Computer Networks: Midterm Summary

Author: Shun (@shun4midx)

Chapter 1 Summary

Basic Definitions

Overview

The internet is a "network of networks" **(Interconnected ISPs, i.e. Internet Service Providers)**, where a **network** is a collection of devices, routers, and links *managed by an organization*

- **Hosts** = End systems
- **Packet Switches** = Forward packets (chunks of data), e.g. *routers, switches*
- **Communication Links**: E.g. *fiber, copper, radio, satellite*; **transmission rate** = *bandwidth* in bps
- **Internet Services**: **Infrastructure** to provide services to apps (e.g. web, email, streaming, etc), **programming interface** such as *hooks* to "connect" and *service*

Protocol

Protocols define the **format, order** of **messages sent and received** among network entities, and **actions taken** on message transmission, receipt.

Network Edge

Access Networks

Access networks are how end systems connect to **edge routers**.

- **HFC** (Hybrid Fiber Coax): A network of *fiber* (to the **neighborhood node**) and *shared coax* (to homes) attaches homes to the ISP router at the **cable headend**; Higher downstream transmission compared to upstream
  - Uses **cable-based access** via **FDM** (frequency division multiplexing), i.e. different channels are transmitted in different frequency bands
- **DSL** (Digital Subscriber Line): Uses existing *telephone line* to central office **DSLAM** (Data connects to internet, voice connects to telephone net); Higher downstream transmission compared to upstream

There are two main kinds of **wireless access networks** that connect end systems to router:

- **WLANs** (Wireless Local Area Networks): Typically within or around building, e.g. *Wi-Fi*. Can have low or high transmission rates.
- **Wide-Area Cellular Access Networks**: Provided by *mobile, cellular network operator*, typically medium transmission rate (around **tens of Mbps**), e.g. *4G/5G cellular networks*

Examples of Access Networks

- **Home Access:** DSL, HFC/Cable
- **Wide-area Wireless:** 4G LTE, 5G NR.

Local Networks (LANs, Inside Premises)

- **Home LAN:** Wi-Fi + Ethernet switch + NAT/firewall (often one gateway).
- **Enterprise LAN:** Switched Ethernet access + Managed Wi-Fi.

Physical Media Links

Physical Media Links

A **physical link** is what lies between transmitter and receiver

- **Guided**: Signals propagate in **solid media**, e.g. *twisted pair/TP* (two insulated copper wires), *coaxial cable* (bidirectional, two concentric copper conductors), *fiber optic cable* (high speed operation, low error).
- **Unguided**: Signals propagate **freely**, e.g. **radio**, such as *Wi-Fi, wide-area, bluetooth, terrestrial microwave, satellite*

Network Core

Packet Switching and Circuit Switching

There are two key network-core functions:

- **Forwarding** (i.e. *switching*): *Local*, moves packets from router's input link to output link
- **Routing**: *Global* action, determines best paths taken by packets

Packet vs. Circuit Switching

Packet switching: Messages $\rightarrow$ packets; each hop does *store-and-forward*. It **queues** your desired packets into a buffer whenever **arrival rate $>$ transmission rate**.

- **Pros**: Efficient sharing, good for bursty traffic, no call setup.
- **Cons**: Queueing delay, possible loss under congestion.

Circuit switching: End-to-end resources reserved for **call between source and destination**, i.e. you posses the entire line at a time.

- **Pros**: Predictable performance.
- **Cons**: Idle when unused, setup overhead.

Example Calculation

Problem: Say we have a 1 Gb/s output link, and each user uses 100 Mb/s when active. Each user is active around 10% of the time. How many users can use this network under circuit-switching and packet-switching?

Circuit-Switching: Each user possesses the *entire line at a time*, so $\frac{\text{1Gb/s}}{\text{100Mb/s}} = \boxed{\text{10 users}}$

Packet-Switching: Say there are 35 users, then by binomial distribution, the probability that **more than 10** (maximum amount which we can share) are active at the same time is less than 0.0004. This means, we can hold 35 users with only 0.0004 of the data not being able to be sent.

Internet Structure

Intuitively, the reasoning is we require all hosts to be able to interconnect with each other, with as few connections as possible. As we cannot have one big global ISP (**Internet Service Provider**) in practice, we end up creating **several ISPs**, including *regional ISPs* and other **"tier-1" commercial ISPs**, e.g. *AT&T, NTT*, which regional ISPs connect to. Then, we interconnect them with **IXPs (Internet Exchange Points)**. Otherwise, a **content provider network** (e.g. Google), which is a private network, connects its data center to the Internet, often **bypassing most ISPs**. Hence why we say the internet is a *"network of networks"*.

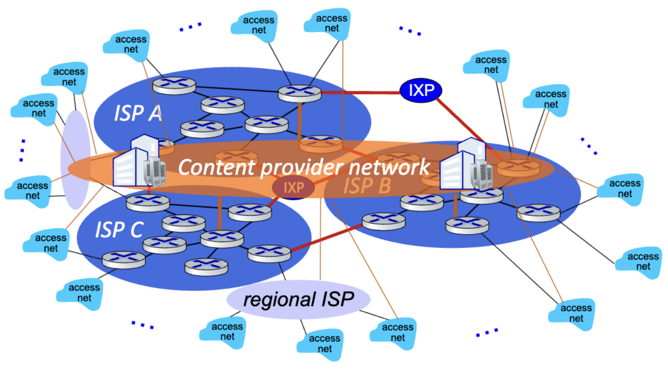


Figure 1: A visual for internet structure.

Performance Loss: Delay and Throughput

Packet Delay

There are four main types of packet delay, as better shown in the figure below.

