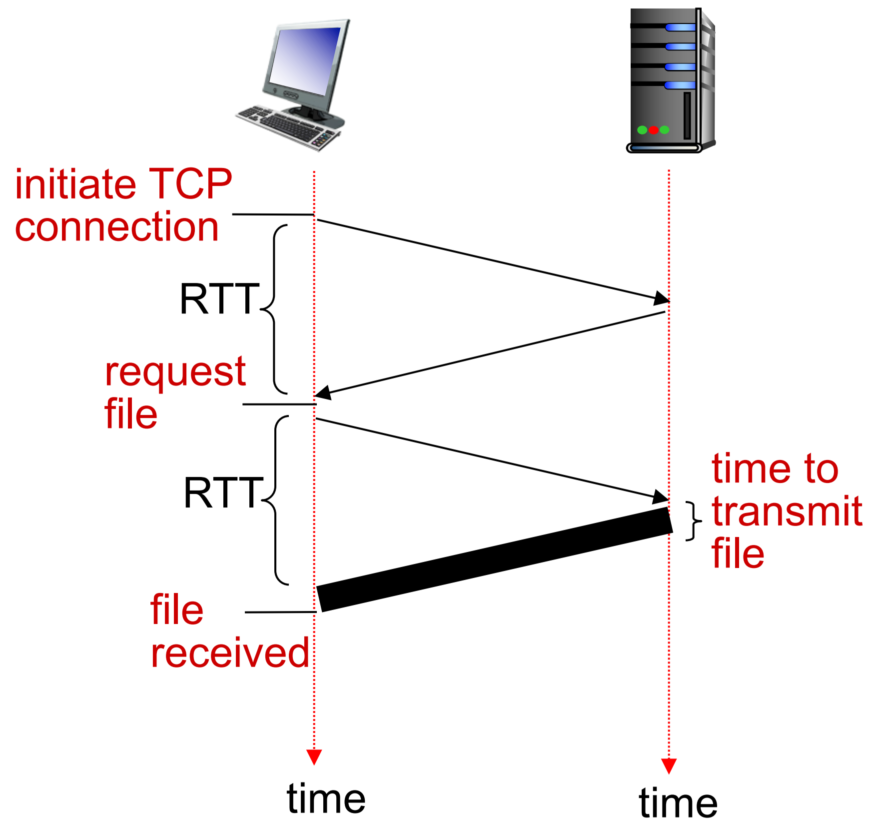
Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time
- non-persistent HTTP response time =

  2RTT+ file transmission time



Persistent HTTP

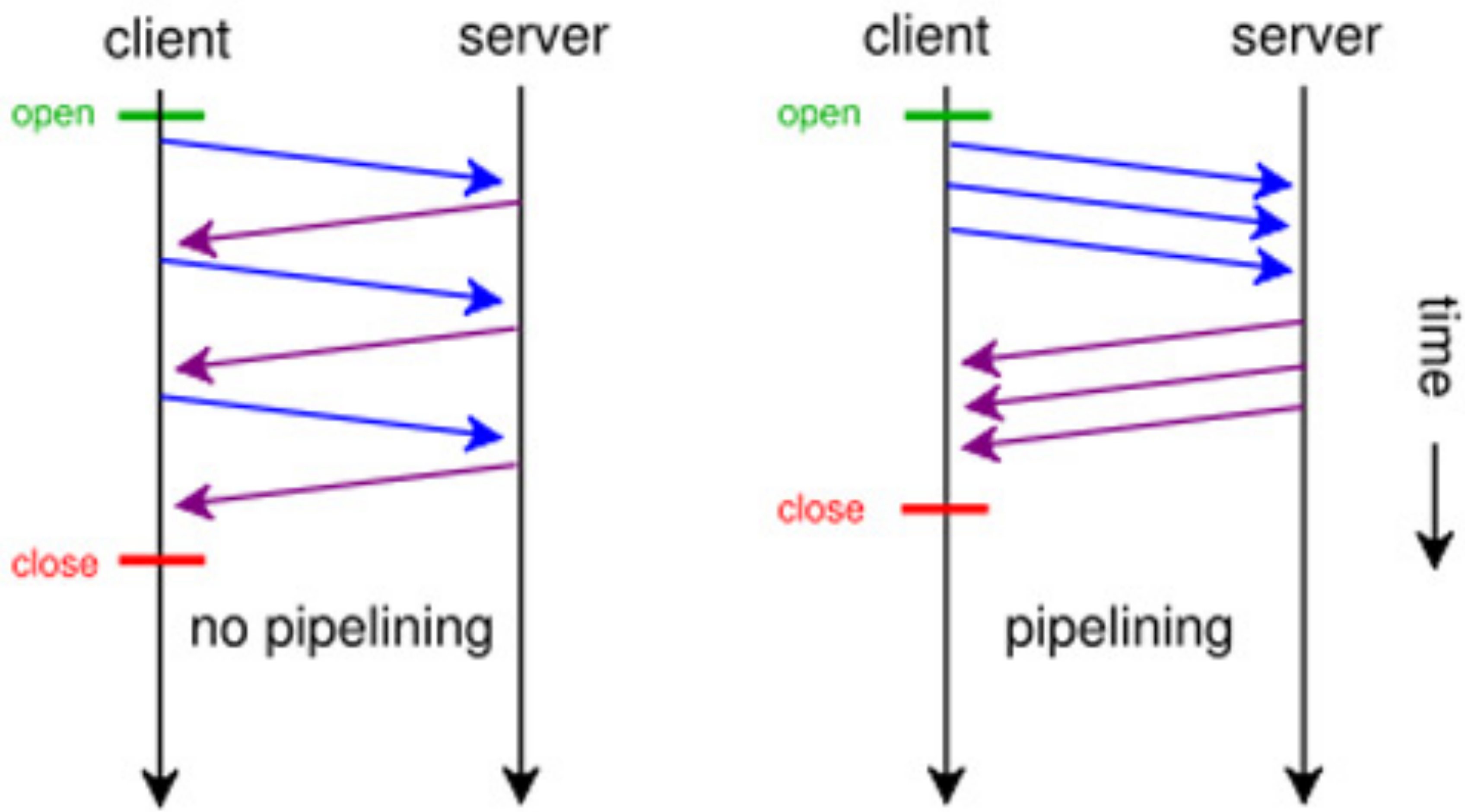
non-persistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

