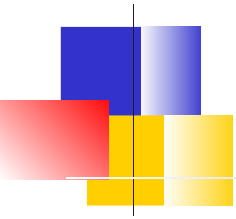


Hypertext Transport Protocol

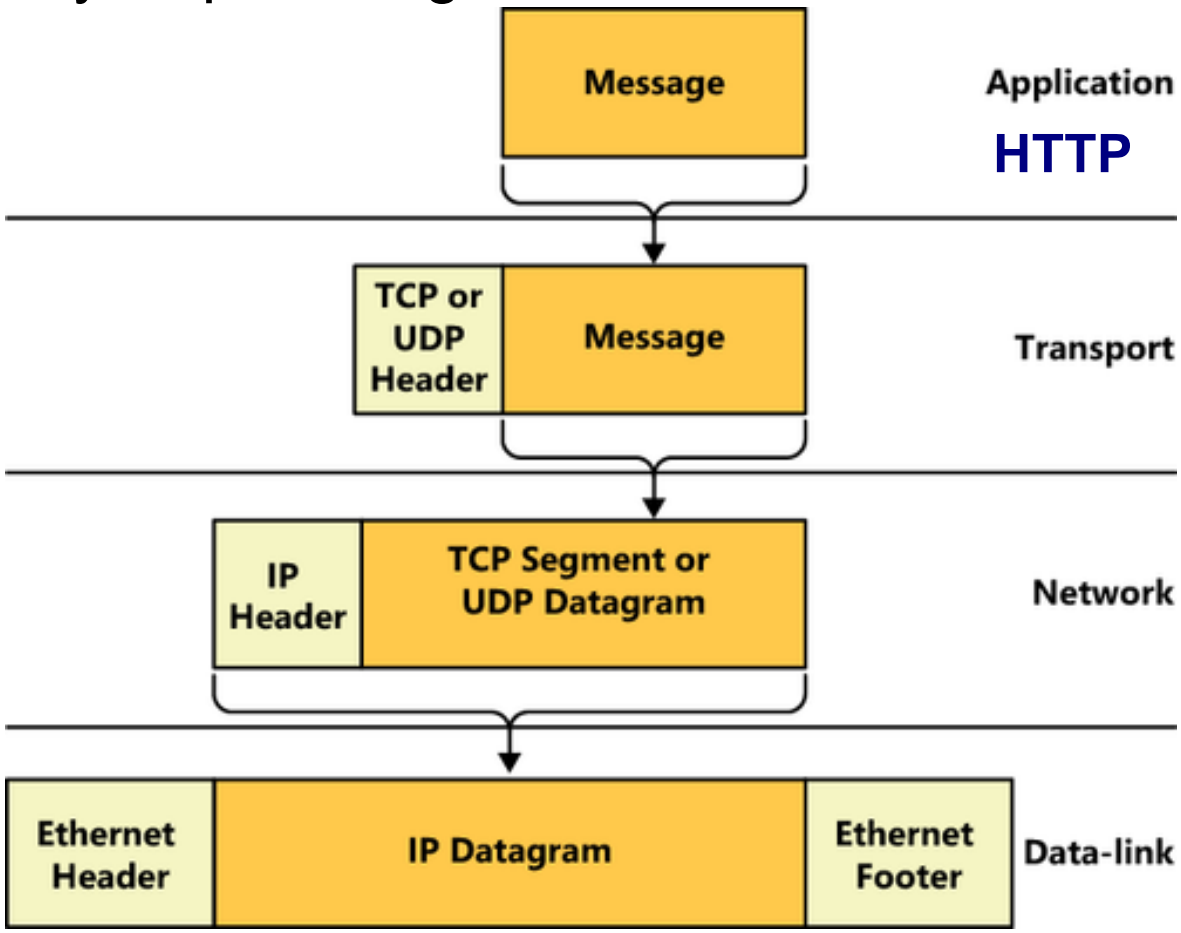
James Brucker

Hypertext Transport Protocol (HTTP)

- Protocol used to access the Web
- Mostly widely used protocol on the Internet
- Platform independent
- Human readable

HTTP uses TCP and IP

Internet protocols are divided into "layers" -- a packet inside a packet.
Each "layer" providing a different kind of functionality.



Internet Protocol (IP)

Internet Protocol (IP) transports packets over Internet.

Every device on the Internet uses the IP protocol.

IP provides:

- addressing - each site has an IP address
- routing - how to "route" a packet from source to dest.

IP does not:

- guarantee delivery of packets
- deliver packets in the order sent
- maintain a "connection" between source and dest.

IP Addresses

Every device on the Internet has an IP address

IP version 4 - 4 byte addresses

158.108.216.5 - address of www.ku.ac.th

172.217.27.228 - www.google.com (many addresses)

127.0.0.1 - "localhost". Address of your own host.

0.0.0.0 - address pattern meaning "anything"

IP version 6 - 16 byte addresses. Newer version of IP.

2406:3100:1010:100:0:0:0:5 - www.ku.ac.th

2406:3100:1010:100**::**5 - same thing, 0-bytes omitted

2404:6800:4001:80e**::**2004 - www.google.com (many)

What's My IP Address?

To see your local IP addresses:

```
linux>   ifconfig [interface_name]
windows> ipconfig [interface_name]
```

You can also find this in your network settings.

To see your public IP address visible on the Internet (which may be *translated* by your router or ISP):

Chrome: `my ip address` (space required)

Any Browser: `myipaddress.com`

`ipinfo.io` (Cloudflare, identifies any IP)

IPv4 May Differ: `whatismyipaddress.com`, `myip.com`

IP Transports a "Payload"

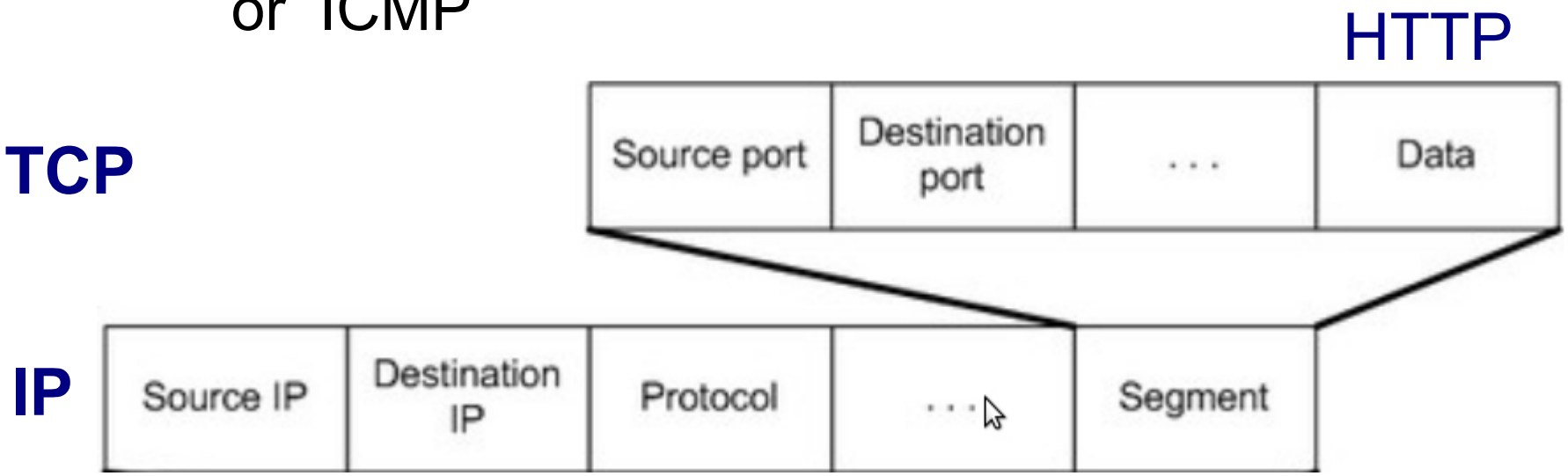
IP transports data between hosts.