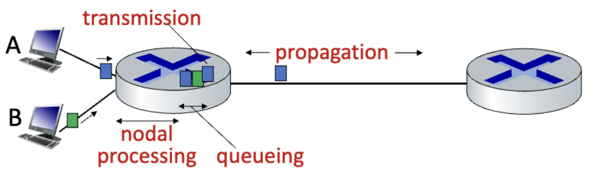


Figure 2: Depiction of the four main sources of packet delay

Packet Delay Formula

In general, we have:

$$\boxed{d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}}$$

- **Nodal Processing *(Const)*:** When checking bit errors and determining *output link*; typically within *microsecs*
- **Queueing Delay:** Time waiting at output link for transmission, depends on the congestion level of the router
- **Transmission Delay:** $d_{\text{trans}} = \frac{L}{R}$, L = *packet length (bits)*, R = link *transmission rate (b/s)*
- **Propagation Delay *(Const)*:** $d_{\text{prop}} = \frac{d}{s}$, d = *length of physical link*, s = *propagation speed* ($\approx 2 \times 10^8$ m/s)

Traffic Intensity (Packet Queueing Delay)

Define a as the average packet *arrival rate*, L as the packet length in bits, and R as the link bandwidth (bit transmission rate). We define the **traffic intensity** as follows:

$$\frac{L \cdot a}{R} = \frac{\text{arrival rate of bits}}{\text{service rate of bits}}$$

Usually, traffic intensity could tell us certain info as follows:

- $\frac{La}{R} \approx 0$: average *queueing delay is small*
- $\frac{La}{R} \to 1$: average *queueing delay is large*
- $\frac{La}{R} > 1$: work arriving $>$ servicable work $\Rightarrow$ infinite delay!

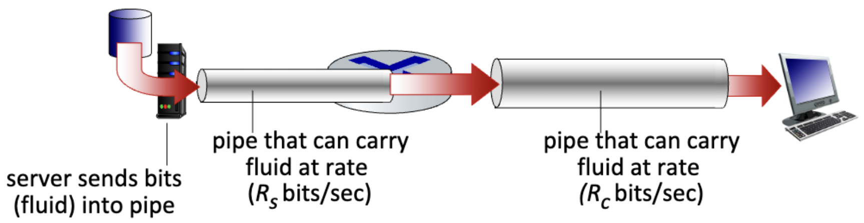
In real life, we can use `traceroute <website>` in the terminal to measure internet delay. It will output the time elapsed for each (publically accessible) step required, to reach said website.

Throughput

Throughput Definition

Throughput is defined as the **rate** (bits/time) at which bits are being **sent** from sender to receiver.

In practice, R_c or R_s, as demonstrated below, is the bottleneck. In extreme cases, R/(number of shared connections) may be the bottleneck. **The bottleneck is the minimum rate R**.


server sends bits (fluid) into pipe
pipe that can carry fluid at rate (R_s bits/sec)
pipe that can carry fluid at rate (R_c bits/sec)

Network Security

The main takeaway is the Internet was designed **without considering adverseries**, so it's **prone to attacks**.

Types of Attacks

- **Denial of Service (DoS):** Overwhelm resources to block service
- **Packet Sniffing:** Reads all packets (including passwords) in a network
- **Spoofing:** Injection of packet with false sourcce address

Countermeasures

- **Cryptography:** Encryption, authentication
- **Integrity Checks:** Signatures to prevent and detect tampering
- **Firewalls** to avoid DoS attacks and filter incoming packets

Protocol Layers and Reference Models

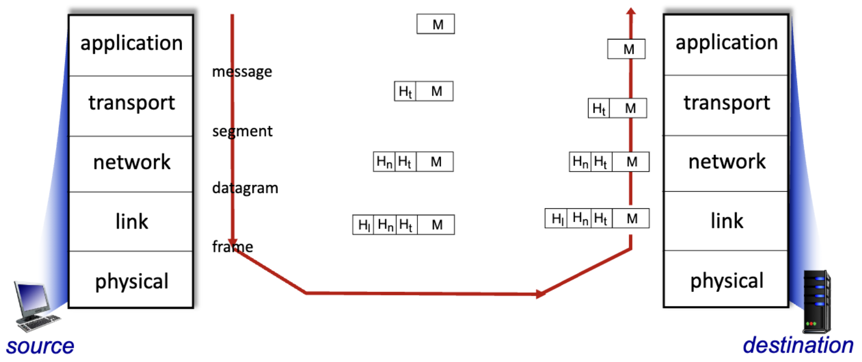
Layering Principle: Modular design; each layer offers services to the layer above and relies on services from the layer below. Then, change in layer service implementation is isolated from other layers.

Internet (TCP/IP) model: 5 layers

1. **Application:** Supports network applications (e.g. *HTTP, IMAP, SMTP, DNS*)
2. **Transport:** Process-to-process data transfering (e.g. *TCP, UDP*)
3. **Network:** Routes datagrams from source to destination (e.g. *IP, routing protocols*)
4. **Link:** Transfers data between neighboring network elements (e.g. *Ethernet, Wi-Fi, PPP*)
5. **Physical:** bits "on the wire"

Encapsulation

As data travels down the stack, each layer **adds a header**. As data travels up, each layer **removes its header**. This crates a **link-layer frame** via what we call **encapsulation**.



Chapter 2 Summary

Principles of Network Applications

The key idea is to enable programs to be able to run on **different end systems** — network-core devices **do not run user applications**.

Client-Server Paradigm

Here, we divide communication into two types of entities:

- **Server:** **Always-on** host, **permanent IP** address, *often in data centers*
- **Clients:** *Communicate with server*, may be intermittently connected, may have dynamic IP addresses, **do not communicate directly with each other**

Some examples of client-server paradigm are: *HTTP, IMAP, FTP*

Peer-Peer (P2P) Architecture

- **No always-on server**: **arbitrary end systems** directly communicate
- Peers request service from other peers, and provide service in return to other peers
  $\Rightarrow$ **Self scalability** – new peers bring **new service capacity and demands**
- Peers are *intermittently connected* and *change IP addresses*

An example of this is *P2P file sharing*.