persistent HTTP:

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP/1.1 Pipelining



Source: http://slides.com/ipeychev/http-2-0-and-quic-protocols-of-the-near-future-and-why-they-re-important#/

Homework

Admita pretende aceder a uma página web que contém um código html base e referências a 5 objetos (imagens) que são descarregados para apresentar a totalidade da página. Assuma que RTT = 40 ms e despreze os atrasos de transmissão face a RTT.

Quanto tempo é necessário para receber a página no caso de:

a) HTTP não persistente sem conexões TCP paralelas.

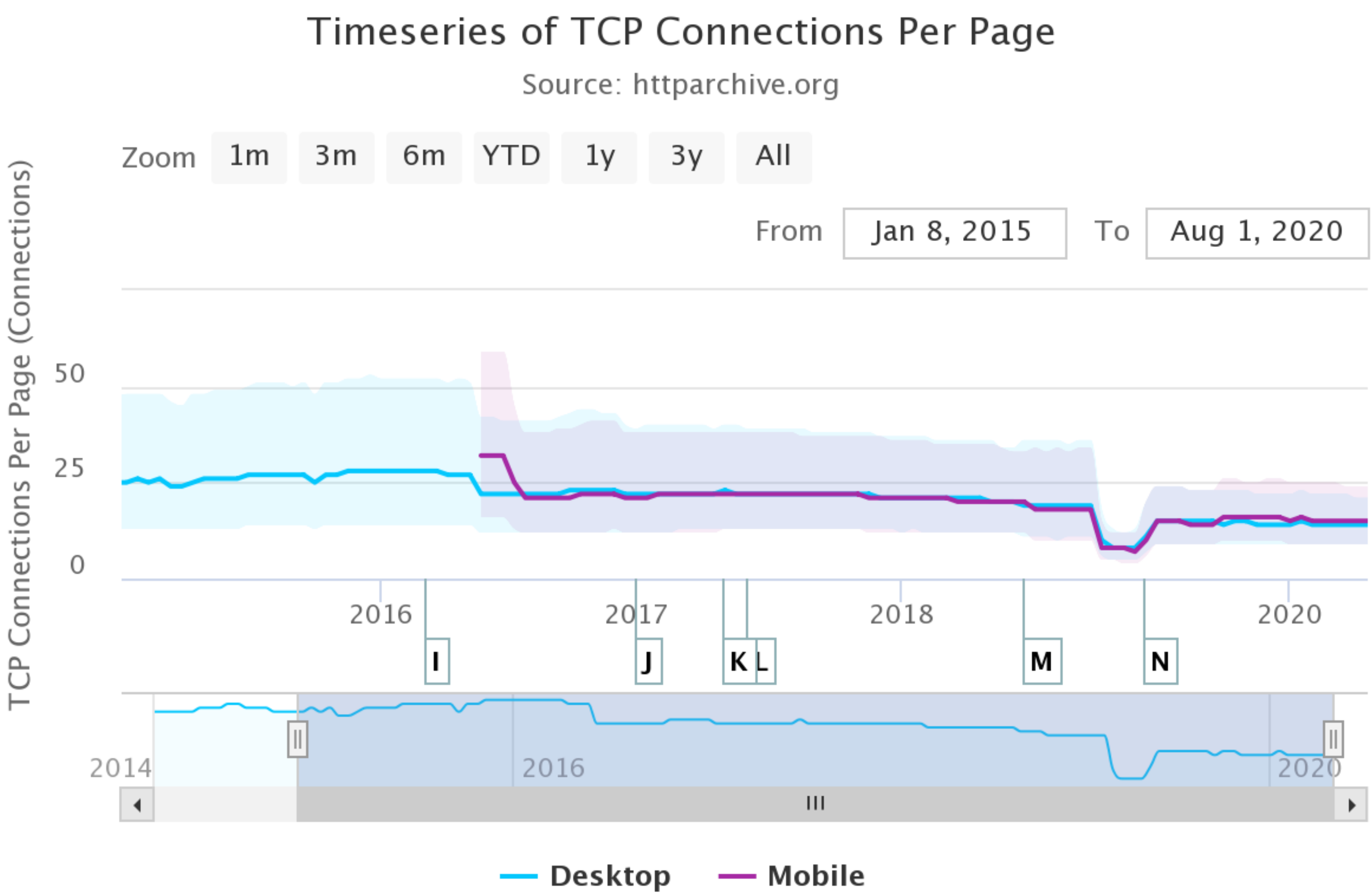
b) HTTP não persistente com 5 conexões TCP paralelas.

