



CIS 553: Networked Systems

HTTP

April 1, 2020



Announcements

- Sorry for the additional change, but we are now back to group projects
 - HW4 stays the same
 - HW5/6 are in pairs; timeline adjusted
 - HW6 still has no demo; timeline adjusted
- You should find a group by 10pm April 8th
 - <https://forms.gle/P7TVJiANoEoHFgM7A>
 - Tips when interviewing potential group members:
 - Ask about time zone
 - Ask about pass/fail



Agenda

- Transmission Control Protocol ✓
 - Congestion control ✓
 - Fairness ✓
 - ACK Clocking ✓
 - Router-assisted congestion control ← NEXT
- HTTP



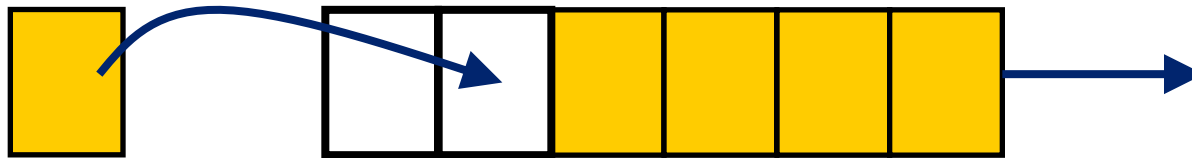
TCP has lots of problems!

- Misled by non-congestion losses
- Fills up queues leading to high delays
- Tight coupling with reliability mechanisms
- Short flows complete before discovering available capacity
- AIMD impractical for high speed links
- Saw tooth discovery too choppy for some apps
- Unfair under heterogeneous RTTs
- End hosts can cheat



Typical queue management policy

- Access to the bandwidth: first-in first-out queue
 - Packets only differentiated when they arrive



- Access to the buffer space: drop-tail queuing
 - If the queue is full, drop the incoming packet





Alternative 1: Fair Queueing

- Routers classify packets into “flows”
 - Let’s assume flows are TCP connections
- Each flow has its own FIFO queue in router
- Router services flows in a fair fashion
 - When line becomes free, take packet from next flow in a fair order
- Provides max-min fairness