The payload (data) uses some other protocol.

Usually: TCP,

UDP,

or ICMP



TCP for "Reliable" Communication

Transmission Control Protocol (TCP) maintains a virtual connection between hosts.

TCP adds functionality to IP:

- maintains a "virtual connection" between hosts
- ensures packets are delivered in the order sent
- request resend of any missing data
- identifies a "**port**" to deliver data to

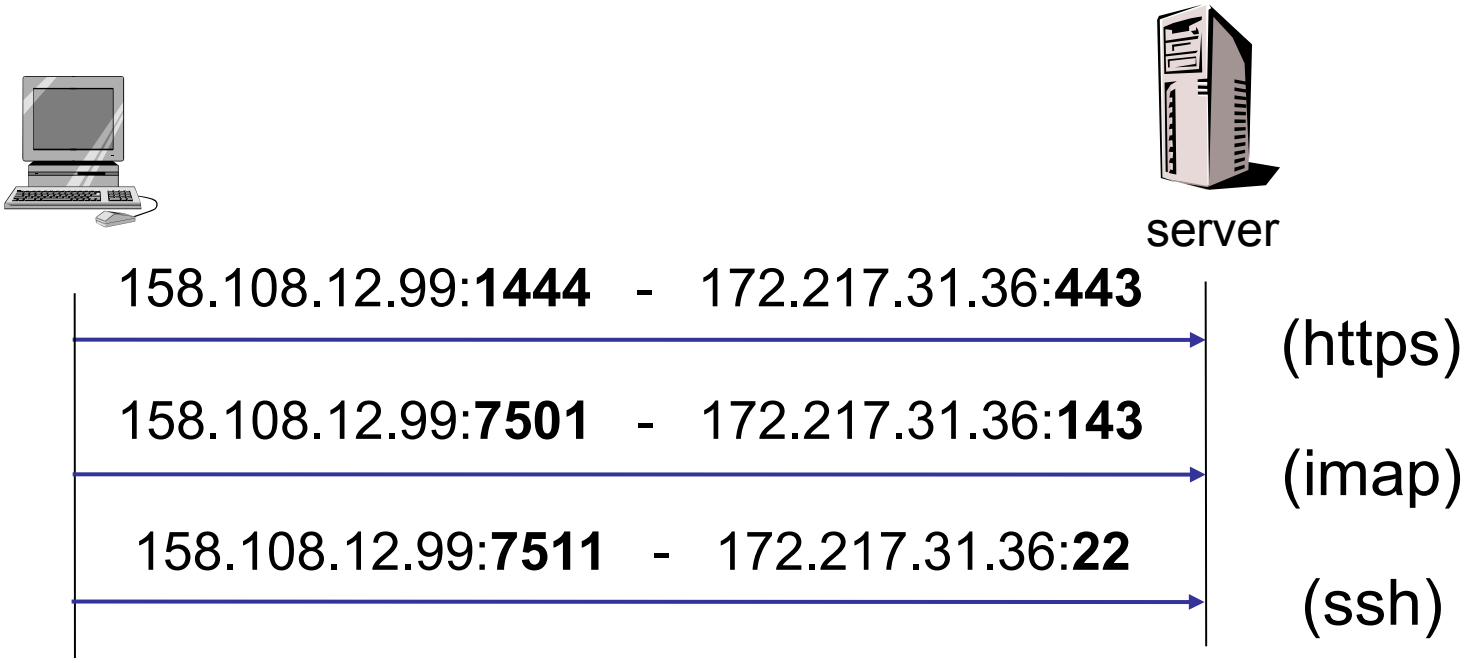
The TCP "payload" is a message using some higher level protocol!

What's a Port? Why do we need ports?

- A host may have many, many of internet connections at the same time!
- A server may offer many services: HTTP, mail, ssh, ...
- How does a host know which packets should go to which application???

Port is a number to identify connection

- A connection has a **port number** 1 - 65,535 for each end point.
- Servers ***listen*** for connections on well-known port nums.
- Each `ip_address:port` pair identifies an endpoint.



Port Numbers Identify *Services*

Standard services are assigned a unique port number

Service	TCP Port
HTTP	80
HTTPS	443
Mail Transport (SMTP)	25, 465 (secure)
IMAP (client mail delivery)	143, 993 (secure)
SSH	22
MySQL server*	3306

See /etc/services *or Wikipedia for more services.*
** For security, you should not expose a database service to the Internet.*

A Service Can Use *Any* Port

Web servers usually use port 80 (http) and 443 (https).

But you can use **any port** for your web server.

Django development server listens on **port 8000** by default...

but you can tell it to use *any* port.

Ports 1-1023 are *privileged ports*. Only "root" or admin user can start a process on those ports.

Exercise 1: View your connections

1. In a terminal window type:

   Linux/MacOS> **`netstat -n --tcp`**

   Windows> **`netstat -n -p tcp`**

2. In a web browser, visit a new web site.

3. Type "netstat" again ... are there new connections?

"**`-n`**" means show IP address instead of host name.
Omit -n to show host names, but it is slower.

Exercise 2: Create Your Own Server

Use netcat (nc) or ncat for this:

1. Open a terminal window and start a server. **`-l`** means "listen", 4444 is port number. Any port > 1024 is ok.

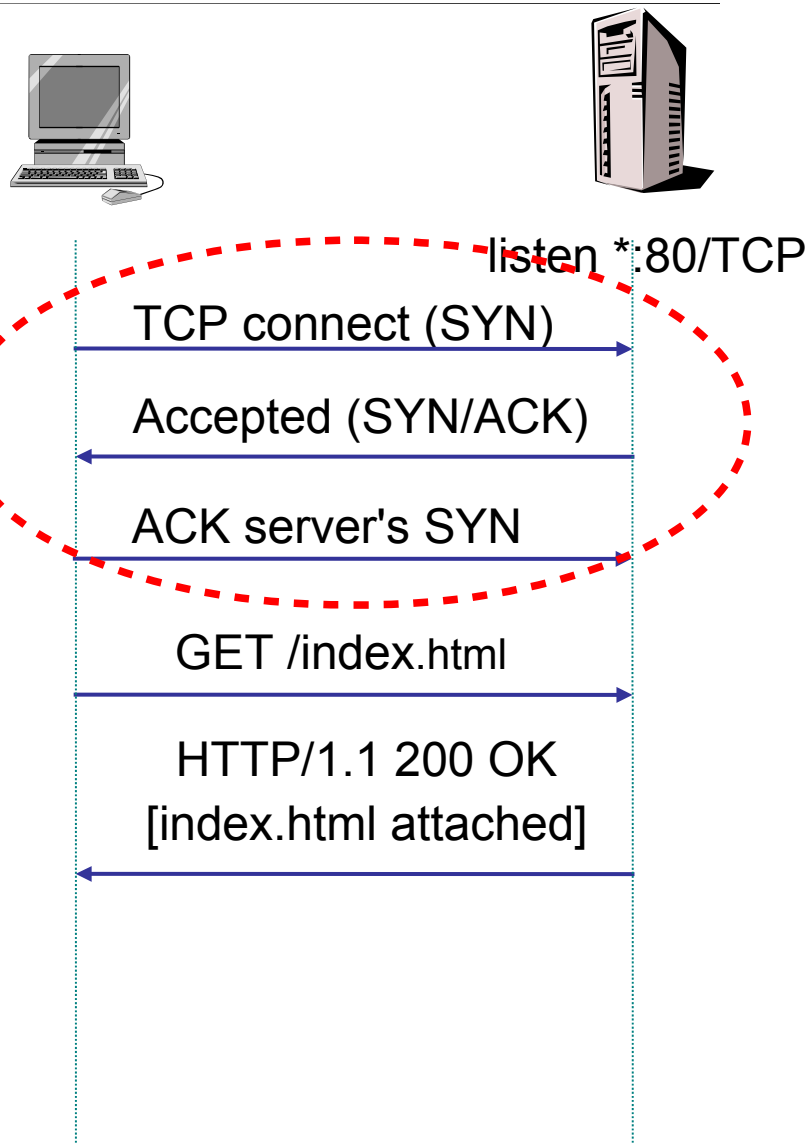
```
cmd> netcat -v -l -p 4444
```

2. Open another terminal window and connect to "localhost" on port 4444. Type something...

```
cmd> netcat localhost 4444
Hello?  Is anyone there?
```

Establishing a TCP Connection

- TCP needs **3 packets** just to establish a connection
- This adds delay and overhead
- When delay or overhead are important, use a different protocol, e.g. UDP

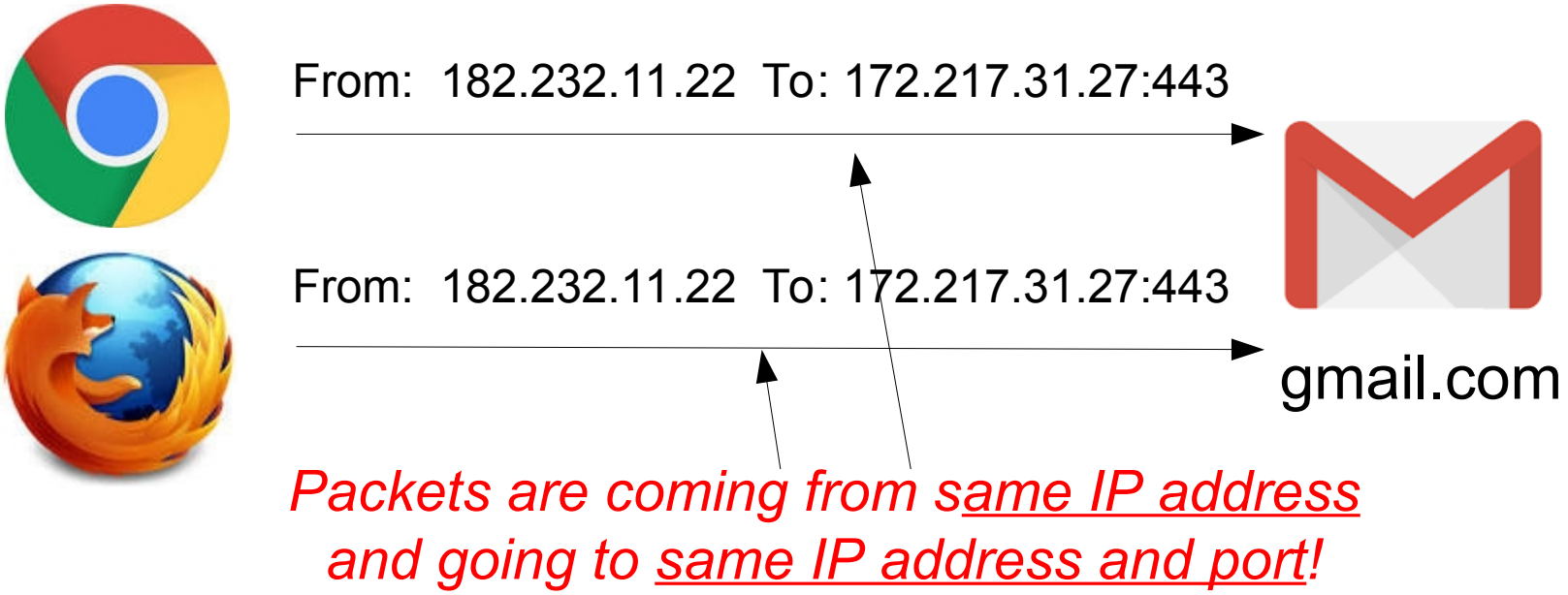


Test Your Understanding

You can have many connections to the same service!

1. Open Chrome and connect to gmail.com
2. Open Firefox or Safari and connect to gmail.com

What the server sees:

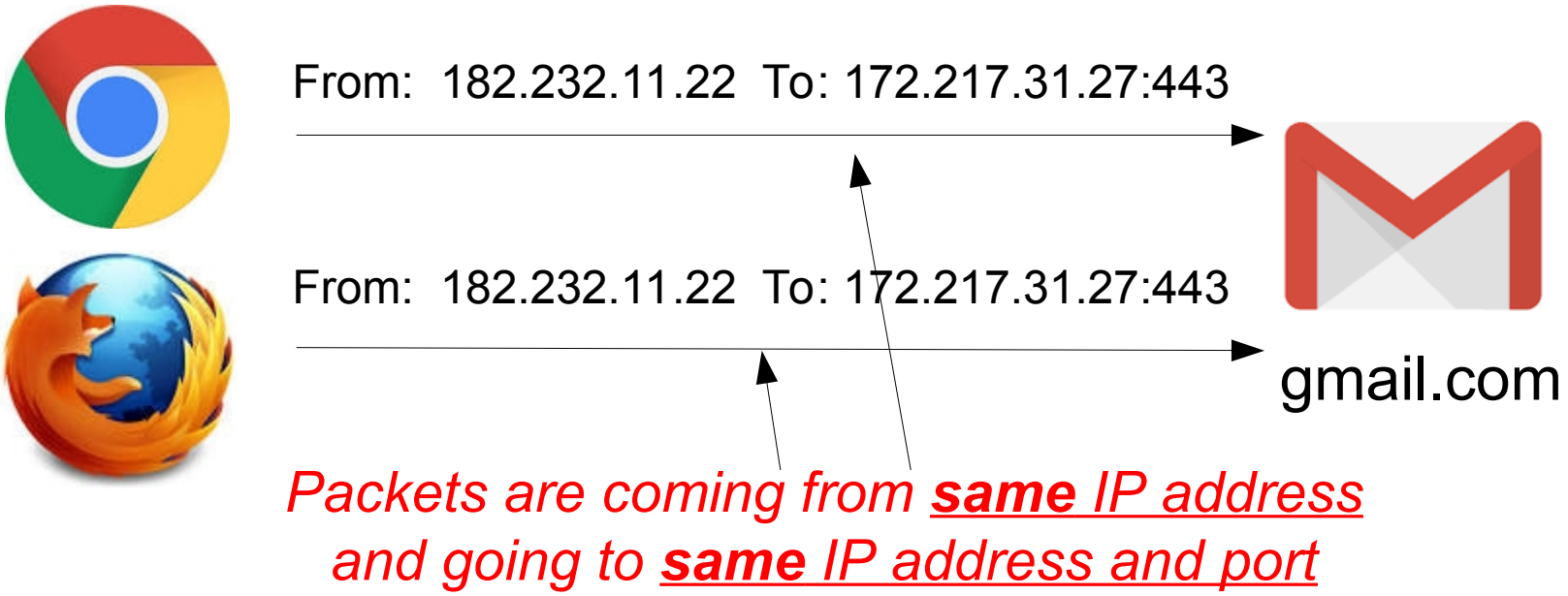


Where to Send a Reply?