c) HTTP persistente com pedidos imediatos dos 5 objetos logo que sejam identificados na página, sem aguardar a respetiva receção (em pipelining - com sobreposição de transações).

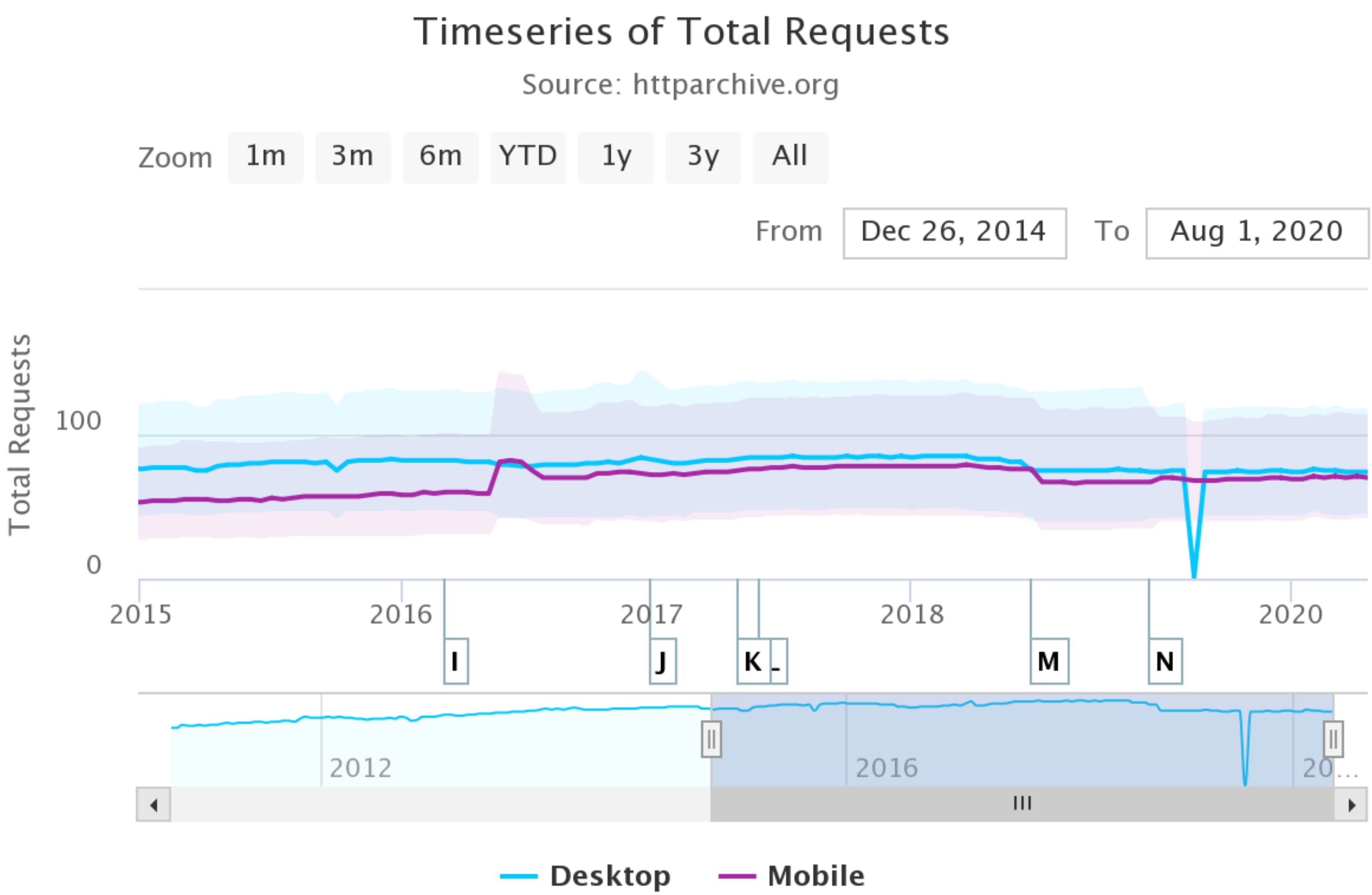
TCP Connections Summary

- HTTP 1.0
  - Non-persistent TCP connections
- HTTP 1.1
  - Persistent TCP connection
    - Lower connection setup overhead
    - Take advantage of expanded congestion window
  - Request pipelining
  - Improved caching support with new headers
    - We will see this next week
  - Others…