Process Communication and Addressing

Definition of Processes

A **process** is a *program* running within a *host*.

- Within the ***same host***, two processes communicate using **inter-process communication**
- Between ***different hosts***, they communicate by exchanging **messages**
- **Client Process:** A process that ***initiates*** *communication*
- **Server Process:** A process that ***waits*** *to be contacted*

P2P applications still have client and server processes, despite not having clients or servers.

Sockets

A process *sends or receives* messages to or from its **socket**. In between a sending and receiving socket, we require some **transport infrastructure**.

Addressing Processes

We need an **identifier** to receive addresses, which includes both the **(unique 32-bit) IP address** and **port numbers** needed for the process (since the same host may run many different processes).

Transport-Layer Services

Application-Layer Protocol

An **application-layer protocol** defines:

- **Types of messages exchanged** (e.g. request, response)
- **Message Syntax**
- **Message Semantics** (i.e. meaning of information in fields)
- **Rules** for when to **send and respond** to messages

Examples of **open protocols** are *HTTP, SMTP*. Examples of **proprietary protocols** are *Skype, Zoom*.

TCP vs UDP Services

TCP and UDP services can almost be seen as polar opposites:

- **TCP: Reliable transport** *flow control, congestion control* (throttle sender when network overloaded), connection-oriented **(setup required between client and server processes)**
- **TCP doesn't provide:** Timing, minimum throughput guarantee, security
- **UDP: Unreliable data transfer** (may be not received), but typically **faster speeds**
- **UDP doesn't provide:** What TCP doesn't + what TCP does

Most applications want to use TCP, but interactive games and Internet telephony may use UDP.

Transport Layer Security

Transport Layer Security (TLS) provides **encrypted** TCP connections and end-point authentication, since otherwise passwords are sent in plain text. It is implemented in the **application layer**.

Web and HTTP (HyperText Transfer Protocol)

Overview

HTTP is the web's *application-layer protocol* that uses the client/server model, where:

- **Client:** Browser that requests/receives (using HTTP protocol), and displays web objects
- **Server:** Web server sends (using HTTP protocol) objects in response to the requests

HTTP uses TCP and does it in basically the following steps:

- The client **initiates TCP connection** (creates a **socket**) to the server at **port 80**
- The server **accepts TCP connection** from the client
- HTTP messages are exchanged between the browser (HTTP client) and Web server (HTTP server)
- **TCP connection closed**

Note, HTTP is **stateless** – the server maintains **no information about past client requests**.

HTTP Connections: Persistent vs Non-Persistent

Persistent vs Non-Persistent HTTP Connections

Non-Persistent HTTP	**Persistent HTTP**
1. TCP connection opened	1. TCP connection opened to a server
2. **At most one object** sent over TCP connection	2. **Multiple objects** can be sent over a TCP connection between client and that server
3. TCP connection closed	3. TCP connection closed

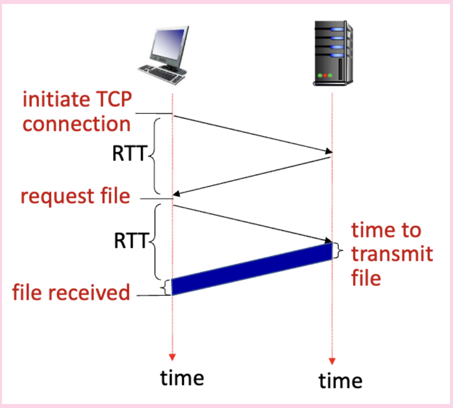
For non-persistent HTTP, downloading **multiple objects requires multiple connections.**

RTT

RTT is the time for a small packet to **travel from client to server and back**.

HTTP Response Time Per Object

For **non-persistent HTTP**, the HTTP response time **per object** is:



Hence, **Non-persistent PER OBJECT** HTTP response time = **2RTT + File transmisison time**

Similarly, for **persistent HTTP**, instead of per object, we can transmit multiple files, hence **Persistent TOTAL** HTTP response time = **2RTT + ALL files transmission time**

HTTP Request Messages

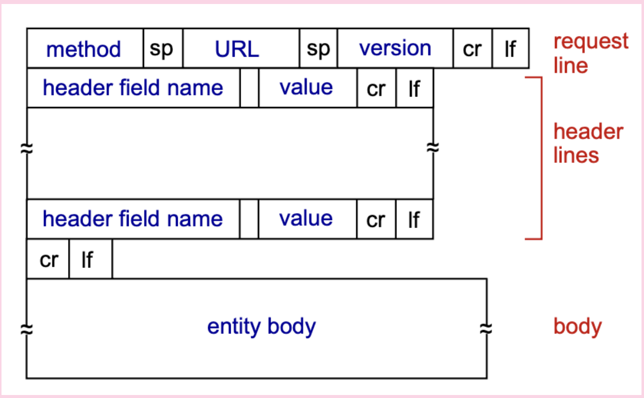
General Format

The image below provides the typical body of an HTTP request message.

Usually in the request line, the method is in all caps, such as *GET, POST, HEAD, PUT*.

`cr` and `lf` refer to **carriage return character** and **line-feed character** respectively, typically denoted as `\r` and `\f`.

A **blank line (CRLF CRLF)** indicates the **end of the header section**, not the entire message. The optional **entity body** follows this blank line.