How does the server distinguish the two?

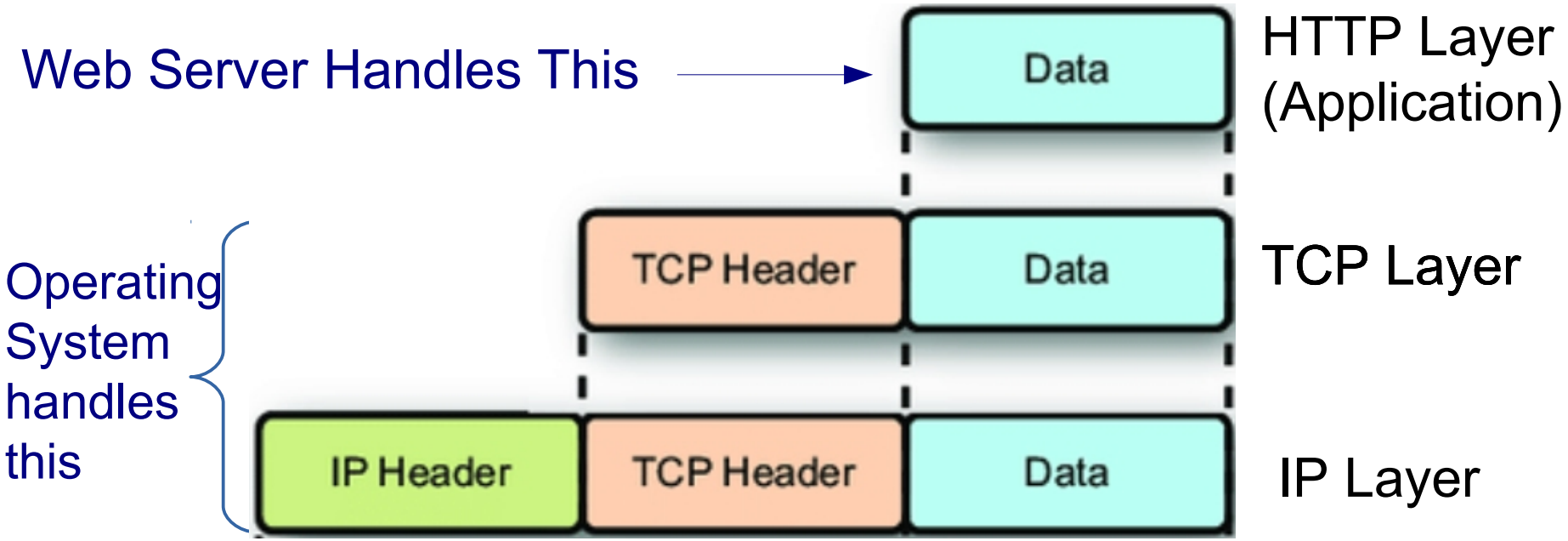
Requests from each browser have same src & dest.

Could Gmail get "confused" and send a reply to the wrong browser? Explain.



HTTP uses TCP

- HTTP uses TCP for connection and IP for transport
- TCP/IP connections are managed by the OS.
- Web Server handles only the HTTP message

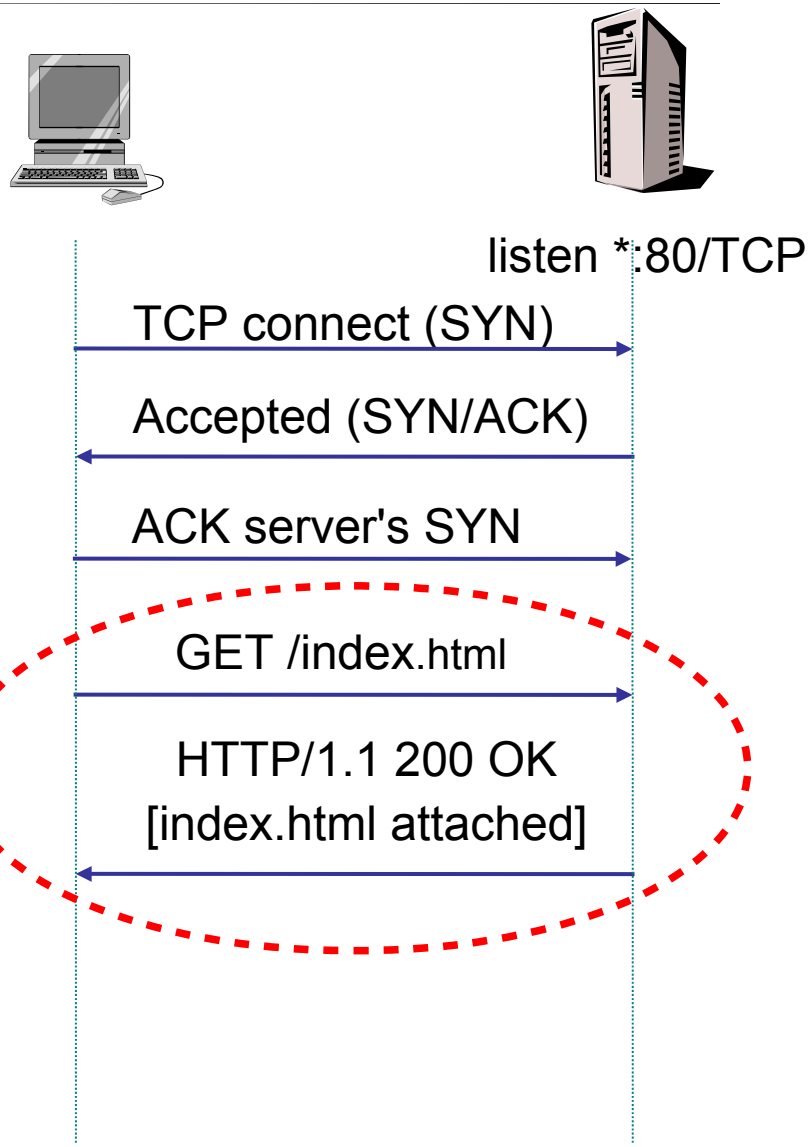


HTTP is Request / Response Protocol

- Client sends an HTTP **request**, server sends a **response**
- Server listens (waits) for incoming requests.
- Server is *stateless* - not required to remember any previous requests or connections (but web apps may).

HTTP Request - Response

- HTTP 1.0: one request/reply. Connection closed immediately.
- HTTP 1.1 allows *persistent* connections (many requests) and data compression for performance
- HTTP/2.0 is much faster: header caching, overlapping requests, better compression



HTTP Protocol Basics

1. HTTP Request format
2. HTTP Request methods
3. HTTP Response format
4. Header fields
5. Response codes (status codes)
6. URLs

HTTP Request Example

In browser enter: http://www.cpe.ku.ac.th/index.html

```
GET /index.html HTTP/1.1
Host: www.cpe.ku.ac.th
User-Agent: Mozilla/5.0
Accept: text/html, text/plain, image/gif,
  image/jpeg
Accept-Language: en, th;q=0.5
Accept-Charset: ascii, ISO8859-1, ISO8859-13
Accept-Encoding: gzip,deflate
```

Two CR/LF (one empty line) indicates end of headers

Accept: includes "**text/plain**" or "*/*" as a last resort.