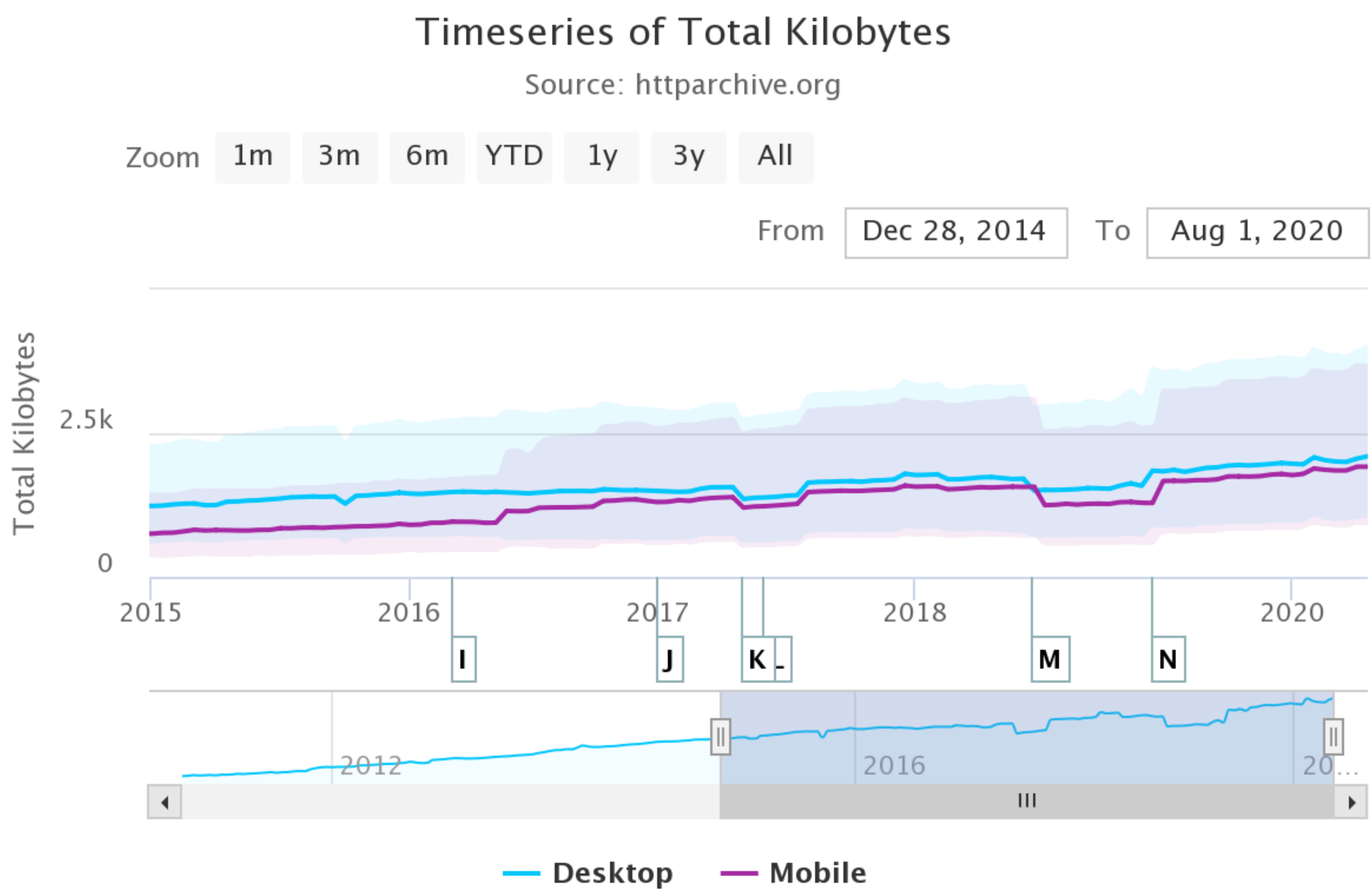
TCP Connections per Page



Objects per Page



Total Page Size



Web Page Today

- Median 75 requests per page
  - Median 3 requests for HTML
  - Median 30 requests for images
  - Median 21 requests for scripts
- From median 15 distinct hosts
- Median 2MB total page size
  - Median 27KB HTML size
  - Median 1MB image size
  - Median 500KB script size
- Most TCP flows are small

Web Page Today

- Most TCP flows are small
  - How many bytes do you need to open the window?