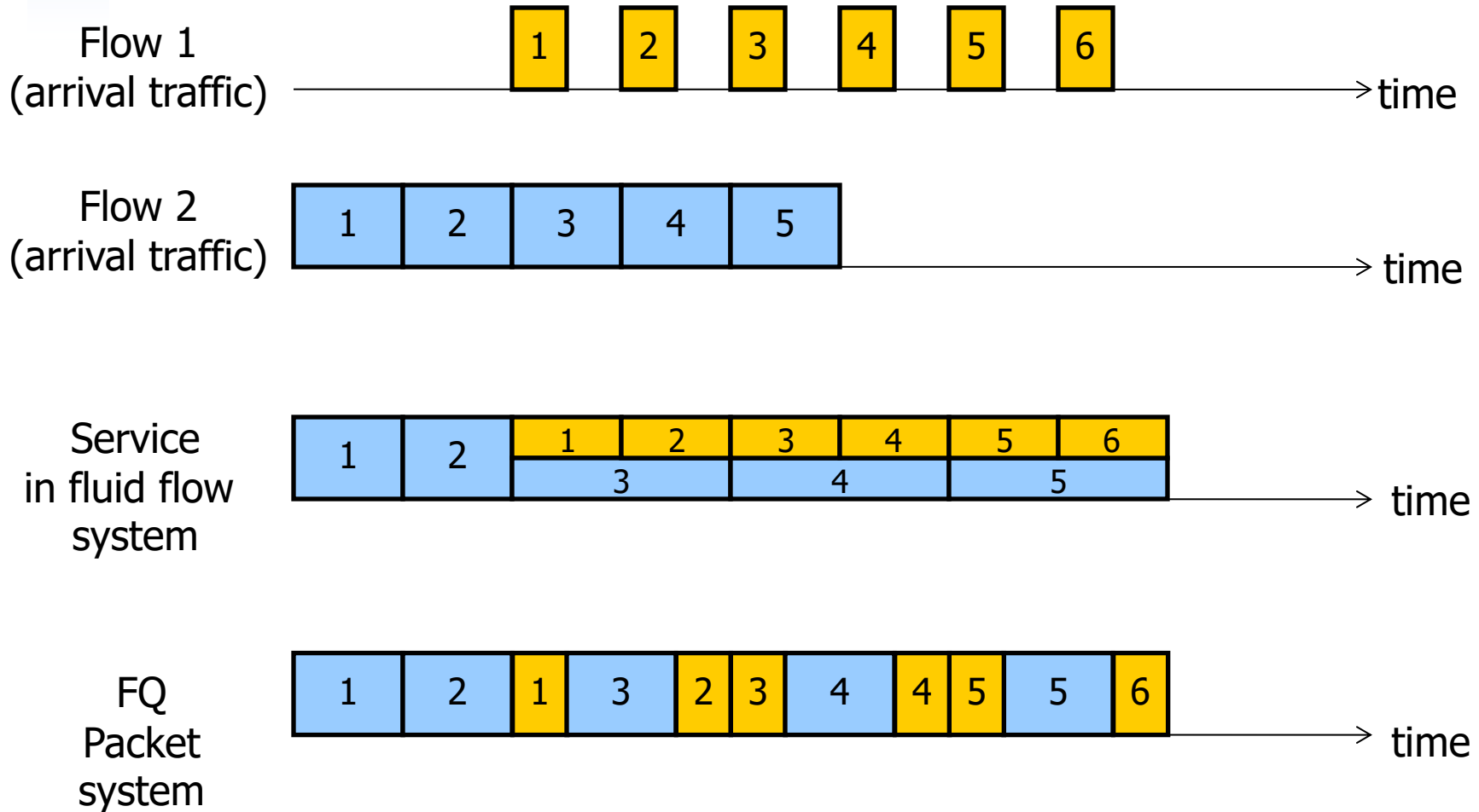


How do we deal with packets of different sizes?

- Mental model: Bit-by-bit round robin (“fluid flow”)
- Can you do this in practice?
 - No, packets cannot be preempted
- But we can approximate it
 - For each packet, compute the time at which the last bit of a packet would have left the router if flows are served bit-by-bit
 - Then serve packets in the increasing order of their deadlines



Example





Fair Queuing (FQ)

- Implementation of round-robin generalized to the case where not all packets are equal sized
- **Weighted fair queuing (WFQ)**: assign different flows different shares
- Today, some form of WFQ implemented in almost all routers
 - Not the case in the 1980-90s, when CC was being developed
 - Mostly used to isolate traffic at larger granularities (e.g., per-prefix)