Details About HTTP Request Message Methods

There are four main message methods:

- **POST:** Web page includes **form input**, with the user input sent from client to server in the **body of a HTTP POST** request
- **GET:** It **sends data to a server**, oftentimes including **user data in the URL field** of a HTTP GET request message **following a '?'**
- **HEAD: Requests headers** that would be returned **if** the URL were requested with HTTP GET
- **PUT: Uploads a new file** to the server, and completely **replaces the file** that exists at the URL with content in the **body of a HTTP PUT** request

Example of a Real HTTP Request Message

```
GET /index.html HTTP/1.1 \r\n
Host: www.example.com \r\n
User-Agent: Chrome/133.0 \r\n
Accept-Language: en-US \r\n
\r\n
<body>
\r\n
```

HTTP Response Messages

General Format

```
<protocol><status code>
<header lines>
<blank line>
<optional entity body>
```

HTTP Response Status Codes

The **status code** is what appears in the first line in the server-to-client response message. Here are some common sample codes:

- **200 OK:** Request **succeeded**, requested object later in this message
- **301 Moved Permanently:** Requested **object moved**, new location specified later in this message (in `Location: field`)
- **400 Bad Request:** Request message **not understood by server**
- **404 Not Found:** Requested document **not found** on this server
- **505 HTTP Version Not Supported**

Example of a Real HTTP Response Message

```
HTTP/1.1 200 OK\r\n
Date: Fri, 17 Oct 2025 07:00:00 GMT\r\n
Server: Apache/2.4\r\n
Content-Type: text/html\r\n
Content-Length: 512\r\n
\r\n
<html>...</html>
```

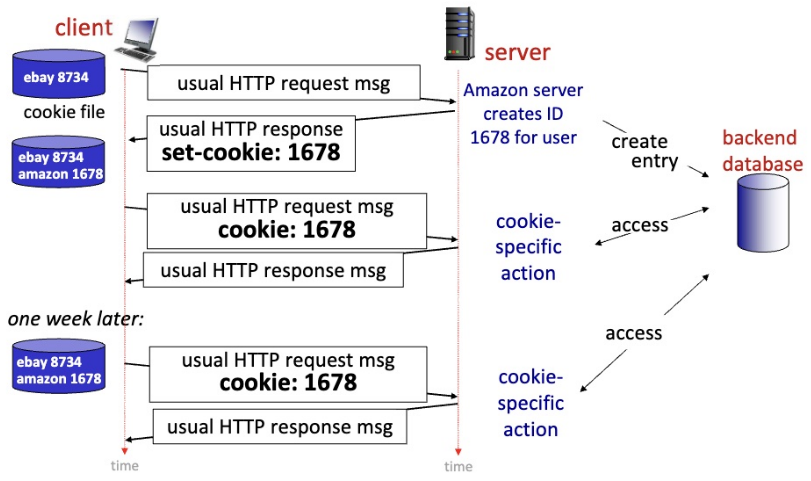
Cookies

Recall that the HTTP GET/response interaction is **stateless**. In order to maintain some states between transactions, websites and client browsers use **cookies**. To maintain cookies, we need four main components:

- Cookie header line in **HTTP response** message
- Cookie header line in **next HTTP request** message
- Cookie file kept on **user's host** (managed by user's browser)
- Backend **database at website**

They are typically used for *authorization, shopping carts, or ad recommendations*. Third party cookies (persistent cookies) allow for the same cookie value to be tracked across multiple websites, which is an invasion of privacy.

The following image below shows how cookies are maintained:



Web Caches (i.e. Proxy Servers)

Goal

To satisfy client requests without requesting from the origin server, which helps reduce response time and **traffic on an institution's access link**

Implementation

We have the user's broswer point to a **web cache**, then have the browser send all HTTPs to the web cache only. *(More specifically, if the object is in the cache, the cache returns the object directly to the client. Else, it requests the object from the origin server, caches the object, then returns the object to the client.)*

The server tells cache about the object's allowed caching in the response header as either of the two:

- `Cache-Control: max-age=<seconds>`
- `Cache-Control: no-cache`

Thus, the web cache acts **both as a client (to origin) and a server (to users)**.

Calculation

Typically, the $\boxed{\text{End-end delay} = \text{Internet delay} + \text{Access link delay} + \text{LAN delay}}$, where the LAN delay is typically in microseconds.

What **caching minimizes is the number of times the access link is used**, which thus minimizes access link delay.

Conditional GET

Goal: Don't send object if cache has up-to-date cached version, to reduce object transmission delay.

To do so, we maintain the following:

- **Client:** Store date of cached copy in HTTP request via `if-modified-since: <date>`
- **Server:** If cached copy is up-to-date, don't return an object, and send status code **`HTTP/1.0 304 Not Modified`**. Otherwise, do a *normal HTTP GET request*.

HTTP/2 and HTTP/3

Key Goal

Decrease delay in **multi-object** HTTP requests.

HTTP/1.1

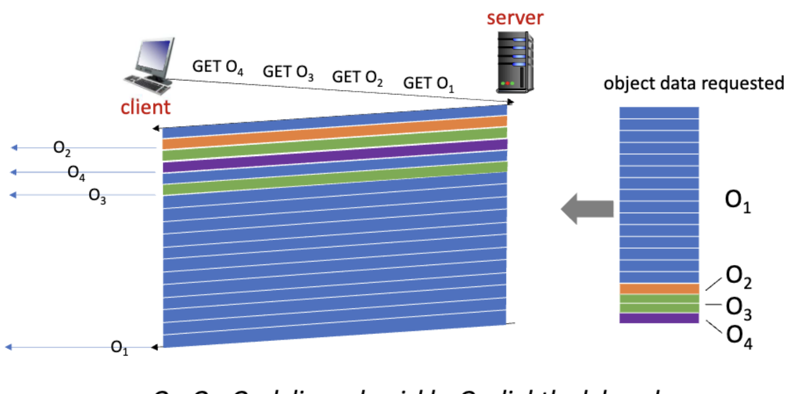
Introduced **multiple, pipelined GETs** over a *single TCP connection*

- Server responds **in-order FCFS (First Come First Serve)** to GET requests
- With FCFS, small objects may have to **wait for transmission behind large objects** (which is known as **HOL, i.e. head-of-line, blocking**)
- Loss recovery (retransmitting lost TCP segments) stalls object transmission

HTTP/2

HTTP/2 increased flexibility at server in sending objects to client:

- Transmission order of requested objects based on *client-specified object priority* (may not be FCFS)
- Push unrequested objects to the client
- Divide objects into "frames", and **schedule frames to mitigate HOL blocking**, as shown below



O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

HTTP/3

HTTP/2 over a single TCP connection means **recovery from packet loss** still stalls all object transmissions

HTTP/3 added **security** over vanilla TCP connection and **per object error-control and congestion-control** over UDP (hence minimizing packet loss recovery).

