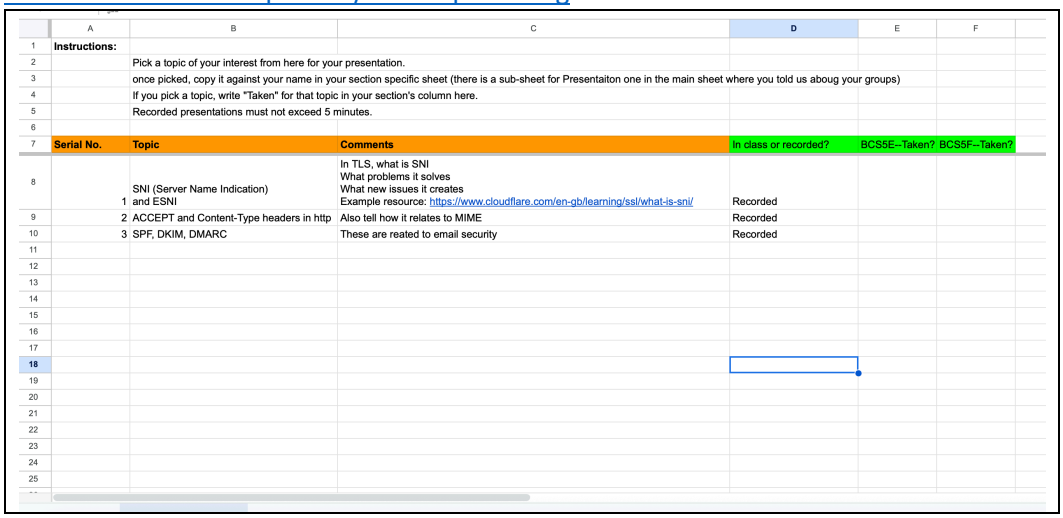
Class Notes

Announcements:

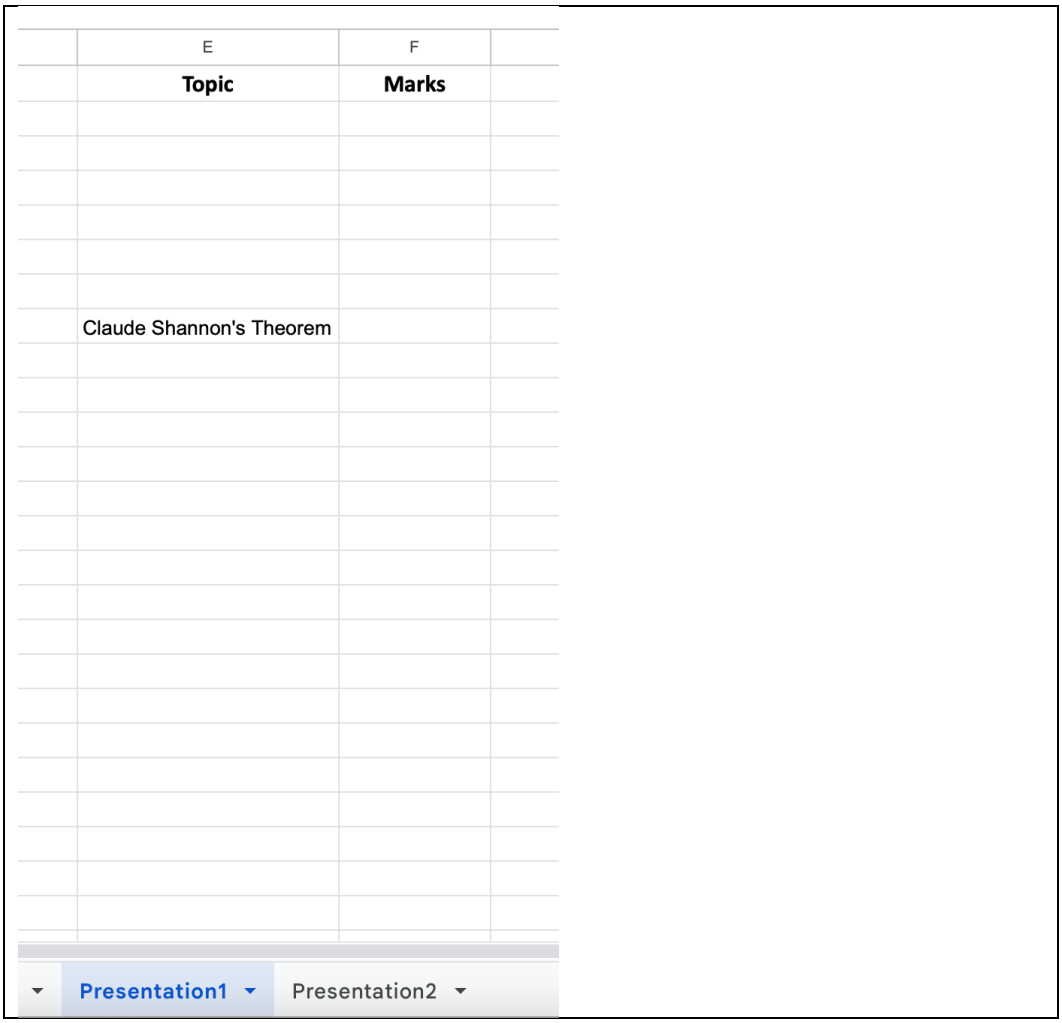
(1) **Sessional-1 exam** coming.
   a. First two chapters are the syllabus. (See the exclusions of few sub-sections that I mentioned earlier.)
   b. Focus on concepts + numerical
   c. Textbook and its associated website have tons of practice questions
   d. Do not forget to bring your calculators to the exams