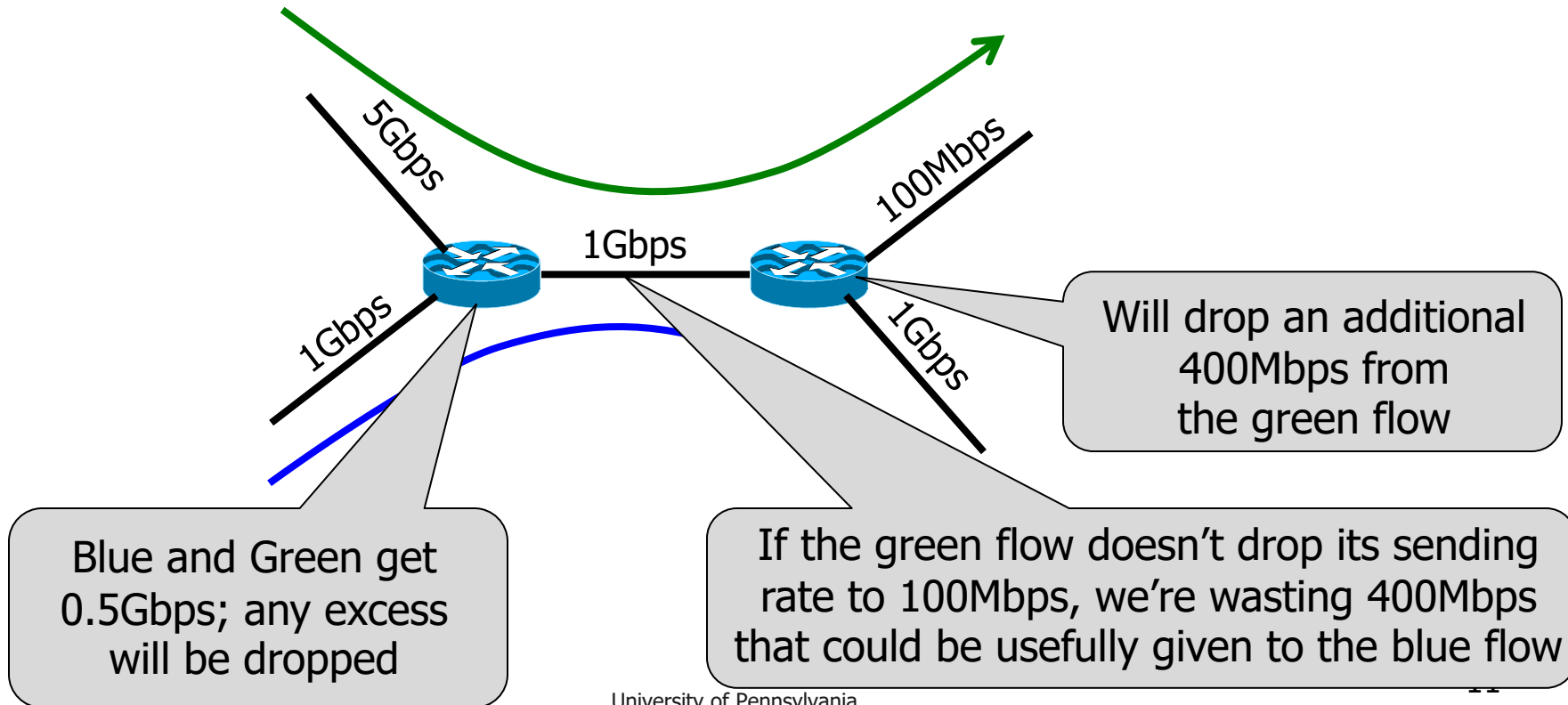
FQ Tradeoffs

- FQ advantages:
 - Isolation: cheating flows don't benefit
 - Bandwidth share does not depend on RTT
 - Flows can pick any rate adjustment scheme they want
- Disadvantages:
 - More complex than FIFO: per flow queue/state, additional per-packet book-keeping



FQ in the big picture

- FQ does not eliminate congestion → it just manages the congestion






Alternative 2: Explicit Congestion Notification (ECN)

- Single bit in packet header; set by congested routers
 - If data packet has bit set, then ACK has ECN bit set
- Many options for when routers set the bit
 - Tradeoff between (link) utilization and (packet) delay
- Congestion semantics can be exactly like that of drop
 - i.e., end-host reacts as though it saw a drop



Agenda

- Transmission Control Protocol ✓
 - Congestion control ✓
 - Fairness ✓
 - ACK Clocking ✓
 - Router-assisted congestion control ✓
- HTTP  **NEXT**
 - Protocol
 - Fetching web pages
 - Optimizations



The Web: History

- World Wide Web (WWW): a distributed database of “pages” linked through Hypertext Transport Protocol (HTTP)
 - First HTTP implementation – 1990
 - Sir Tim Berners-Lee at CERN

World Wide Web

The WorldWideWeb (W3) is a wide-area [hypermedia](#) information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an [executive summary](#) of the project, [Mailing lists](#) , [Policy](#) , November's [W3 news](#) , [Frequently Asked Questions](#) .

[What's out there?](#)

Pointers to the world's online information, [subjects](#) , [W3 servers](#), etc.

[Help](#)

on the browser you are using

[Software Products](#)

A list of W3 project components and their current state. (e.g. [Line Mode](#) ,X11 [Viola](#) , [NeXTStep](#) , [Servers](#) , [Tools](#) , [Mail robot](#) , [Library](#) .)

[Technical](#)

Details of protocols, formats, program internals etc

[Bibliography](#)

Paper documentation on W3 and references.

[People](#)

A list of some people involved in the project.

[History](#)

A summary of the history of the project.

[How can I help ?](#)

If you would like to support the web..