E-Mail, SMTP, IMAP

Components

There are three major components:

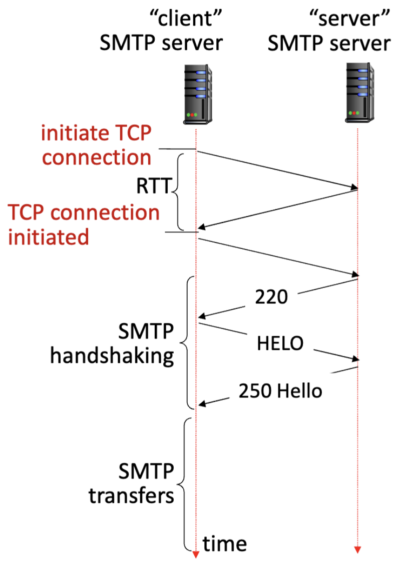
- **User agents:** The "mail reader" used to compose, edit, and read mail messages (such as *Gmail, Outlook, mail client*)
- **Mail servers:** We have the **mailbox containing incoming messages** for the user, and **message queue of outgoing** (to be sent) mail messages
- **Simple Mail Transfer (SMTP) Protocol:** Used between mail servers to send email messages, with the *"client" being the sending mail server, and "server" being the receiving mail server*

SMTP RFC 5321 (SMTP Protocol)

The SMTP uses TCP to *reliably* transfer email message from client (mail server initiating connection) to server at **port 25** via command/response interaction like HTTP.

There are three phases of transfer (as also shown in the image below):