- What does this mean for response time?
- How to reduce latency?
  - HTTP/2
  - Content Delivery Networks (next week)
  - Both
  - Improve transport

HTTP/2.0

HTTP/2.0: Main Features

- Binary
  - To reduce overhead, on the wire and processing
- Full multiplexing of requests and responses
  - To reduce latency
- Parallelism on single TCP connection
  - To reduce network and system overhead
- Server push
  - To probabilistically reduce latency
- Header compression
  - To reduce overhead

http://cacm.acm.org/magazines/2013/12/169944-making-the-web-faster-with-http-2-0/fulltext
https://http2.github.io/

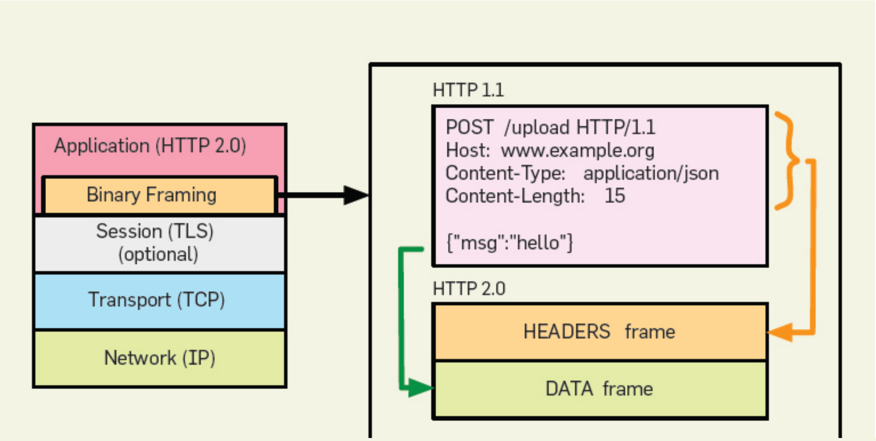
HTTP/2.0 Full Req-Rep Multiplexing

- Full request and response multiplexing
  - Stream: bi-directional byte flow
  - Message: conceptual messages that aggregates frames, e.g. a request or a response
  - Frames: basic communication unit

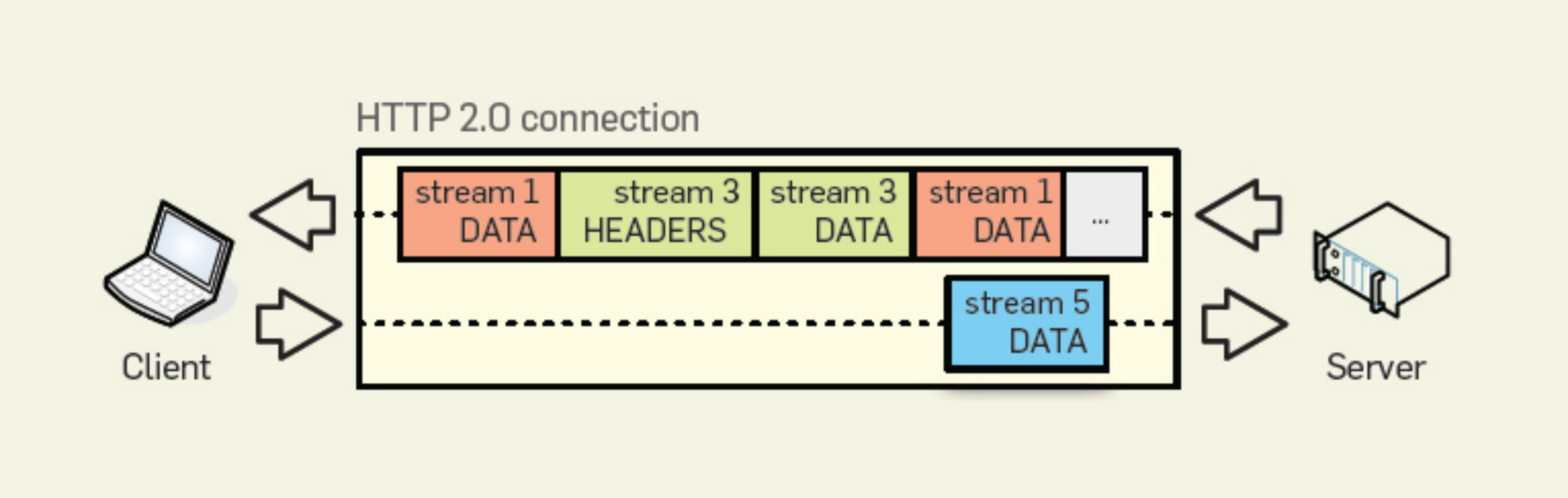
- Enables
  - Interleave multiple requests in parallel without blocking on any one.
  - Interleave multiple responses in parallel without blocking on any one.
  - Use a single connection to deliver many requests and responses in parallel.
  - Remove unnecessary HTTP 1.x workarounds from application code.