(2) **Presentation 1**: I am putting topics in a sheet. Pick a topic for yourself from there to write it to your main sheet.
   a. Topics sheet: https://docs.google.com/spreadsheets/d/1HvesJOL-F4m1jLD8sj-AaTm13EVX512a8ZcNpJslD7k/edit?usp=sharing



	A	B	C	D	E	F
1	**Instructions:**					
2		Pick a topic of your interest from here for your presentation.				
3		once picked, copy it against your name in your section specific sheet (there is a sub-sheet for Presentaiton one in the main sheet where you told us aboug your groups)				
4		If you pick a topic, write "Taken" for that topic in your section's column here.				
5		Recorded presentations must not exceed 5 minutes.				
6						
7	**Serial No.**	**Topic**	**Comments**	**In class or recorded?**	**BCS5E--Taken?**	**BCS5F--Taken?**
8	1	SNI (Server Name Indication) and ESNI	In TLS, what is SNI What problems it solves What new issues it creates Example resource: https://www.cloudflare.com/en-gb/learning/ssl/what-is-sni/	Recorded		
9	2	ACCEPT and Content-Type headers in http	Also tell how it relates to MIME	Recorded		
10	3	SPF, DKIM, DMARC	These are reated to email security	Recorded		
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						

   b. Write the topic in the Presentation1 sub-sheet of your section (where you told us about your project groups and have your late-day balance)



Lesson plan for today:

(1) Wrap-up DNS discussion
   a. Resource record details
   b. DNS packet structure
   c. DNS as a "crude" load-balancer
   d. PTR queries in DNS
(2) HTTP discussion
(3) SMTP discussion
(4) If time permits, we will discuss TCP sockets (we discussed UDP sockets earlier)

Point to ponder:

Q1: How a DNS resolver gets the IPs of the root DNS service to kickstart the DNS resolution process?

Answer: Ask someone! “named.root” file maintained by IANA at
https://www.internic.net/domain/named.root

Looks something like the following:

```
;       This file holds the information on root name servers needed to
;       initialize cache of Internet domain name servers
;       (e.g. reference this file in the "cache  .  <file>"
;       configuration file of BIND domain name servers).
;
;       This file is made available by InterNIC
;       under anonymous FTP as
;           file                /domain/named.cache
;           on server           FTP.INTERNIC.NET
;       -OR-                    RS.INTERNIC.NET
;
;       last update:     September 04, 2025
;       related version of root zone:     2025090401
;
; FORMERLY NS.INTERNIC.NET
;
.                        3600000      NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.      3600000      A     198.41.0.4
A.ROOT-SERVERS.NET.      3600000      AAAA  2001:503:ba3e::2:30
;
; FORMERLY NS1.ISI.EDU
;
.                        3600000      NS    B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.      3600000      A     170.247.170.2
B.ROOT-SERVERS.NET.      3600000      AAAA  2801:1b8:10::b
;
; FORMERLY C.PSI.NET
;
.                        3600000      NS    C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET.      3600000      A     192.33.4.12
C.ROOT-SERVERS.NET.      3600000      AAAA  2001:500:2::c
;
; FORMERLY TERP.UMD.EDU
;
.                        3600000      NS    D.ROOT-SERVERS.NET.
D.ROOT-SERVERS.NET.      3600000      A     199.7.91.13
D.ROOT-SERVERS.NET.      3600000      AAAA  2001:500:2d::d
;
; FORMERLY NS.NASA.GOV
;
.                        3600000      NS    E.ROOT-SERVERS.NET.
E.ROOT-SERVERS.NET.      3600000      A     192.203.230.10
E.ROOT-SERVERS.NET.      3600000      AAAA  2001:500:a8::e
;
; FORMERLY NS.ISC.ORG
;
.                        3600000      NS    F.ROOT-SERVERS.NET.
F.ROOT-SERVERS.NET.      3600000      A     192.5.5.241
F.ROOT-SERVERS.NET.      3600000      AAAA  2001:500:2f::f
:
```

Point to ponder:

Q2: If you are developing a new DNS client or server, from where you will get authoritative information about the specifications (requirements)?

Other auxiliary services of DNS:

(1) Host aliasing.
   a. Canonical name could be: relay1.west-coast.enterprise.com
   b. Alias could be: www.enterpirse.com
   c. DNS can give the canonical name and IP of the host given the alias.

> Think of canonical names as "official names" which are usually meant for back-end services.
> Example for EC2 in AWS: `ec2-54-82-1-32.compute-1.amazonaws.com`
>
> You can think of alias as nice short names for something longer.
> Ex: Picaso's full name is Pablo Diego José Francisco de Paula Juan Nepomuceno María de los Remedios Cipriano de la Santísima Trinidad Ruiz y Picasso

(2) Mail server aliasing.
   a. Canonical name of the mail server could be relay1.west-coast.yahoo.com
   b. Alias could be yahoo.com
   c. Using a special DNS record type (called MX record), both host names and mail server name could be the same (aliases).

(3) Load distribution
   a. Large sites (such as cnn.com or google.com) have excessive user load (for example billions of search queries coming to the web server at Google.) Such sites therefore have many replicated services on many servers, each server having a different IP address. DNS might rotate available IPs for a name for different queries, say in round-robin fashion.
   b. At times DNS server return multiple IP addresses to the client. Other times, server picks one IP of its choice and provides that one. DNS might provide an IP address that is "near" to the client. What constitutes "near" varies. It could be geographical distance or network distance (say in terms of number of hops from the client to the service.)

DNS record types:

Type	Meaning	Value
SOA	Start of authority	Parameters for this zone
A	IPv4 address of a host	32-Bit integer
AAAA	IPv6 address of a host	128-Bit integer
MX	Mail exchange	Priority, domain willing to accept email
NS	Name server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
SPF	Sender policy framework	Text encoding of mail sending policy
SRV	Service	Host that provides it
TXT	Text	Descriptive ASCII text

Figure 7-4. The principal DNS resource record types.