1. SMTP **handshaking**
2. SMTP **transfer of messages**
3. SMTP **closure**



SMTP Messages

Example 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
```

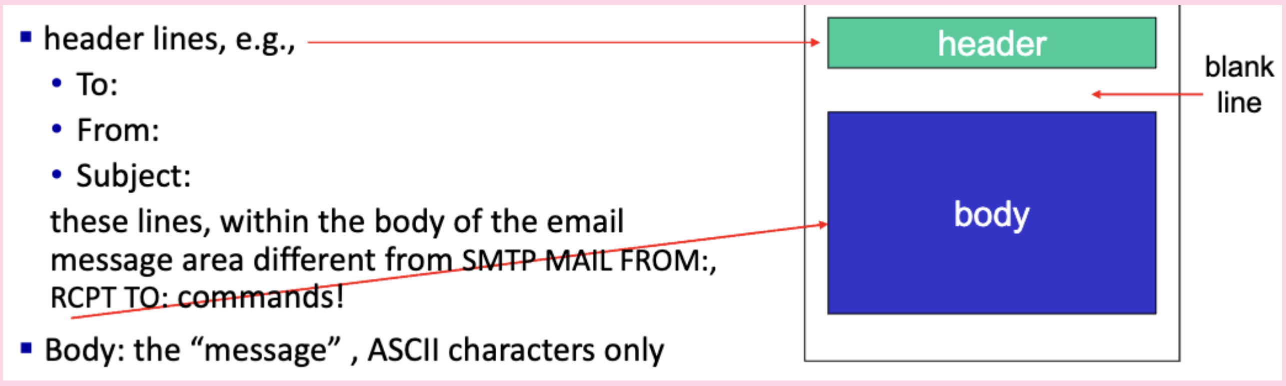
Ending of a Message

To end a message in SMTP, we use a line with only “.”

SMTP vs HTTP

HTTP is client pull whereas SMTP is client push . Both have ASCII commands/response interaction and status codes, but HTTP has **each object** encapsulated in its **own response message**, whereas SMTP sends **multiple objects** in a **multipart message**. Thus, **SMTP uses persistent connections** .

SMTP Mail Messsage Format (RFC 2822)



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

  these lines, within the body of the email message area different from SMTP MAIL FROM:, RCPT TO: commands!
- Body: the “message” , ASCII characters only

Retrieval Protocols

Notice, **SMTP is to send emails** only, we need **IMAP (Internet Mail Access Protocol) to retrieve emails** . In fact, **HTTP** provides web-based interface on top of SMTP and IMAP to form a full sending and receiving mail system.

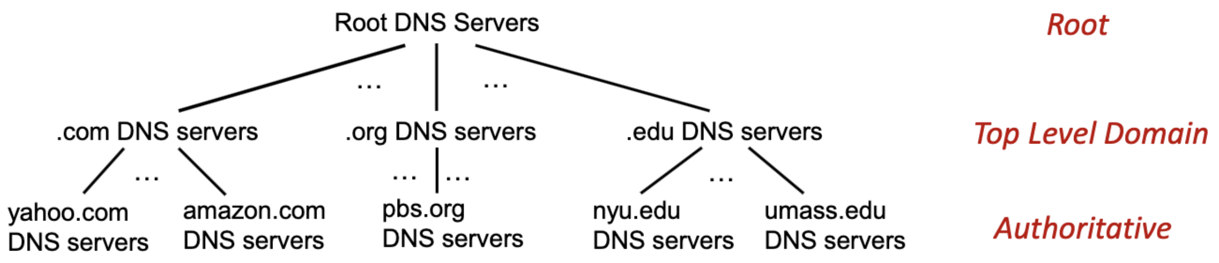
Domain Name System (DNS)

The **Domain Name System (DNS)** is a **distributed database** in the hierarchy of many name servers. DNS servers communicate to resolve names (**hostname-to-IP translation**) via **mappings**, and thus appears often in *application-layer protocols*. DNS also provides services such as **host or mail server aliasing**.

It tends to be physically **decentralized** because otherwise there will be too much traffic, and difficult maintance. This is especially because DNS needs to be *reliable and secure*, as a backbone of the Internet.

DNS Server Types

In fact, we can represent different types of DNS in a hierarchical database. Going from **top to down** based on proximity is how a corresponding IP address for an alias is found. As we can see, the servers are divided into four main types: **root name, top-level domain (TLD), authoritative, and local servers**. Note, **local DNS doesn't belong to the hierarchy below**



Root Name Servers

They act the official **last resort** of any name servers that cannot resolve a domain name. Root name servers know where to **find the Top-Level Domain (TLD)** servers and help initiate the DNS resolution process. There are 13 worldwide root name servers, managed by the ICANN (Internet Corporation for Assigned Names and Numbers).

Top-Level Domain (TLD) Servers

Responsible for **`.com`, `.org`, `.net`, `.edu`, `.aero`, `.jobs`, `.museums`** and all **top-level country domains**, such as `.ca`, `.uk`, `.fr`, etc. They redirect you to the authoritative server.

- **Network Solutions:** Manages **`.com` and `.net`** TLDs.
- **EDUCAUSE:** Manages **`.edu`** TLD, used by educational institutions.

Authoritative Servers

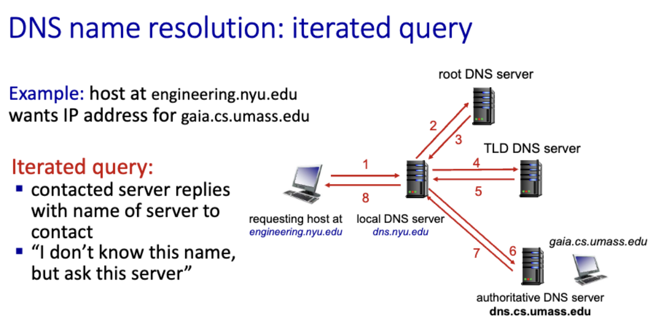
These are the **organization's own DNS servers**, providing authoritative hostname to IP mappings for organization's named hosts. They are maintained by the organization or service provider.

Local Servers

When a host makes a DNS query, it is sent to the local DNS server. The local DNS first answers by returning from its **local cache, and if unavailable, forwards request into the DNS hierarchy**. Each ISP has its own local DNS name server.

DNS Name Resolution: Iterated vs Recursive Query

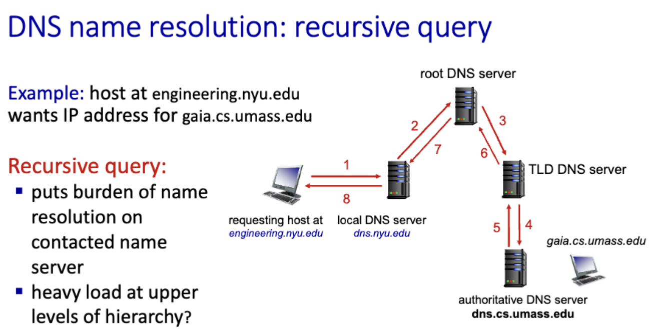
Iterated query is like **"I don't know this name but I'll recommend someone who does"**, whereas **recursive query** is like **"I don't know this name, but I'll ask someone who does then get back to you"**.


DNS name resolution: iterated query

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Iterated query:
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"

root DNS server
TLD DNS server
requesting host at engineering.nyu.edu
local DNS server dns.nyu.edu
gaia.cs.umass.edu
authoritative DNS server dns.cs.umass.edu
1
2
3
4
5
6
7
8


DNS name resolution: recursive query

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

Recursive query:
- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?

root DNS server
TLD DNS server
requesting host at engineering.nyu.edu
local DNS server dns.nyu.edu
gaia.cs.umass.edu
authoritative DNS server dns.cs.umass.edu
1
2
3
4
5
6
7
8

Caching Information

Once a name server learns a mapping, it caches it, so that it can **immediately return** a cached mapping in response to a query, until its entry timeouts after some time—**Time to Live (TTL)**. TLD servers are typically cached in **local name servers**. However, cached entries may be **out-of-date**. If a named host changes its IP address, it may not be known Internet-wide until all TTLs expire.

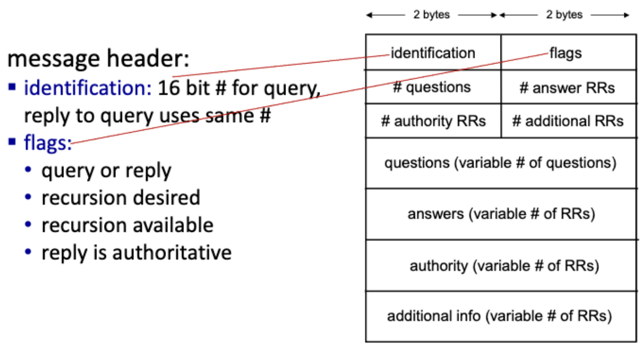
DNS Records