Source (also for images): http://cacm.acm.org/magazines/2013/12/169944-making-the-web-faster-with-http-2-0/fulltext

HTTP/2.0



Application (HTTP 2.0)
Binary Framing
Session (TLS)
(optional)
Transport (TCP)
Network (IP)
HTTP 1.1
POST /upload HTTP/1.1
Host: www.example.org
Content-Type: application/json
Content-Length: 15
{"msg":"hello"}
HTTP 2.0
HEADERS frame
DATA frame



HTTP 2.0 connection
stream 1 DATA
stream 3 HEADERS
stream 3 DATA
stream 1 DATA
...
stream 5 DATA
Client
Server

HTTP/2.0

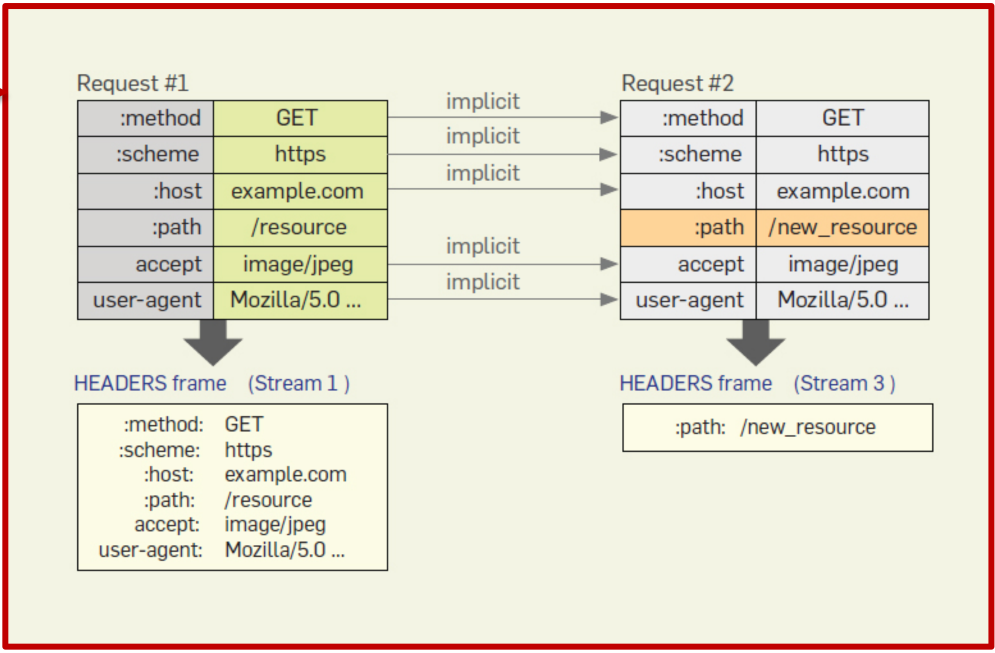
- Binary Framing
  - More compact representation
  - Easier and more efficient to process
- Separation of data and control planes

Bit	+0..7	+8..15	+16..23	+24..31
0	Length		Type (1)	Flags
32	R	Stream Identifier		
64	R	Priority		
...	*Header Block*			

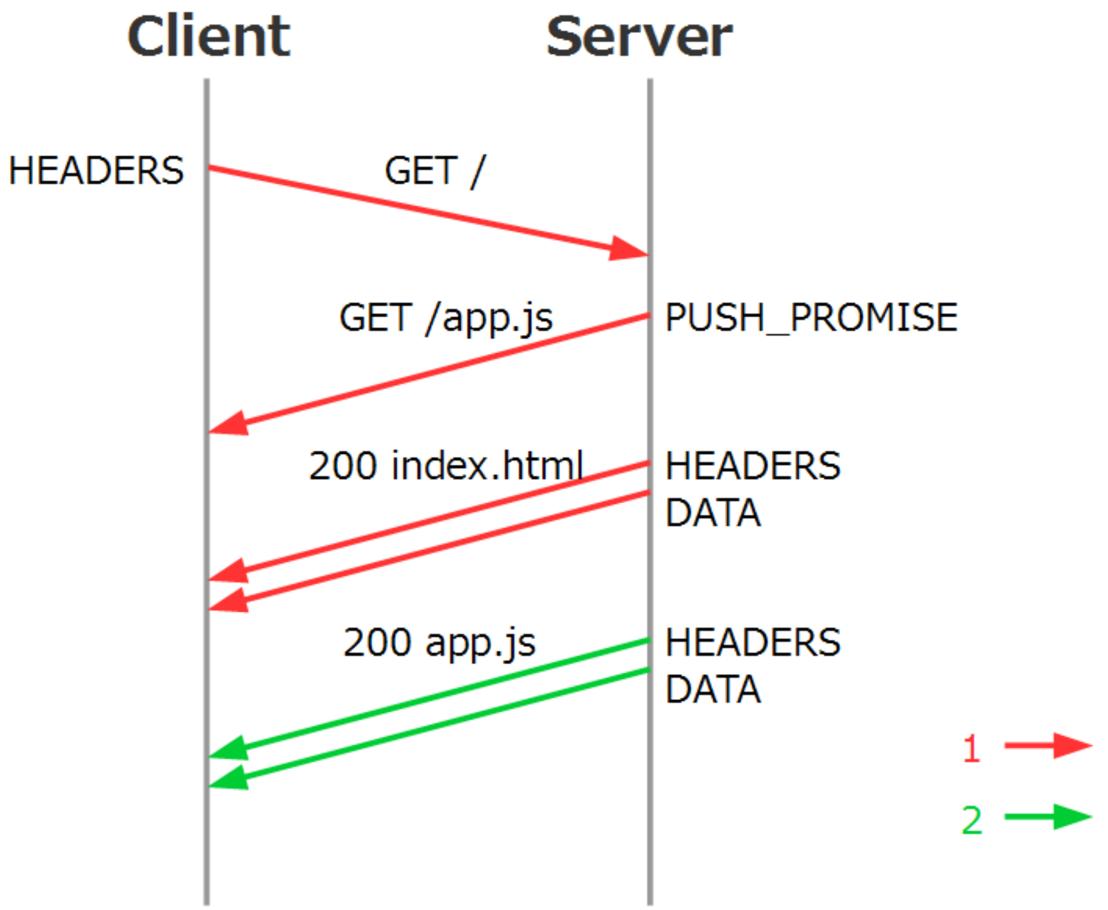
Bit	+0..7	+8..15	+16..23	+24..31
0	Length		Type (0)	Flags
32	R	Stream Identifier		
...	*HTTP Payload*			