HTTP Request Format

```
METHOD /relative-url HTTP/1.1
Host: server.host.name
Header1: xxxx
Header2: yyyy
```

Blank Line (CR/LF) indicates end of headers

REQUEST BODY (POST and PUT only)

Only POST and PUT requests have a REQUEST BODY

HTTP Request Methods

GET	get the *resource* specified by URL
POST	send information to server using body may have side effects; not repeatable
PUT	save or update resource at the given URL used to create or update resource at URL
DELETE	delete specified URL
OPTIONS	request info about available options
HEAD	retrieve meta-information about URL (used by search engines & web crawlers)
TRACE	trace request through network
CONNECT	connect to another server; used by proxies

Common Request Headers

`Accept`: text/html,application/xhtml+xml,text/plain

`Accept-Language`: en-US,en-GB;q=0.5

`Accept-Encoding`: gzip, deflate

`Host`: www.google.com

`User-Agent`: Mozilla/5.0

`Connection`: Keep-Alive

`Content-Length`: 2048 (for POST and PUT)

`X-Powered-By`: Godzilla (X- = *custom headers*)

w3schools.net and httpwatch.com have a longer list.

RFC2616: http://www.w3.org/Protocols/rfc2616/rfc2616.html

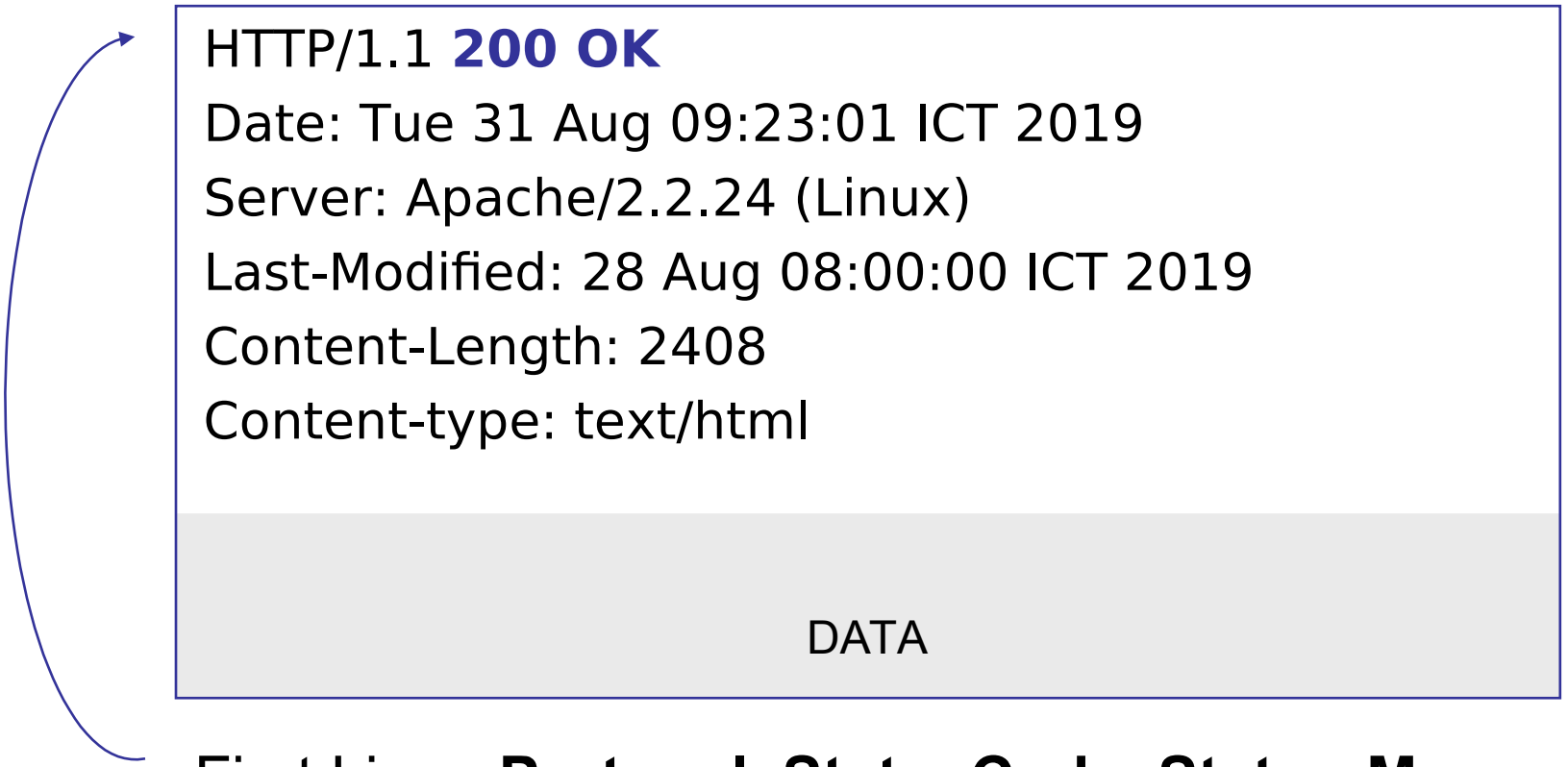
HTTP Response Example

```
HTTP/1.1 200 OK
Date: Mon, 28 Jul 2019
Server: Apache/2.2.24
Keep-Alive: timeout=5,max=100
Content-Type: text/html
Content-Length: 240

<html>
<head>blah blah</head>
<body>content of the page</body>
</html>
```

Blank Line (CR) indicates end of headers

HTTP Response Format



First Line: **Protocol StatusCode Status-Msg**

Response Content-Length

HTTP/1.1 **200 OK**
Date: Tue 31 Aug 09:23:01 ICT 2019
Server: Apache/2.2.24 (Linux)
Last-Modified: 28 Aug 08:00:00 ICT 2019
Content-Length: 16400
Content-type: image/jpeg

DATA (16400 Bytes)

For persistent connections, client needs to know how much data is in the response.
Example: server sends a JPEG file How many bytes is it?

Client uses the `Content-Length` header.

Unknown Content Length

If content length is not known by server, it uses the header "Connection: close".

After the response is sent, server closes the connection.

This way, the client can read data until end-of-input.

Response Codes

```
HTTP/1.1 200 OK
```

Response Codes:

1xx Information

100 Continue

2xx Success

200 OK

201 Created (a new resource was successfully created)

202 Accepted (I'll process your request later)

3xx Redirection

301 Moved Permanently. New URL in `Location` header.

302 Moved Temporarily. New URL in `Location` header.

303 Redirect and change POST to GET method

304 Not Modified ("*Look in your cache, stupid*")

Error Response Codes

4xx Client Error

400 Bad Request

401 Not Authorized (client not authorized to do this)

404 Not Found

5xx Server Error

500 Internal Server Error (application error, config prob.)

503 Service Unavailable

List of all HTTP status codes:

http://stat.us

http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Why is "Host" header required?

HTTP Requests always include a "**`Host`**" header.

It is the name of the *destination* host.

```
GET /index.html HTTP/1.1
Host: www.ku.ac.th
```

WHY?

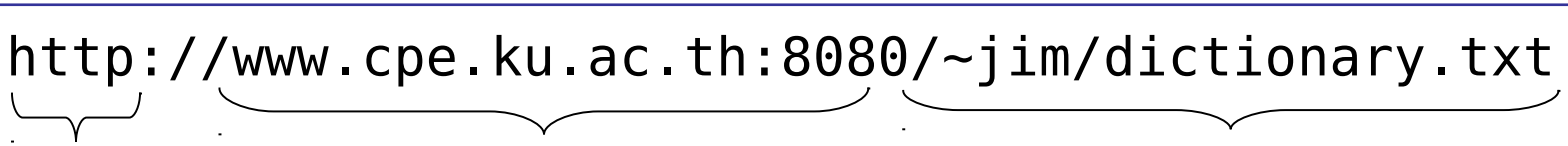
Surely, the server must know its own host name!