Source: Computer Networks by Tanenbaum et al.

DNS packet format:

1. **Header**: Contains metadata about the DNS query or response, including flags, operation codes, and counts of various sections.

2. **Question**: Specifies the domain name being queried, along with the type of query (e.g., A, MX) and the class (usually IN for internet).

3. **Answer**: Holds the resource records (RRs) that directly answer the question, containing the resolved data like IP addresses or mail server names.

4. **Authority**: Lists the resource records pointing to authoritative name servers for the domain, used to help resolve the query further if needed.

5. **Additional**: Provides extra information that may be useful for resolving the query, such as additional RRs not directly answering the question but related to the authority section.

And further details in the header:

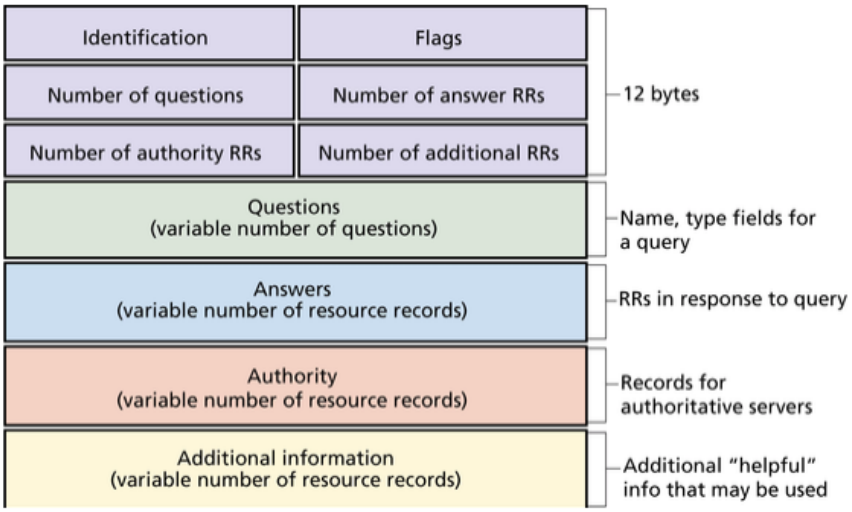
DNS Header

Offset	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	*Transaction ID*																*QR*	*OPCODE*				*AA*	*TC*	*RD*	*RA*	*Z*	*AD*	*CD*	*RCODE*			
4	32	*Number of Questions*																*Number of Answers*															
8	64	*Number of Authority RRs*																*Number of additional RRs*															

Source: Wikipedia

Field	Description
ID	A unique identifier for matching responses with queries.
QR (Query/Response)	Indicates if the packet is a query (0) or a response (1).
Opcode	Specifies the type of query (e.g., standard, inverse, or server status).
AA (Authoritative Answer)	Indicates if the response is from an authoritative name server.
TC (Truncation)	Indicates if the message was truncated due to size limits.
RD (Recursion Desired)	Requests that the server perform recursion to resolve the query.
RA (Recursion Available)	Indicates if the server supports recursion in the response.
Z	Reserved for future use, must be set to 0.
AD (Authenticated Data)	Indicates if the data has been authenticated by the server (DNSSEC).
CD (Checking Disabled)	Instructs the server to disable DNSSEC validation.
RCODE (Response Code)	Indicates the status of the response (e.g., no error, format error).
QDCOUNT (Question Count)	Number of entries in the Question section.
ANCOUNT (Answer Count)	Number of resource records in the Answer section.
NSCOUNT (Authority Count)	Number of resource records in the Authority section.
ARCOUNT (Additional Count)	Number of resource records in the Additional section.

Figure 2.19: DNS message format



Source: Computer Networks: A top down approach, 9th edition

Security aspects:

Any compromise to the DNS resolution system can badly impact security of the user. Example: If yourbank.com could be mapped to a malicious server, that impersonate your bank's usual look and feel, malicious actors might be able to deceive to give up your login and password. Which then they can use on the real bank website to transfer funds!

The DNS root server H-root is managed by U.S. Army Research Lab (ARL). Turns out naming can help us catch many illicit activities on the Internet.

DNS PTR records can help detecting malicious activity such as scanning and spams:

Detecting Malicious Activity with DNS Backscatter Over Time

Kensuke Fukuda John Heidemann Abdul Qadeer

***Abstract*—Network-wide activity is when one computer (the *originator*) touches many others (the *targets*). Motives for activity may be benign (mailing lists, CDNs, and research scanning), malicious (spammers and scanners for security vulnerabilities), or perhaps indeterminate (ad trackers). Knowledge of malicious activity may help anticipate attacks, and understanding benign activity may set a baseline or characterize growth. This paper identifies *DNS backscatter* as a new source of information about network-wide activity. Backscatter is the reverse DNS queries caused when targets or middleboxes automatically look up the domain name of the originator. Queries are visible to the authoritative DNS servers that handle reverse DNS. While the fraction of backscatter they see depends on the server's location in the DNS hierarchy, we show that activity that touches many targets appear even in sampled observations. We use information about the queriers to classify originator activity using machine-learning. Our algorithm has reasonable accuracy and precision (70–80%) as shown by data from three different organizations operating DNS servers at the root or country-level. Using this technique we examine nine months of activity from one authority to identify trends in scanning, identifying bursts corresponding to Heartbleed and broad and continuous scanning of ssh.**

does not generalize to network-wide activity. Darknets [38], [35], [56], [13], [14], [17] and honeypots (for example, [42]) are effective at understanding network-wide activity, but they miss targeted scans (scanning only Alexa top websites [17]), and new large darknets are unlikely given IPv6 full allocation and the huge IPv6 space. Search engines gather information about activity that appears in the public web, but information is unstructured and may be delayed by indexing [50]. (§ VII has detailed related work.)