[Getting code](#)

Getting the code by [anonymous FTP](#) , etc.



The Web: History

- World Wide Web (WWW): a distributed database of “pages” linked through Hypertext Transport Protocol (HTTP)
 - First HTTP implementation – 1990
 - Sir Tim Berners-Lee at CERN
 - HTTP/0.9 – 1991
 - Simple GET command for the Web
 - HTTP/1.0 – 1992
 - Client/server information, simple caching
 - HTTP/1.1 – 1996
 - Performance and security optimizations
 - HTTP/2 – 2015
 - Latency optimizations via request multiplexing over single TCP connection
 - Binary protocol instead of text
 - Server push



Web components

- Web architecture:
 - Clients
 - Servers (DNS, CDN, Datacenters)
- Many **objects** on the web:
 - HTML file, JPEG, JavaScript, etc.)
 - Objects are addressable by a URL
 - Base HTML file references objects
- Protocol for exchanging objects/info: HTTP



URL: Uniform Record Locator

- `protocol://host-name[:port]/directory-path/resource`
- Can be used to locate files
 - `https://www.cis.upenn.edu/~cis553/index.html`
- Can be used to execute programs as well...
 - `https://graph.facebook.com/me?fields=id,name,birthday,email&access_token={your-user-access-token}`



URL: Uniform Record Locator

- `protocol://host-name[:port]/directory-path/resource`
 - `protocol`: http, ftp, https, smtp, rtsp, *etc.*
 - `hostname`: DNS name, IP address
 - `port`: defaults to protocol's standard port
 - *e.g.*, http: 80, https: 443
 - `directory path`: hierarchical, reflecting file system
 - `resource`: Identifies the desired resource
- And `?query`
- And `#fragment`



Hyper Text Transfer Protocol (HTTP)

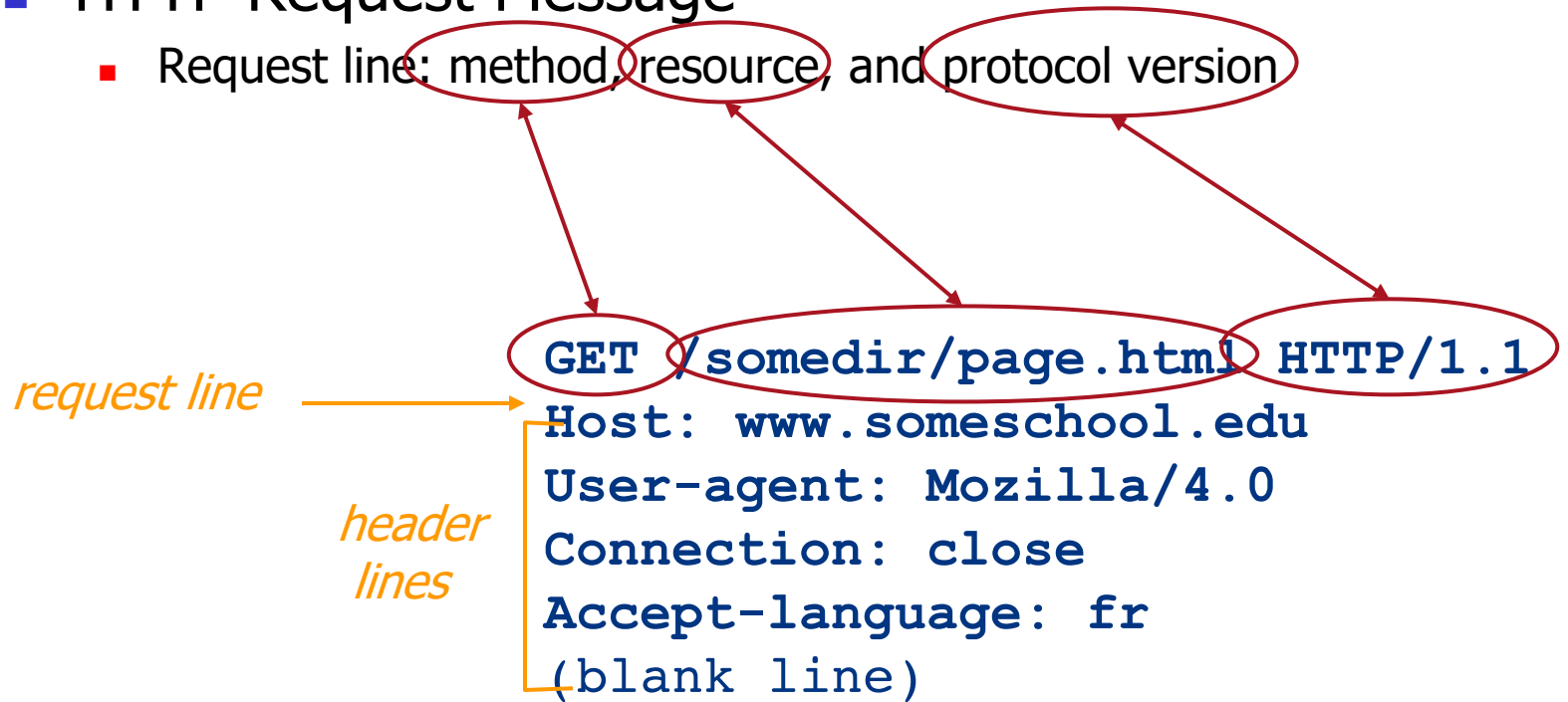
- Client-server architecture
 - Server is “always on” and “well known”
 - Clients initiate contact to server
- Synchronous request/reply protocol (before HTTP/2)
 - Runs on top of TCP/IP (before HTTP/3)
 - Default port 80 (unsecure), or 443 (secure, with SSL)
- Stateless, one message at a time (before HTTP/1.1)
- ASCII format (before HTTP/2)