… or does it?

Uniform Resource Locators (URL)

A Uniform Resource Locators (URL) locate resources on the Internet (not just the web).

Structure of a URL:



Protocol:	Hostname and port	Path and resource name,

`http`
`ftp`
`jdbc`
`file`
`mysql`
	or IP address
	Port is optional
		the path is optional

Uniform Resource Locator

http://www.cpe.ku.ac.th/forms/junk.html?
name=jim%40.cpe.ku.ac.th&msgid=0x4412858798

General Form of a Uniform Resource Location (URL)

`http://www.cpe.ku.ac.th:80/~jim/images/cat.jpg`

Protocol:

`http`
`ftp,...`

Authority

hostname or
IP address

Port

Path and resource name,

the path is optional

URL Details

Encode special characters using %

http://host.com/web svc becomes:

http://host.com/seb%20svc

Path Parameters - extra info in path segment

`http://finger.com/person;name=joe/telephone;co=th`

Query Parameters - used for GET

http://host.com/adduser.cgi*?name=joe&age=23*

URL for File, URL with user info

Use a web browser to open a FILE on your computer:

```
file:///home/me/workspace/unittesting/fraction.py
You can omit "//" since there is no host:
file:/home/me/workspace/unittesting/fraction.py
```

May include user info in a URL:

```
protocol://username:password@hostname/...
http://jim@cpe.ku.ac.th/something
```

URL for database (Django dj-database-url uses this):

```
mysql://myuser:mypassword@hostname/mydatabase
```

Exercises

End of the HTTP basic slides.

Do the exercises described in class,
or see the "*HTTP-in-Action*" slides.

Optional Material

Stuff you aren't required to know.

Do the "HTTP in Action" exercises first.

"GET" in HTML Forms

Two methods of sending data from HTML forms to Web server: GET and POST.

GET puts all form data in the URL.

```
<HTML>
Here is my form:
<FORM ACTION=”/cgi-bin/parse.cgi” METHOD=”GET”>
Your name:<INPUT TYPE=text NAME="Name">
<BR><INPUT TYPE=checkbox NAME="SpamMe"> Want spam?
</FORM>
```

```
GET /cgi-bin/parse.cgi?Name=Jim+Brucker&SpamMe=yes
HTTP/1.1
Host: register.seo.com
Accept: text/html, text/plain, ...
```

"POST" in HTML Forms

POST puts the form data in the *body* of the HTTP request. POST can transfer more data than GET.

```
<HTML>
Here is my form:
<FORM ACTION=”/cgi-bin/parse.cgi” METHOD=”POST”>
Your name:<INPUT TYPE=text NAME="Name">
<BR><INPUT TYPE=checkbox NAME="SpamMe"> Want spam?
</FORM>
```

```
POST /cgi-bin/parse.cgi  HTTP/1.1
Host: register.seo.com

Name=Jim+Brucker
SpamMe=yes
```

Implementing State

- HTTP is *stateless*
- So, how can web server remember (identify) a client?
- How can server remember what page you are on?

How to Implement State

3 common ways:

1. **Hidden fields**

```
<form method="GET">
<input type="hidden" name="id" value="123456789">
```

2. **Path parameters** or custom URL

3. **Cookies**. In HTTP response, server adds header:

`Set-cookie:` some_string_or_random_number

Exercise: View your Cookies

- Look at some cookies in your browser cache.
- What information is included in a cookie?

Firefox: Preferences → Privacy → Remove Individual Cookies

Chrome: Settings → Show Advanced → [Content Settings] button → [All Cookies and Site Data]

Why does Chrome make cookies so hard to find?

Exercise: How many requests per page?

1. Open Chrome More Tools -> Developer Tools
   (also works in Brave, maybe in Edge)
2. Select the Network tab.
3. In a normal Chrome browser window, visit any site.
   Try: cnn.com www.cpe.ku.ac.th

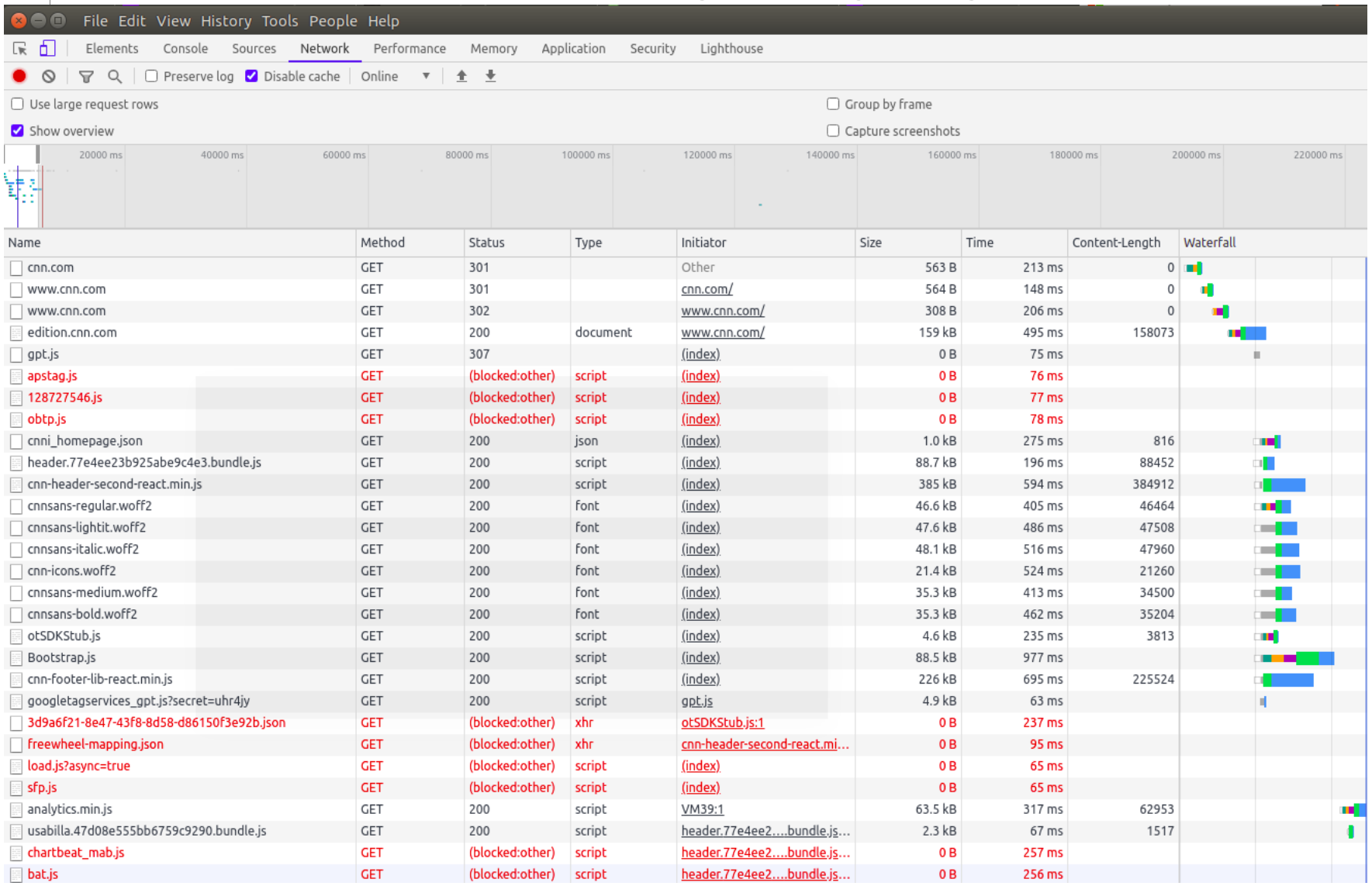
   How many requests did the browser send?