This paper identifies a new source of information on network-wide activity: *DNS backscatter*, the reverse DNS queries triggered by such activity (see Figure 1 and § II). Activities of interest are those that touch many Internet devices, including malicious or potentially malicious activity such as spamming and scanning, as well as widespread services such as CDNs, software updates, and web crawling. These activities trigger *reverse DNS queries* as firewalls, middleboxes, and servers (*queriers*) resolve mapping of the IP address of the

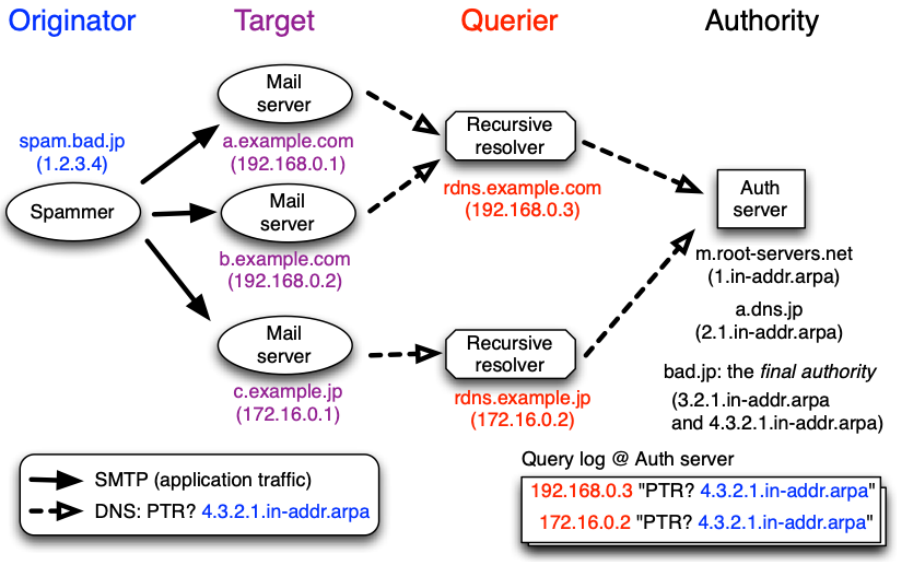
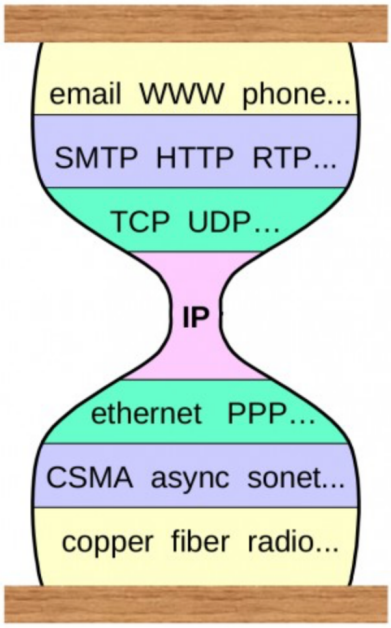


Fig. 1: The process behind a DNS backscatter sensor: an originator sends mail to targets, causing queries that are observed at the authority.

https://ant.isi.edu/~johnh/PAPERS/Fukuda17a.pdf

HTTP – New Narrow Waist of the Internet?



- Source: 2001 Presentation by Steve Deering (PDF)

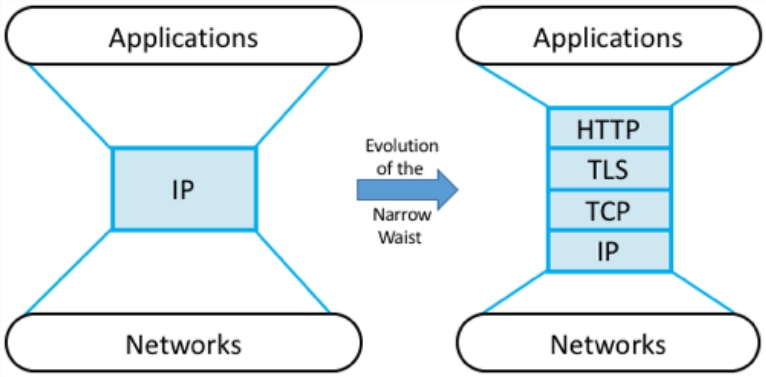


Figure 151. *HTTP (plus TLS, TCP, and IP) forming the narrow waist of today's Internet architecture.*

Source: https://book.systemsapproach.org/e2e/trend.html

What apps use HTTP?

🌍 Everyday Apps That Use HTTP

- **Web browsers** (Chrome, Safari, Firefox, Edge, etc.) → load websites using HTTP/1.1, HTTP/2, or HTTP/3.
- **Mobile apps** (Facebook, Twitter/X, Instagram, TikTok, WhatsApp, etc.) → use HTTP(S) under the hood for API calls to their servers.
- **Desktop apps** (Slack, Zoom, Spotify, VS Code, etc.) → communicate with their backends using HTTP(S) APIs.

⚙️ System & Developer Tools

- **Package managers** (npm, pip, apt, yum, cargo, etc.) → download packages over HTTP(S).
- **Software updates** (Windows Update, macOS Software Update, app stores) → fetch updates via HTTP(S).
- **Command-line tools** (`curl`, `wget`, `httpie`) → directly issue HTTP requests.
- **APIs and SDKs** → almost all modern APIs (REST, GraphQL, gRPC-Web) use HTTP as their transport.

📺 Media & Streaming

- **YouTube, Netflix, Spotify, Twitch, etc.** → deliver video/audio over **HTTP-based streaming protocols** like HLS (HTTP Live Streaming) and DASH.
- Even "live" streams are typically just **segments served over HTTP**.