Resource Records (RR) are a distributed database, whose entries are (**`name`, `value`, `type`, `ttl`**). We type **`nslookup`** in order to access the following:

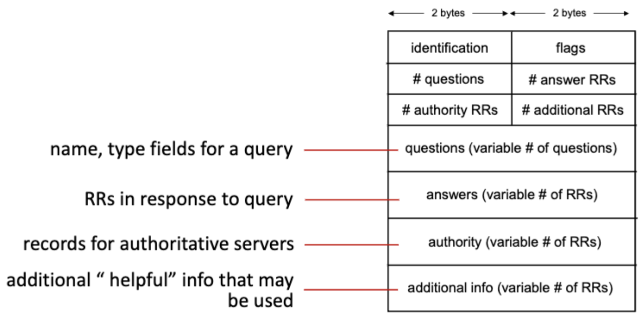
- **`type=A`**: `name` = **hostname**, `value` = **IP address**
- **`type=NS`**: `name` = **domain** (e.g. `.com`), `value` = hostname of **domain's authoritative name server**
- **`type=CNAME`**: `name` = **alias name** for some "canonical" (real) name, `value` = **canonical name**
- **`type=MX`**: `value` = name of **SMTP mail server** associated with `name`

DNS Message Format

DNS query and reply messages both have the same format, as shown in the images below.


message header:
- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative

2 bytes
2 bytes
identification
flags
# questions
# answer RRs
# authority RRs
# additional RRs
questions (variable # of questions)
answers (variable # of RRs)
authority (variable # of RRs)
additional info (variable # of RRs)


2 bytes
2 bytes
identification
flags
# questions
# answer RRs
# authority RRs
# additional RRs
name, type fields for a query
questions (variable # of questions)
RRs in response to query
answers (variable # of RRs)
records for authoritative servers
authority (variable # of RRs)
additional " helpful" info that may be used
additional info (variable # of RRs)

Example Setup for DNS

Say you have a new startup called "Network Utopia".

You must first create an **authoritative server locally**, perhaps with type `A` record at `www.networkutopia.com` and type `MX` record for `networkutopia.com`.

Then, you must first register a name, such as `networkutopia.com` at **DNS registrar** (e.g. Network Solutions). You'll have to provide **names and IP addresses of authoritative name server**, and the registrar will insert RRs of **types `NS` and `A`** into the `.com` TLD server.

Security

One type of attacks are **DDoS (Distributed DoS) attacks**, which **bombard root servers with traffic**, so that they are not successful to date. They may also bombard TLD servers.

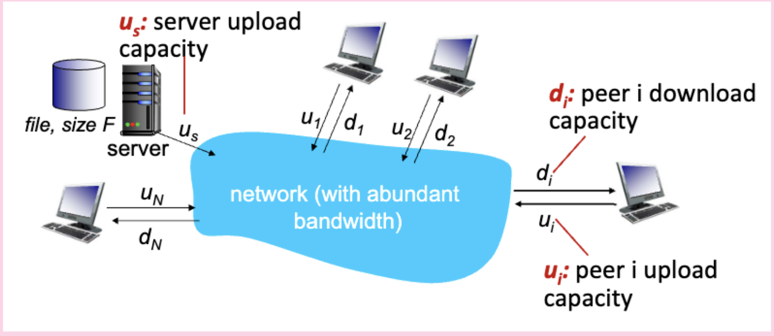
Another type of attacks are **spoofing attacks**, which intercept DNS queries and return **bogus replies**.

P2P Applications

File Distribution Time: Client-Server vs P2P

File Distribution Example

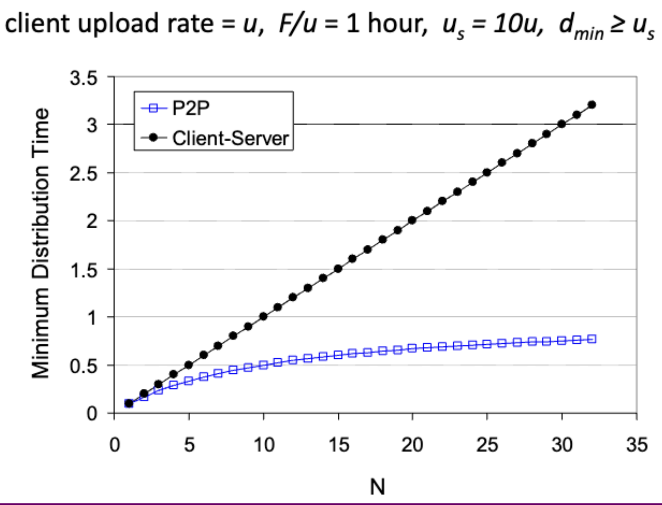
How much time does it take to distribute a file of **size F** from one server to **N peers**?


u_s: server upload capacity
file, size F
server
u_s
u_1
d_1
u_2
d_2
d_i: peer i download capacity
d_i
u_N
d_N
network (with abundant bandwidth)
u_i
u_i: peer i upload capacity

For **client-server**, the server transmission is done by sequentially sending N file copies. The time it takes to send one copy is $\frac{F}{u_s}$, so the time it takes to send N copies is $\frac{NF}{u_s}$. Each client must download one copy, so let d_{min} be the minimum client download rate, then the maximum client download rate is $\frac{F}{d_{\text{min}}}$, so the total time taken is $\boxed{D_{c-s} \geq \max\{\frac{NF}{u_s}, \frac{F}{d_{\text{min}}}\}}$.

For **P2P**, the server must upload at least one copy, which takes time $\frac{F}{u_s}$. Each client must download a file copy, with maximum time $\frac{F}{d_{\text{min}}}$. Hence, all clients in total must download NF bits, with limiting maximum upload rate as $u_s + \sum u_i$, i.e. both increase linearly in N. Thus, $\boxed{D_{P2P} \geq \max\{\frac{F}{u_s}, \frac{F}{d_{\text{min}}}, \frac{NF}{u_s + \sum u_i}\}}$

Here is an example of the difference between the time elapsed for client-server and P2P.



P2P File Distribution: BitTorrent

Overview

BitTorrent is a type of P2P file distribution, where each file is divided into **256Kb chunks**.

- **Torrent:** A group of peers exchanging chunks in a file
- **Tracker:** Tracks peers that participate in the torrent

When a peer joins a torrent, it has **no chunks**, but will **accumulate** them over time from other peers, and registers with tracker to get a list of peers, which connects it to neighbors.

While downloading, the peer must upload chunks to other peers. Once the peer has the entire file, it may **selfishly leave or altruistically remain** in torrent. We say that peers **"churn"**, meaning they may enter and leave the torrent.

Requesting Chunks

At any given time, different peers have different subsets of file chunks. Periodically, the user asks each peer for the list of chunks they have, they **request missing chunks from peers, rarest first**.

Sending Chunks: Tit-For-Tat

BitTorrent encourages fairness through a **Tit-for-Tat** strategy:

- Each peer **uploads to the four peers** that send to it at the **highest rate** (the "unchoked" peers)
- All other peers are **choked**, meaning they are temporarily denied uploads.
- Every 10 seconds, the peer re-evaluates which peers are its top four.