   Why so many?

Note: Look at the *timeline* of requests. Does the browser wait for a reply before sending next request?

Traffic to load page "cnn.com"

178+ requests to get a single page.



Name	Method	Status	Type	Initiator	Size	Time	Content-Length	Waterfall
cnn.com	GET	301		Other	563 B	213 ms	0	
www.cnn.com	GET	301		cnn.com/	564 B	148 ms	0	
www.cnn.com	GET	302		www.cnn.com/	308 B	206 ms	0	
edition.cnn.com	GET	200	document	www.cnn.com/	159 kB	495 ms	158073	
gpt.js	GET	307		(index)	0 B	75 ms		
apstag.js	GET	(blocked:other)	script	(index)	0 B	76 ms		
128727546.js	GET	(blocked:other)	script	(index)	0 B	77 ms		
obtp.js	GET	(blocked:other)	script	(index)	0 B	78 ms		
cnni_homepage.json	GET	200	json	(index)	1.0 kB	275 ms	816	
header.77e4ee23b925abe9c4e3.bundle.js	GET	200	script	(index)	88.7 kB	196 ms	88452	
cnn-header-second-react.min.js	GET	200	script	(index)	385 kB	594 ms	384912	
cnnsans-regular.woff2	GET	200	font	(index)	46.6 kB	405 ms	46464	
cnnsans-lightit.woff2	GET	200	font	(index)	47.6 kB	486 ms	47508	
cnnsans-italic.woff2	GET	200	font	(index)	48.1 kB	516 ms	47960	
cnn-icons.woff2	GET	200	font	(index)	21.4 kB	524 ms	21260	
cnnsans-medium.woff2	GET	200	font	(index)	35.3 kB	413 ms	34500	
cnnsans-bold.woff2	GET	200	font	(index)	35.3 kB	462 ms	35204	
otSDKStub.js	GET	200	script	(index)	4.6 kB	235 ms	3813	
Bootstrap.js	GET	200	script	(index)	88.5 kB	977 ms		
cnn-footer-lib-react.min.js	GET	200	script	(index)	226 kB	695 ms	225524	
googletagservices_gpt.js?secret=uhr4jy	GET	200	script	gpt.js	4.9 kB	63 ms		
3d9a6f21-8e47-43f8-8d58-d86150f3e92b.json	GET	(blocked:other)	xhr	otSDKStub.js:1	0 B	237 ms		
freewheel-mapping.json	GET	(blocked:other)	xhr	cnn-header-second-react.mi…	0 B	95 ms		
load.js?async=true	GET	(blocked:other)	script	(index)	0 B	65 ms		
sfp.js	GET	(blocked:other)	script	(index)	0 B	65 ms		
analytics.min.js	GET	200	script	VM39:1	63.5 kB	317 ms	62953	
usabilla.47d08e555bb6759c9290.bundle.js	GET	200	script	header.77e4ee2….bundle.js…	2.3 kB	67 ms	1517	
chartbeat_mab.js	GET	(blocked:other)	script	header.77e4ee2….bundle.js…	0 B	257 ms		
bat.js	GET	(blocked:other)	script	header.77e4ee2….bundle.js…	0 B	256 ms		

Tools for a Single Request

Sometimes we want to...

- *manually create* & send an HTTP request (for testing)
- control what headers are sent
- *inspect* details of the request and response

Tools for Viewing Http Traffic

HttpFox or HttpRequester (free) – monitor/inspect http requests (Firefox). Great for seeing what is happening.

Chrome "Developer Tools" – use Network tab to watch network traffic.

Dev HTTP Client *aka* "Rest HTTP API Client" (Chrome)

httpwatch – Watches all traffic. Can perform security checks. Chrome & Firefox plugin (free and paid versions) www.httpwatch.com

These tools are great for testing web services.

Command Line HTTP Tools

Sometimes you need to use command line

- curl - command line HTTP client (from Unix)
- netcat (nc) - send TCP or UDP, listen for TCP or UDP
- telnet - primitive way to access any TCP port

Get KU's Home Page

Try curl --verbose or Chrome DHC extension.

1) send a GET request to: https: //www.ku.ac.th

What is the response?

2) send a GET request to the refresh url in the response.

What is the new response?

Where does it tell you to go? What is different?

3) send a GET to the new location.

Keep going...

How would you make KU's web site more efficient?

Get KU's Home Page in *English*

After you successfully get KU's home page,
try adding some request headers (one at a time):

Accept-Language: en
Accept: text/plain
Accept: image/*

Do they work?

What *methods* does this URL allow? Do they work?

Example Web Services

Explore California

http://services.explorecalifornia.org/pox/tours.php

(pox = Plain Old XML, or "rss" or "json")

Google Maps API

http://maps.googleapis.com/maps/api/geocode/xml?address=Kasetsart%20University&sensor=false

curl Examples

- Get a resource (web page, image, anything):

```
curl -v  http://somehost.com/favicon.jpg
```

- Send a POST request with username=hacker

```
curl http://somehost.com/login.jpg
                          --data username=hacker
```

- Specify a header option in request

```
curl -H "Accept: text/plain" http://somehost.com/path
```

- Get help

```
curl --help
```

Many options have 2 forms: -d *or* --data

curl Exercise

Get KU's home page *in English*.

cmd> `curl -H "Accept-language: en" http://www.ku.ac.th/web2012/index.php`

Experiment with methods & headers

- Use netcat to get a web page from iup.eng.ku.ac.th
- Find the actual location of their default home page
- What METHODS does it accept?
  - GET POST PUT HEAD OPTIONS DELETE ?
- Send some invalid requests and note the responses
  - send to invalid URL
  - send unsupported method: DELETE, PUT, POST
  - try to DELETE something!
  - send header that server can't handle, e.g:
    Accept: text/plain or application/xml
    Accept-language: jp

Insecurity

There seems to be a bug in regis.ku.ac.th that allows unauthenticated download of pages, if you know the URL.

The 01219245 (450) class student list is here:

https://regis.ku.ac.th/grade/download_file/class_01219245_611.txt

(You can download it w/o logging in.)

a) download it. (use wget)

b) can you download other course lists?

You have to guess the last 3 digits, but so what?

Computers are good at repetitive tasks.

c) can you upload a new class list (use PUT or POST)?

Exercise

- Find a web page containing a FORM using POST

  <form method="POST" action="*some_url*">

    <input type="text" name="username" .../>

1. examine the page source
2. note the FORM URL and what fields it sends
3. send the form (with data) using Curl or Dev HTTP

```
POST /some/url HTTP/1.1
    Host: www.example.com
    Content-length: 26

name=jim&birthday=1/1/1900
```

Compression

Accept-Encoding: gzip, deflate

Allow server to compress response body.