☁️ Cloud & Services

- **Cloud storage** (Google Drive, Dropbox, OneDrive, S3) → upload/download files over HTTP(S).
- **IoT devices** (smart bulbs, cameras, thermostats) → call cloud APIs over HTTP(S).
- **Web services/microservices** → usually expose HTTP(S) REST APIs.

📩 Messaging & Communication

- **Email web clients** (Gmail, Outlook Web, Yahoo Mail) → use HTTP(S).
- **Slack, Discord, Microsoft Teams** → rely heavily on HTTP APIs + WebSockets (which *upgrade* from HTTP).

HTTP Details

Problem from last lecture:

HTTP uses TCP from the layer below (the transport layer). Each new TCP connection to a peer takes 2 round trips time (RTT). Initiating peer can piggyback HTTP request on the third message of the three-way TCP handshake. Still, it takes two RTTs before the requested object start coming at the client. A typical webpage can have tens of URLs to different objects (such as logos, pictures, videos etc.). Incurring 2 RTTs per object will add substantial user-visible latency in rendering the page in the browser.

Question: How to solve the above latency issue?

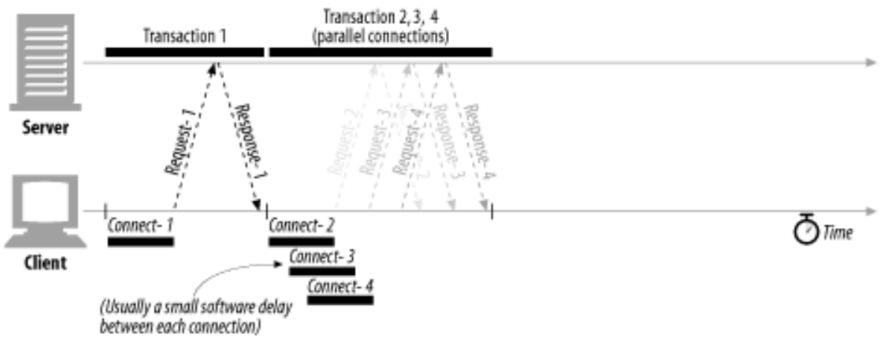
A little digression: Remind from the last lecture that URL are of the form:

http://www.someSchool.edu/someDepartment/home.index

Will the browser need to use DNS for the same host name for other URLs on the same webpage?

Solution 1: Use multiple TCP connections between the client (browser) and the HTTP server (Serial to parallel):

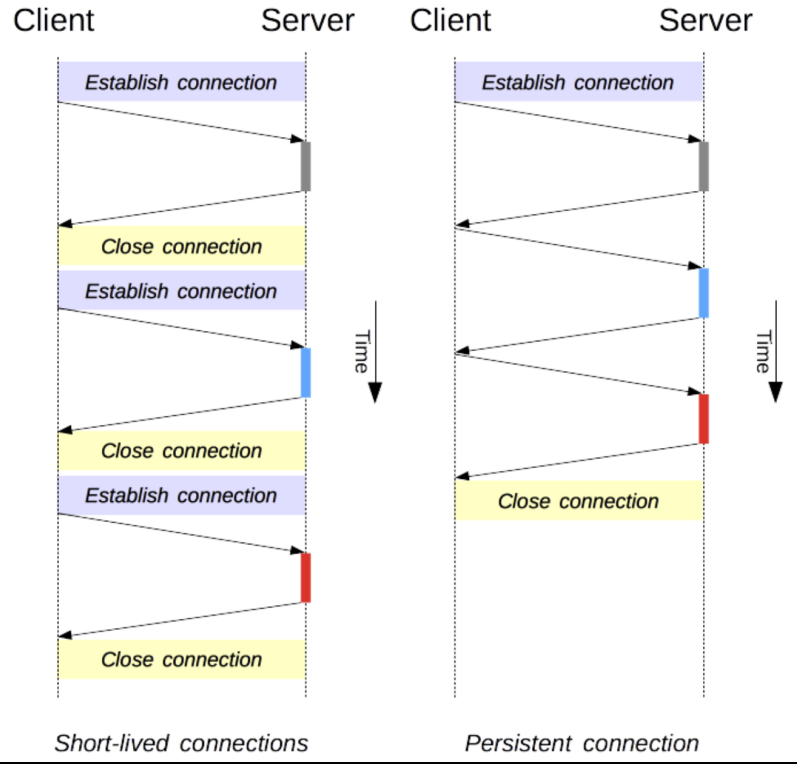
The connection delays are overlapped in time, so it might feel as if page is loading faster.



Source: https://learning.oreilly.com/library/view/http-the-definitive/1565925092/ch04s04.html

How many parallel connections to use depends on the browser. Usually, 4 to 6 is a typical number. Though by using multiple TCP connections, this specific user might be using “unfair” amount of available bandwidth.

Solution 2: Request HTTP server to use persistent TCP connection:



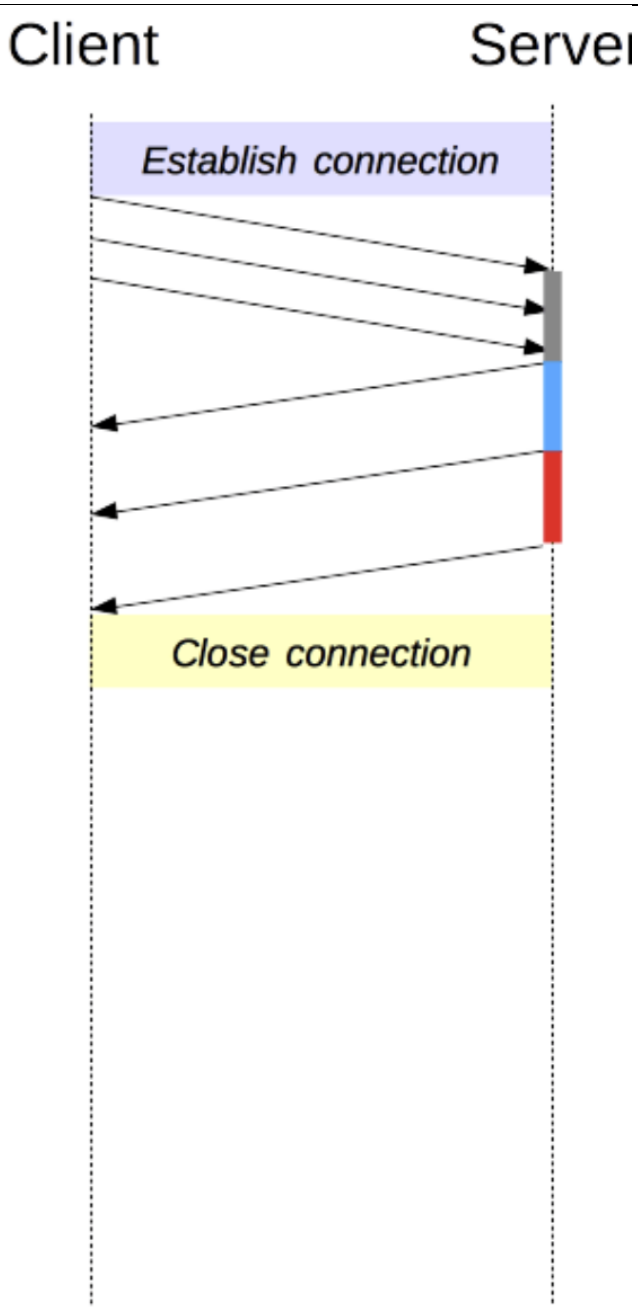
Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Connection_management_in_HTTP_1.x

It depends on the server for how long it keeps a TCP connection open with a specific client. It is a configurable time in the server. A server might drop idle persistent connection in favor of serving new incoming requests.

Persistent connection is an improvement. But we still incur an RTT to request the next object.

Solution 3: Pipeline the requests and responses

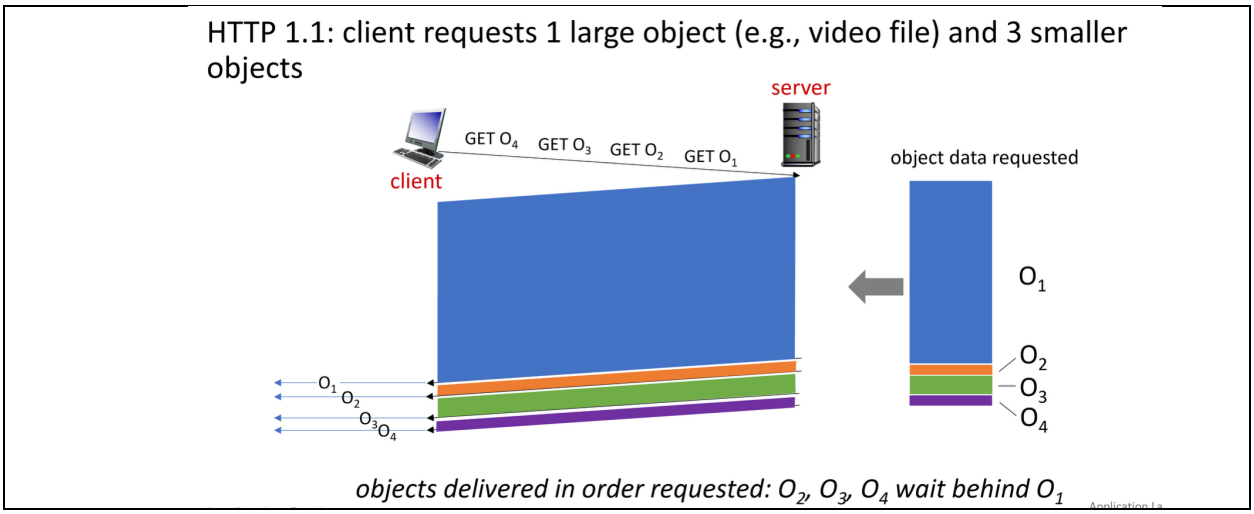
Client sends requests for all the objects back-to-back. The response for those requests come in order from the server.



HTTP Pipelining

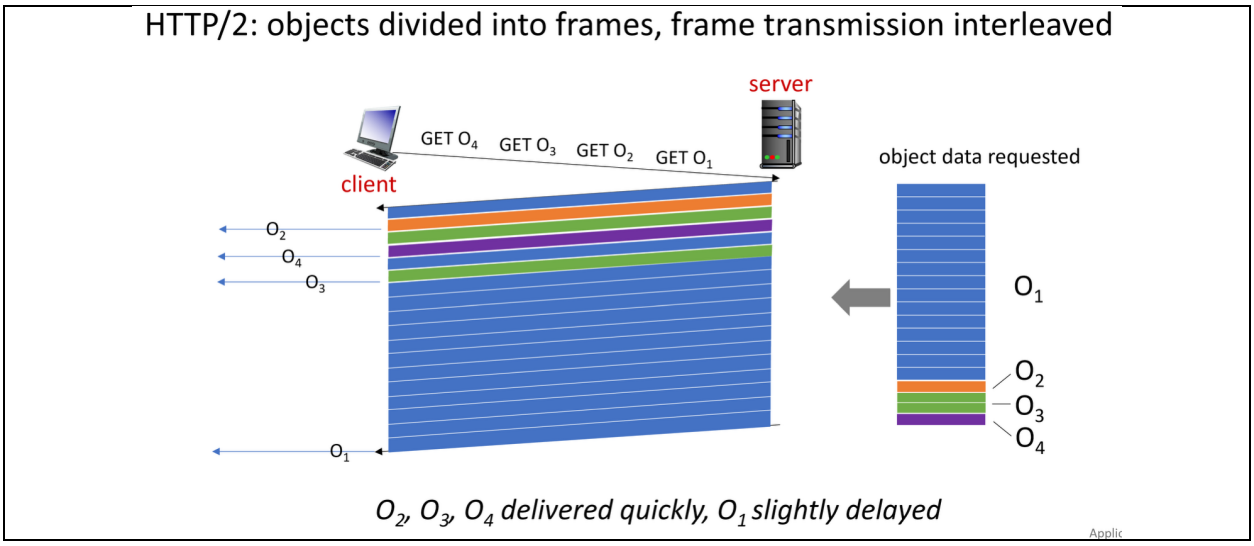
Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Connection_management_in_HTTP_1.x