- Every 30 seconds, it **optimistically unchokes one random peer** to discover new partners.

Tit-for-tat resolves the issue of **free-riders (only downloading without uploading)**, by ensuring one must upload before being able to download.

Video Streaming and CDNs

Multimedia: Video — Terminology

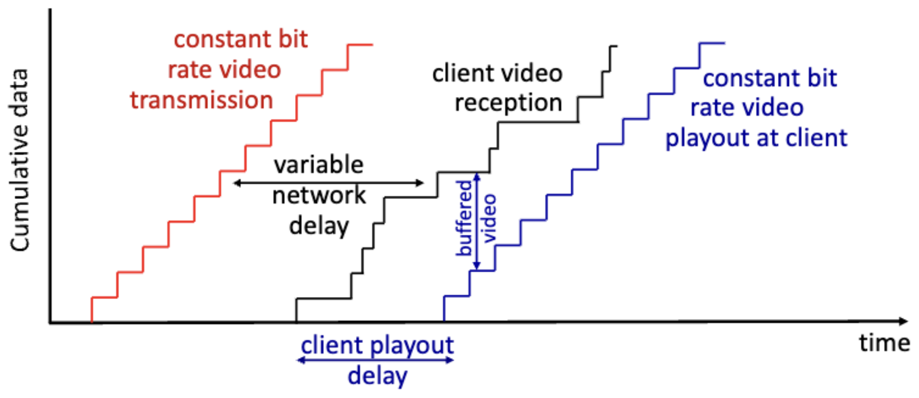
- Video: A sequence of images displayed at a **constant rate**
- Digital Image: Array of pixels, each represented by bits
- **Coding:** Use **redundancy** within and between images to **decrease the number of bits used** to encode images ; *spatial coding refers to it done withiin an image, and temporal coding refers to it done betweeen one image and the next*
- **Constant Bit Rate (CBR):** Video encoding rate is fixed
- **Variable Bit Rate (VBR):** Video encoding rate changes according to *spatial and temporal coding*

Streaming Stored Video

Streaming vs Downloading

Goal: Continuous playback despite network delay and rate variation.

- **Stored video:** File pre-recorded and stored at the server.
- **Streaming:** Client begins playback before the entire file is downloaded.
- **Client Buffer:** Stores several seconds of video to handle delay and jitter, so it plays smoother without having to have a fixed bitrate.



Streaming Multimedia: DASH

DASH: Dynamic Adaptive Streaming over HTTP

- Video file encoded into several versions with **different bitrates**.
- Each version is divided into **small chunks** (e.g. 2–10 seconds).
- **Manifest File:** Lists available bitrates and chunk URLs.
- **Client behavior:**
  - Downloads the manifest.
  - Measures available bandwidth.
  - Requests next chunk at the highest bitrate it can handle.

Advantage: **Adapts to changing network conditions** to avoid stalls.

Content Distribution Networks (CDNs)

Overview

A CDN **distributes copies of content across geographically separated servers** to reduce latency and server load.

- **Enter Deep (Type 1):** CDN servers **placed inside ISPs**, close to users (e.g. Akamai).
- **Bring Home (Type 2):** CDN servers in a **few large data centers** (e.g. Google, Netflix).
- **User Redirection:** DNS or HTTP redirects map clients to the nearest or least-loaded CDN node.

Goal: Reduce RTT, balance load, and minimize origin-server traffic.

Chapter 3 Summary

Transport-Layer Services

Transport services provide logical communication between application processes running on different hosts. Two transport protocols available to internet applications are **TCP and UDP**.

Overview

In transport protocols, the **sender breaks application messages into segments** and passes it to the **nextwork layer**, whereas the **receiver reassembles segments into messages**, and passes it into the **application layer**.

Transport Layer vs Network Layer

The **transport layer** is about the **communication between processes**, whereas the **network layer** is about the **communication between hosts**.

Multiplexing and Demultiplexing

To allow multiple applications to use the network simultaneously, the transport layer performs both **multiplexing** and **demultiplexing**.

Definitions

- **Multiplexing (Sender side):** Gathering data from multiple sockets, adding transport headers (with port numbers and addresses), and passing the resulting segments to the network layer. **Small to large transport.**
- **Demultiplexing (Receiver side):** Using header information (source/destination IP and port numbers) to deliver received segments to the correct socket/application process. **Large to small transport.**

Header Information

Each transport segment includes:

- Source **IP address**
- Destination **IP address**
- Source **port number**
- Destination **port number**

These fields are used by the receiver to identify the **appropriate receiving socket**.

Connectionless vs Connection-Oriented Demultiplexing

- **UDP (Connectionless):** Each segment is directed to a socket based only on its **destination port number**. Multiple senders sending to the same port reach the same receiving socket.
- **TCP (Connection-Oriented):** Each connection is identified by a unique **4-tuple**:

  (Source IP, Source Port, Destination IP, Destination Port)

  A server can distinguish multiple TCP connections on the same port (e.g., port 80 for multiple clients).

[Safe for Exam Criteria] UDP Checksum (Safety Addition)

The **UDP checksum** provides simple error detection. Its purpose is to detect bit errors, not correct them.

Overview

It is computed by treating the UDP segment as a sequence of 16-bit integers, summing them using **one's-complement arithmetic**, and taking the **one's complement** of the result.

At the receiver, all words (including the checksum) are summed again—if the result is all 1s, no error is detected.