HTTP Requests

- HTTP Request Message

- Request line: method, resource, and protocol version



*carriage return line feed
indicates end of message*



Common HTTP methods

- GET
 - Retrieve whatever information is identified by the URI
- HEAD
 - Like GET, but retrieves only metadata, not the actual object
- PUT
 - Store information under the specified URI
- DELETE
 - Delete the information specified by the URI
- POST
 - Adds new information to whatever is identified by the URI
 - Intended, e.g., for newsgroup posts; today, used mostly to implement dynamic content via forms



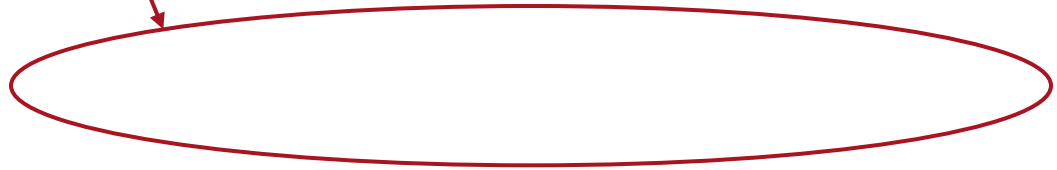
HTTP Requests

■ HTTP Request Message

- Request line: method, resource, and protocol version
- Request headers: provide info or modify request
- Body: optional data (e.g., to "POST" data to server)

request line → GET /somedir/page.html HTTP/1.1
header lines [Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
(blank line)

carriage return line feed indicates end of message





HTTP Responses

- 3 digit response code
 - 1XX – informational
 - 2XX – success
 - 200 OK
 - 3XX – redirection
 - 301 Moved Permanently
 - 303 Moved Temporarily
 - 304 Not Modified
 - 4XX – client error
 - 404 Not Found
 - 5XX – server error
 - 505 HTTP Version Not Supported



HTTP Responses

■ HTTP Response Message

- Status line: protocol version, status code, status phrase
- Response headers: provide information
- Body: optional data

status line
(protocol, status code,
status phrase)

header lines

data
e.g., requested HTML file

HTTP/1.1 200 OK

Connection close

Date: Thu, 06 Jan 2017 12:00:15 GMT

Server: Apache/1.3.0 (Unix)

Last-Modified: Mon, 22 Jun 2006 ...

Content-Length: 6821

Content-Type: text/html

(blank line)

data data data data data ...



How to Mark End of Message?