New problem: Potential head-of-line blocking:



Solution: If client can guess the size of the objects, client might put the request in a "good" order. For example shorter sized objects first. But it is not always possible for the client to correctly guess object sizes.

Solution 4: Multiplex "frames" of requested objects in the response + let the client set the priority of the objects

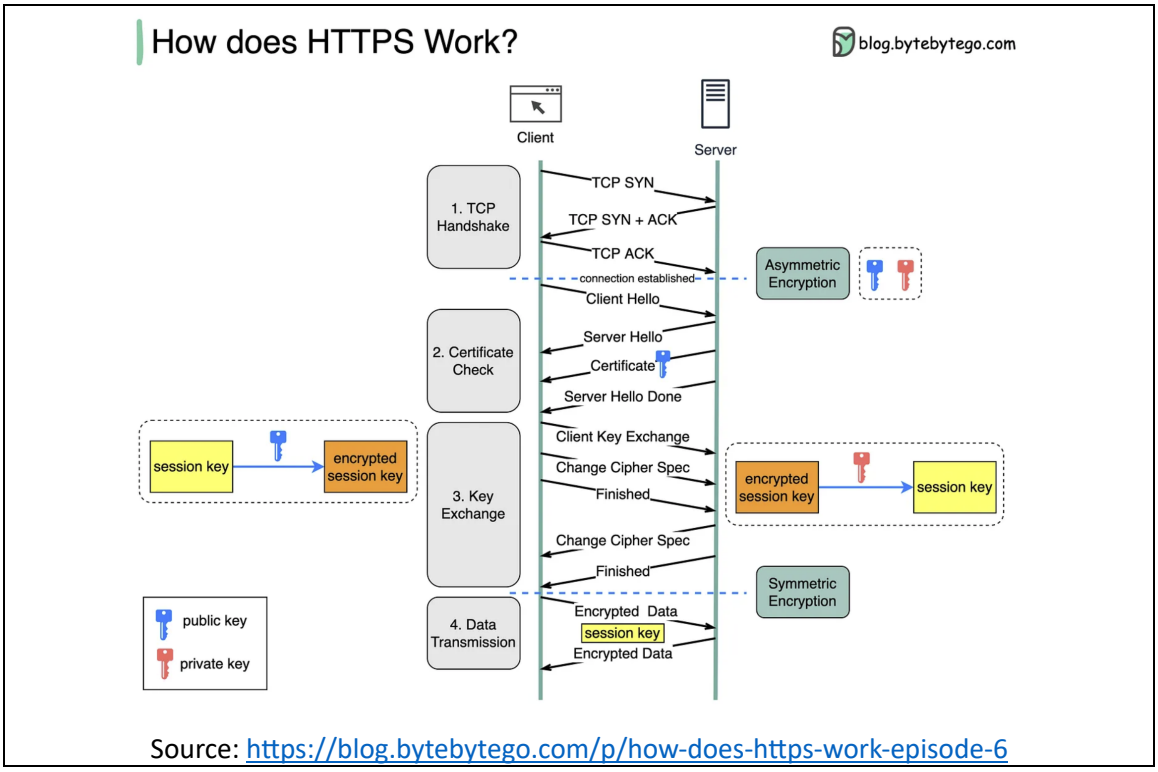


Further tweaks:

(a) Server might send sub-objects in a page to the client without client explicitly sending in the request. Server assumes that, client will request them anyway. Doing so, we can save one RTT when client will parse the main page and send in requests for the sub-objects.

(b) HTTP 2.0 shifted from ASCII text to Binary format. There are many reasons. shifting to a **binary format** allowed HTTP/2 to achieve much higher performance, improved reliability, and better support for modern web requirements, such as multiplexing and efficient resource usage.