HTTP/2.0

- Stream prioritisation
- Server push
  - Send several replies to a single request
- Header compression
- Flow control support
- HTTP 2.0 upgrade and discovery

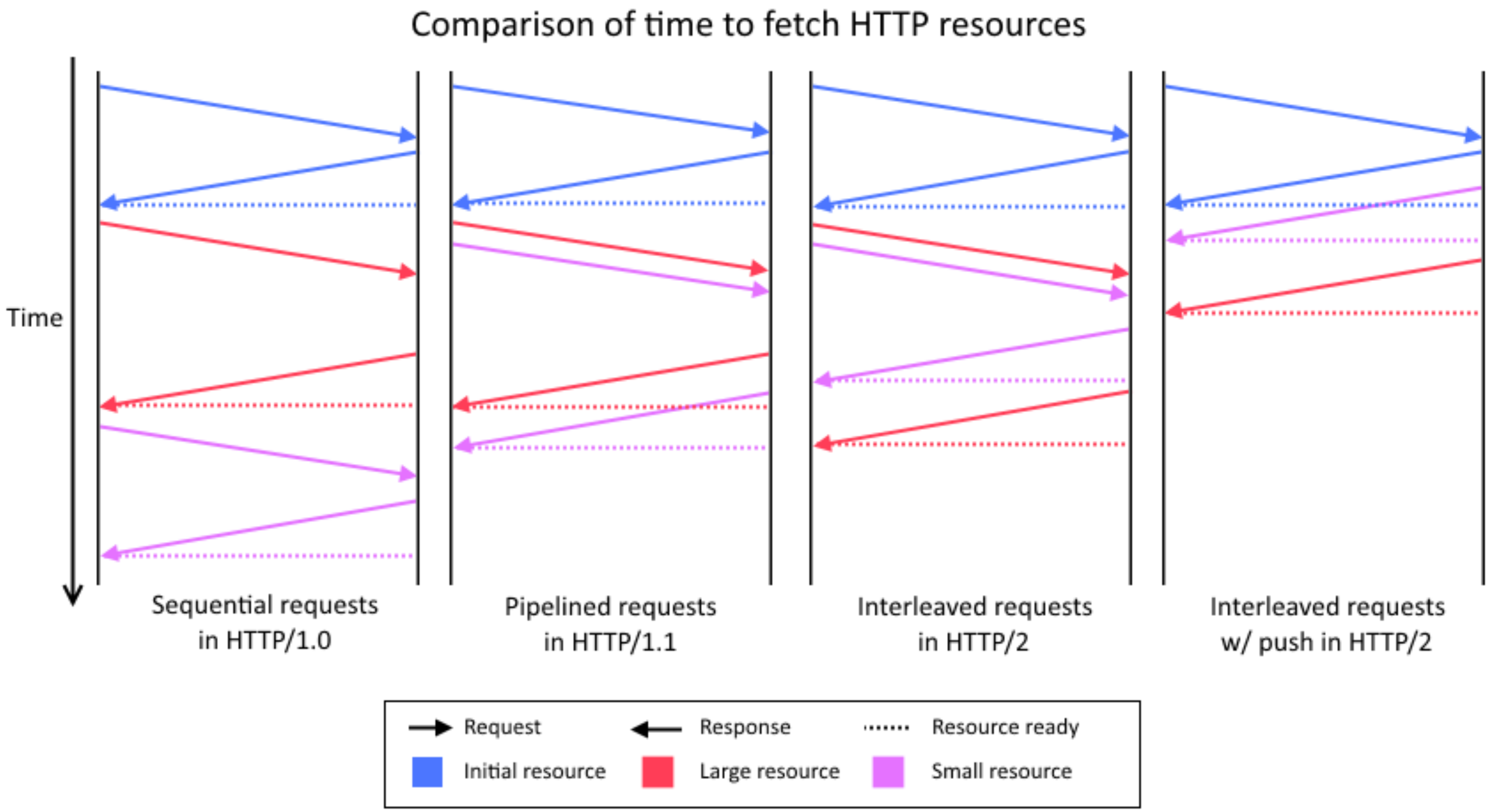


GET w Multiple Elements



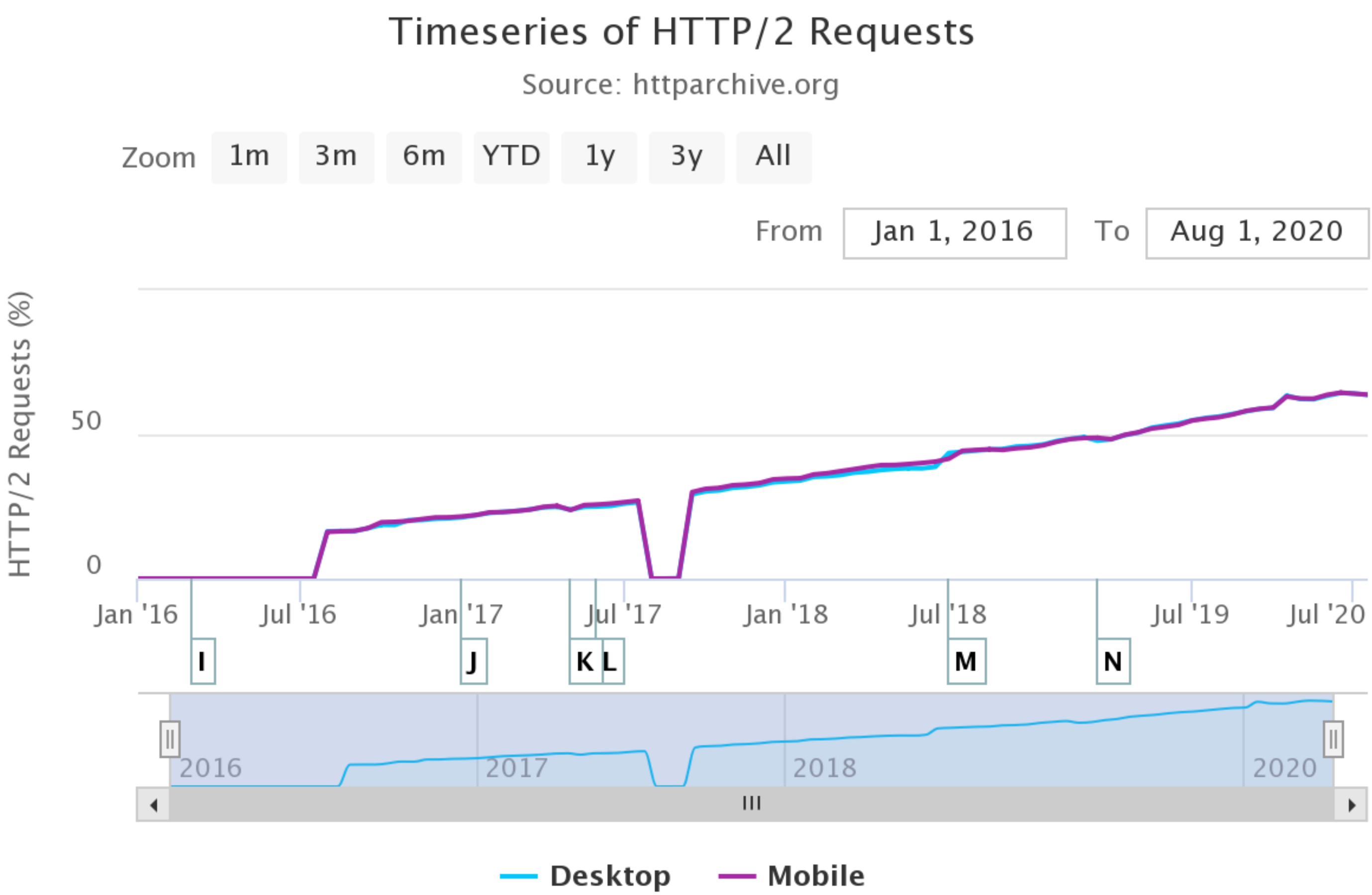
Source: http://jinjor-labo.hatenablog.com/entry/2015/02/27/231753

HTTP/2.0



From blog.scottlogic.com

HTTP/2



Timeseries of HTTP/2 Requests
Source: httparchive.org
Zoom
1m
3m
6m
YTD
1y
3y
All
From
Jan 1, 2016
To
Aug 1, 2020
HTTP/2 Requests (%)
50
0
Jan '16
Jul '16
Jan '17
Jul '17
Jan '18
Jul '18
Jul '19
Jul '20
I
J
K
L
M
N
2016
2017
2018
2020
Desktop
Mobile

Next

CACHING, CONTENT DELIVERY NETWORKS