Q? Can HTTP transmit data in *binary form?*

Surreptitious User Tracking

If you *open an E-mail message*, does the sender know you looked at it?

```
<HTML>
<BODY>
Hello, victim.  So you think just opening e-mail is safe?
Well, think again.  You'll be getting more SPAM from us soon!
<img src=http://www.spammer.com/images/barf.gif?
id=428683927566 />
<!-- this is better, no query params -->
<img src=http://www.spammer.com/images/428683927566.gif? /
>
```

Conditional GET

- A Client can request a resource only if it has been modified since a given date.
- Used for efficiency & caching.
- Use "If-modified-since: " or "Etag:" headers.

```
GET /path/index.html HTTP/1.1
If-modified-since:  1 Aug 18:32:00 ICT 2014
...etc...
```

If page has not been modified, the server responds:

`HTTP/1.1 304 Not Modified`

Conditional GET: server response

- If page **has** been modified, server responds:

```
HTTP/1.1 200 OK
Content-type: blah

DATA
```

- If page has **not** been modified, server responds:

```
HTTP/1.1 304 Not Modified
```

Conditional GET using Etag

- A server can include an "Etag" as page identifier. It is usually an MD5 hash but can be anything:

```
HTTP/1.1 200 OK
Content-Type: image/jpeg
Etag:  "331019636820008"

Image data
```

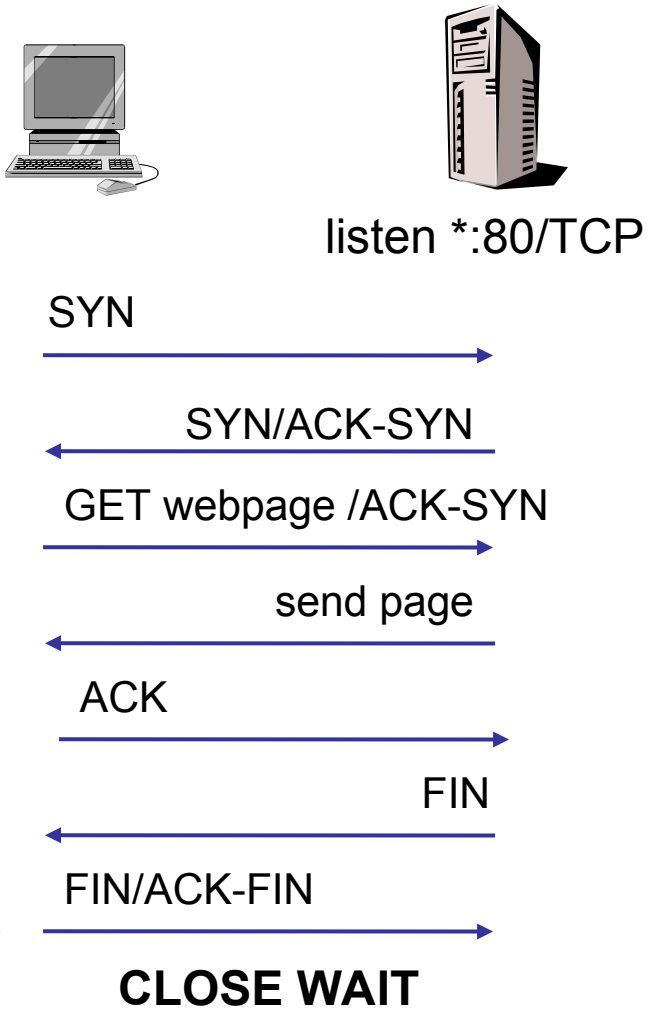
- Next time the client needs the image (but its still in his cache) he sends:

```
GET /path/image.jpeg HTTP/1.1
If-None-Match: "331019636820008"
```

Non-persistent Connection

- In HTTP 1.0, client must open a new connection for **each request**
- Lots of delay
- Lots of traffic and server overhead

Sequence repeated for *every* web request!



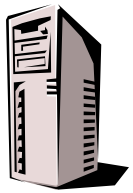
Exercise: How many requests?

- To download and display this web page, how many requests does client have to send to server?
- For HTTP/1.0 how many connections to server are needed?

```
<HTML>
<link rel="stylesheet" href="stylesheet.css">
<BODY>
<h1>My vacation</h1>
<p>
For vacation we went to <a
  href=”http://www.unseen.com/bangkok”>Bangkok</a>.
Here's a photo of <em>Wat Phra Kaeo</em> <br>
<IMG SRC="images/watprakaew.jpeg">
```

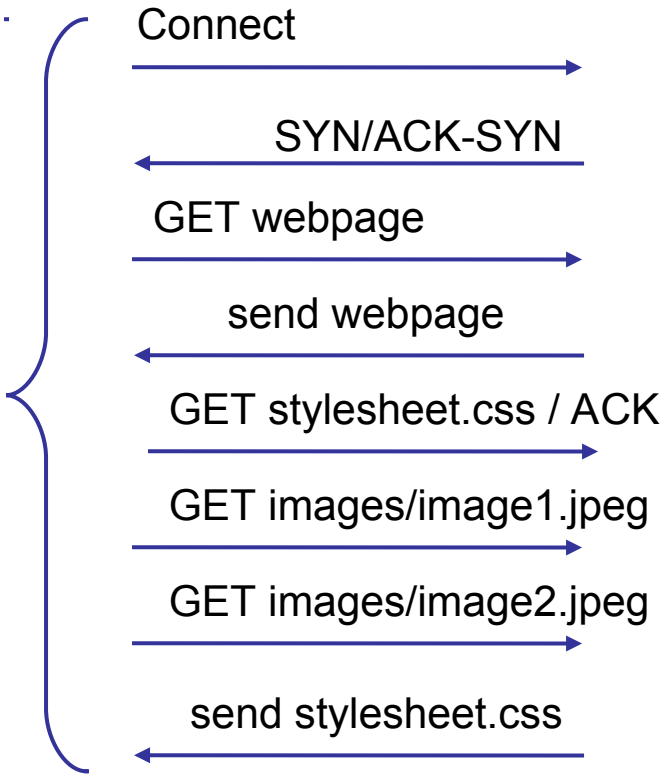
Persistent Connection

- HTTP 1.1 uses persistent connection
- Client can request using keep-alive
- server keeps connection open *briefly*
- client can pipeline requests
- client needs to know length of data



listen *:80/TCP

Multiple request/reply in one connection



Web Caching

- Caching is critical to performance of the web
- Multiple levels of caching:
  - client (web browser cache)
  - server (manually configured cache)
  - gateway (transparent cache engine)
  - network (CDN, cooperating caches)

Cache Engines

- Harvest (free)
- Squid (free)
- Cisco Cache Engine (based on Linux and Harvest)

Why Web Caching?

- Decrease use of network bandwidth
- Faster response time
- Decrease server load
- Security and web access controls (auth, blocking)

Content Delivery Networks

- Akamai, DigitalIsland, etc.
- Has its own network of servers that replicates content of the content provider (e.g. cnn.com), e.g. all images
  - in the index.html file all references of:

    www.cnn.com/images/sports.gif
  - is re-mapped to
    www.akamai.com/www.cnn.com/images/sports.gif
- Akamai servers cache images and index files for cnn.com
- Server domain name: www.akamai.com
- Index file changed to: www.akamai.com/.../images/sports.gif

Content Delivery Networks (2)

- When client downloads http://www.cnn.com/index.html he gets a cached (modified) file from cache server, containing

<img src=http://www.akamai.com/www.cnn.com/images/sports.gif>

- Next, client tries to resolve "www.akamai.com"
- DNS server of Akamai will...
  - identify client's location based on client's IP address (database)
  - chooses one of Akamai's cache servers which is "closest" to the client's location
  - returns IP address for "www.akamai.com" closest to client.