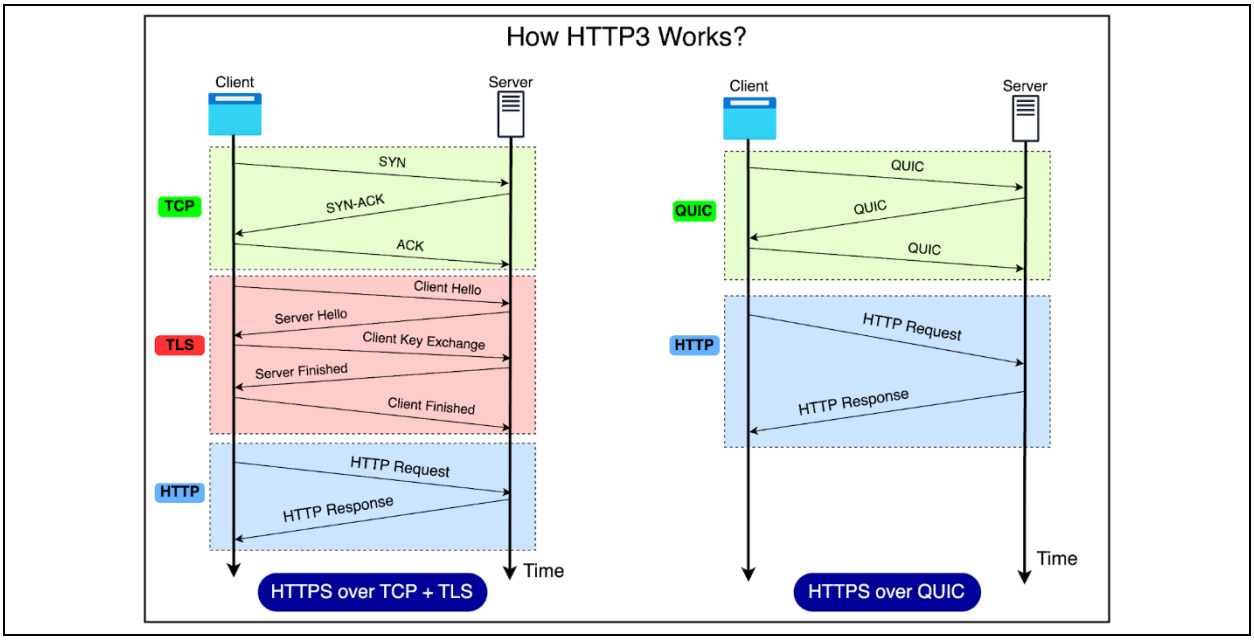
(c) Traditional TCP sockets did not provide any security (confidentiality, integrity, peer authentication). Transport Layer Security (TLS) incorporated security using application-level libraries.



Source: https://blog.bytebytego.com/p/how-does-https-work-episode-6

Solution 5: Shift HTTP from TCP to UDP

Quick UDP Internet Connection (QUIC) does the necessary handshakes in fewer RTTs. This protocol was invented by Google (named Speedy then) and now being standardized as QUIC.



Source: https://blog.bytebytego.com/p/http1-vs-http2-vs-http3-a-deep-dive

Which HTTP version introduced what?:

Please read the book!

We expect you to know which HTTP version introduced which feature.

Feature	HTTP/0.9	HTTP/1.0	HTTP/1.1	HTTP/2.0	HTTP/3.0
Year Introduced	1991	1996	1997	2015	2020
Primary Use Case	Simple, text-based requests	Full-featured protocol	Persistent connections	Multiplexing, performance	QUIC-based, faster connections
Request Methods	Only GET	GET, POST, HEAD	GET, POST, HEAD, PUT, DELETE	Same as 1.1	Same as 1.1
Response Format	Only HTML	HTML, images, files	Supports multiple content types	Binary framing	Binary framing over QUIC
Status Codes	No status codes	Basic status codes (200, 404, etc.)	Full set of status codes	Full set of status codes	Full set of status codes
Persistent Connections	No	No	Yes (default)	Yes	Yes
Connection Handling	Single connection per request	Single connection per request	Keep-Alive (multiple requests per connection)	Multiplexing (multiple streams over a single connection)	Multiplexing over QUIC
Compression	None	None	Optional (Gzip)	Header compression (HPACK)	Header compression (QPACK)
Multiplexing	No	No	No	Yes	Yes
Protocol Layer	Application Layer (TCP-based)	Application Layer (TCP-based)	Application Layer (TCP-based)	Application Layer (TCP-based)	Application Layer over QUIC (UDP-based)
Security (Encryption)	No	Optional (via SSL/TLS)	Optional (via SSL/TLS)	Mandatory TLS	Mandatory TLS 1.3
Header Structure	No headers	Plain text, no compression	Plain text, no compression	Binary, compressed (HPACK)	Binary, compressed (QPACK)
Server Push	No	No	No	Yes	Yes
Transport Protocol	TCP	TCP	TCP	TCP	QUIC (UDP-based)

Next Problem: Repeatedly fetching objects that are not changed at the server

Every time we fetch a webpage, few objects do not change. For example, the logo of FAST when we visit www.nu.edu.pk and so on. So why waste bandwidth fetching such objects repeatedly?

Solution: Conditional GET

Request:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

Response:

```
HTTP/1.1 200 OK
Date: Sat, 3 Oct 2015 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 9 Sep 2015 09:23:24
Content-Type: image/gif

(data data data data data ...)
```

Conditional GET:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 9 Sep 2015 09:23:24
```

Response:

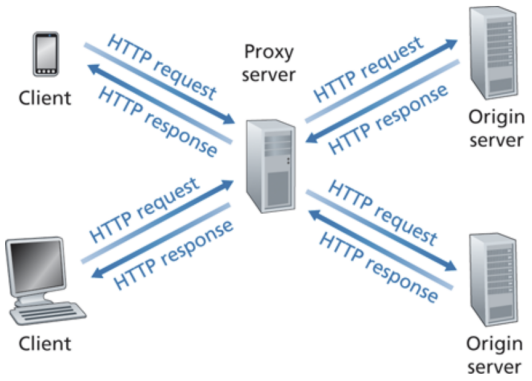
```
HTTP/1.1 304 Not Modified
Date: Sat, 10 Oct 2015 15:39:29
Server: Apache/1.3.0 (Unix)

(empty entity body)
```

But what if multiple parties are asking for the same object again and again?

In an organization like FAST, what if many students are fetching www.nu.eud.pk multiple times in a day. FAST's logo will still be fetched once and cached locally in the browser cache. But can we do better? Can we have a cache for the organization?

Web Caching vi a proxy server (at times called forward proxy)

Figure 2.11 **Clients requesting objects through a Web cache**



Let's switch to second slide deck (already posted on GCR), at slide no 42.

……