- No body content. Double CRLF marks end
 - E.g., 304 never have body content
- Close connection
 - Only server can do this
 - Even at server, one request per TCP connection hurts performance.
- Content-Length
 - Must know size of transfer in advance
- Transfer-Encoding: chunked (HTTP/1.1)
 - After headers, each chunk is content length in hex, CRLF, then body. Final chunk is length 0.



Example: Chunked Encoding

```
HTTP/1.1 200 OK <CRLF>
```

```
Transfer-Encoding: chunked <CRLF>
```

```
<CRLF>
```

```
25 <CRLF>
```

```
This is the data in the first chunk <CRLF>
```

```
1A <CRLF>
```

```
and this is the second one <CRLF>
```

```
0 <CRLF>
```

- Especially useful for dynamically-generated content, as length is not a priori known
 - Server would otherwise need to cache data until done generating, and then go back and fill-in length header before transmitting



HTTP in action

- Try it yourself:
 - As a “browser” fetching a URL
 - Run “telnet vincen.tl 80”
 - Type:
 - GET /index.html HTTP/1.1
 - Host: vincen.tl
 - [blank line]
 - Server will return HTTP response with the page contents

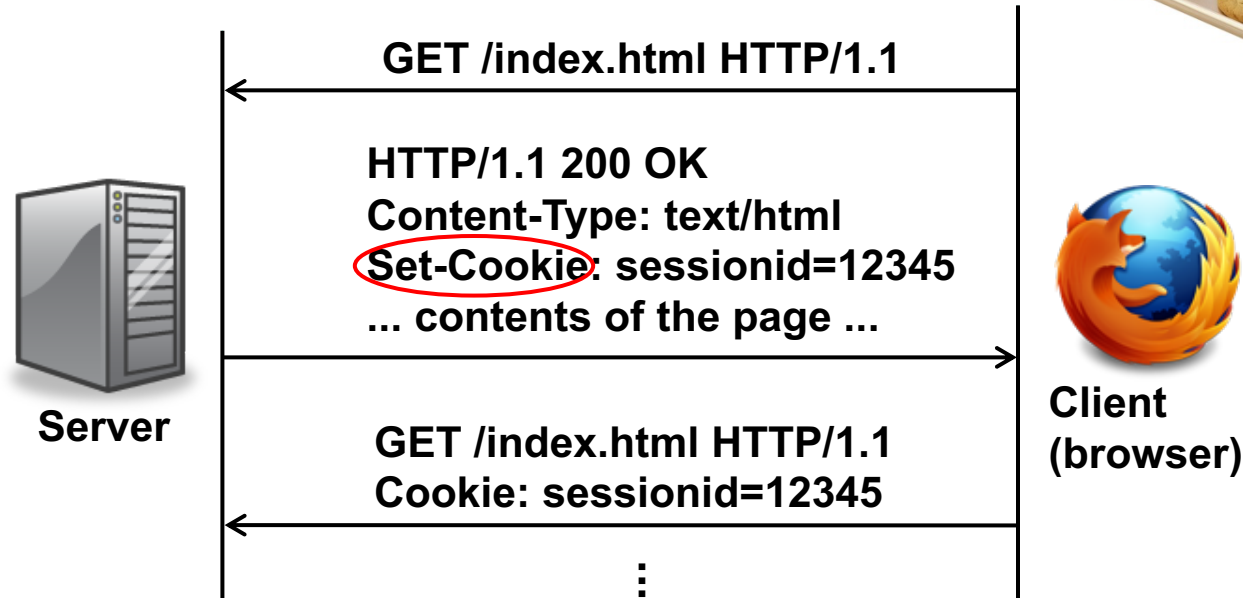


HTTP is stateless

- Each request-response treated independently
 - Servers not required to retain state
- **Good:** Improves scalability on the server-side
 - Failure handling is easier
 - Can handle higher rate of requests
 - Order of requests doesn't matter
- **Bad:** Some applications need persistent state
 - Need to uniquely identify user or store temporary info
 - e.g., Shopping cart, user profiles, usage tracking, ...



HTTP cookies



- What is a **cookie**?
 - A set of key-value pairs that a web site can store in your browser (example: 'sessionid=12345')
 - Created with a Set-Cookie header in the HTTP response
 - Browser sends the cookie in all subsequent requests to the same web site until it expires