Review

Transport

Network layer: communication between hosts

Transport layer: communication between processes

Muxing across many processes

Unit of data: segment

Transport

- Two principal transports: TCP and UDP
- TCP: Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup
- UDP: User Datagram Protocol
  - unreliable, unordered delivery
  - no-frills extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees

Flow control - sliding window

Sender keeps a list of the sequence # it can send

known as the *sending window*

Receiver also keeps a list of the acceptable sequence #

known as the *receiving window*

Sender and receiver negotiate the window size

sending window <= receiving window

Intuitively, we want to give users with "small" demands what they want, and evenly distribute the rest

Max-min fair allocation is such that

the lowest demand is maximized

after the lowest demand has been satisfied,
the second lowest demand is maximized

after the second lowest demand has been satisfied,
the third lowest demand is maximized

and so on...

Max-min fair allocation can be approximated by slowly increasing W until a loss is detected

Intuition

Progressively increase
the sending window size

max=receiving window

Whenever a loss is detected,
decrease the window size

signal of congestion

Repeat

UDP: Datagram messaging service

UDP provides a connectionless, unreliable transport service

- No-frills extension of “best-effort” IP

- UDP provides only two services to the App layer
  - Multiplexing/Demultiplexing among processes
  - Discarding corrupted packets (optional)

Why Would Anyone Use UDP?

- Finer control over what data is sent and when
  - As soon as an application process writes into the socket
  - … UDP will package the data and send the packet
- No delay for connection establishment
  - UDP just blasts away without any formal preliminaries
  - … which avoids introducing any unnecessary delays
- No connection state
  - No allocation of buffers, sequence #s, timers …
  - … making it easier to handle many active clients at once
- Small packet header overhead
  - UDP header is only 8 bytes

TCP: Reliable, in-order delivery

TCP provides a connection-oriented, reliable, bytestream transport service

- What UDP provides, plus:
  - Retransmission of lost and corrupted packets
  - Flow control (to not overflow receiver)
  - Congestion control (to not overload network)
  - “Connection” set-up & tear-down

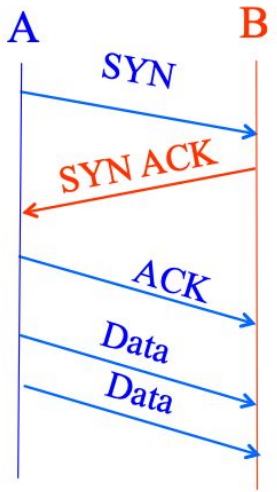
Basic Components of Reliability

- ACKs
  - Can’t be reliable without knowing whether data has arrived
  - TCP uses byte sequence numbers to identify payloads
- Checksums
  - Can’t be reliable without knowing whether data is corrupted
  - TCP does checksum over TCP and pseudoheader
- Timeouts and retransmissions
  - Can’t be reliable without retransmitting lost/corrupted data
  - TCP retransmits based on timeouts and duplicate ACKs
  - Timeout based on estimate of RTT

Other TCP Design Decisions

- Sliding window flow control
  - Allow W contiguous bytes to be in flight
- Cumulative acknowledgements
  - Selective ACKs (full information) also supported
- Single timer set after each payload is ACKed
  - Timer is effectively for the “next expected payload”
  - When timer goes off, resend that payload and wait
    - And double timeout period
- Various tricks related to “fast retransmit”
  - Using duplicate ACKs to trigger retransmission

Establishing a TCP Connection



Each host tells its ISN to the other host.

Three-way handshake to establish connection

- Host A sends a **SYN** (open; “synchronize sequence numbers”)
- Host B returns a SYN acknowledgment (**SYN ACK**)
- Host A sends an **ACK** to acknowledge the SYN ACK

Reliability: TCP Retransmissions

- Reliability requires retransmitting lost data
- Involves setting timer and retransmitting on timeout
- TCP resets timer whenever new data is ACKed
  - Retx of packet containing “next byte” when timer goes off

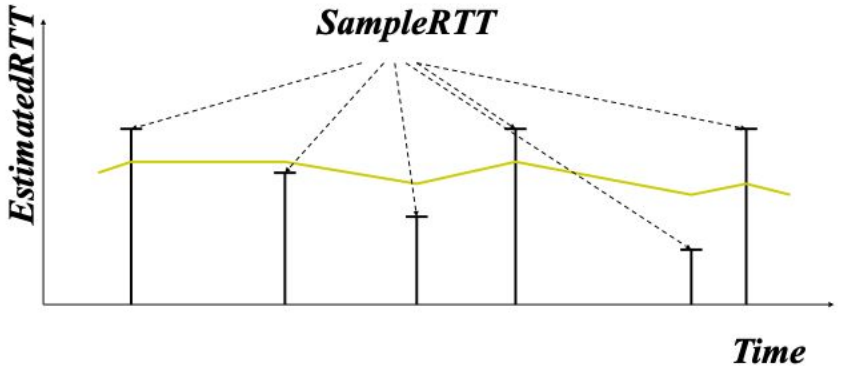
RTT Estimation

Use exponential averaging of RTT samples

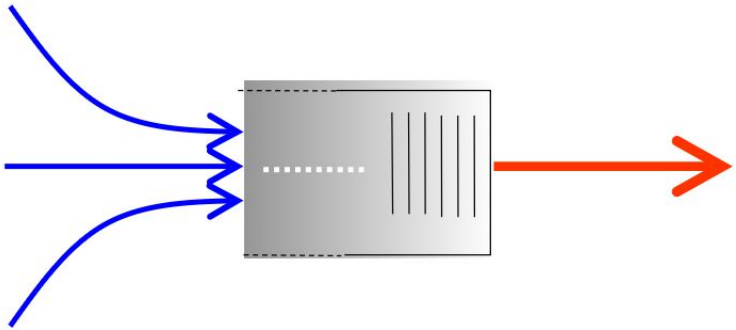
$$SampleRTT = AckRcvdTime - SendPacketTime$$

$$EstimatedRTT = \alpha \times EstimatedRTT + (1-\alpha) \times SampleRTT$$

$$0 < \alpha \leq 1$$



Because of traffic burstiness and lack of BW reservation, congestion is inevitable



If many packets arrive within
a short period of time
the node cannot keep up anymore

Congestion control differs from flow control

Flow control	prevents one fast sender from overloading a slow receiver
Congestion control	prevents a set of senders from overloading the network

TCP solves both using two distinct windows

Flow control

prevents one fast sender from overloading a slow receiver

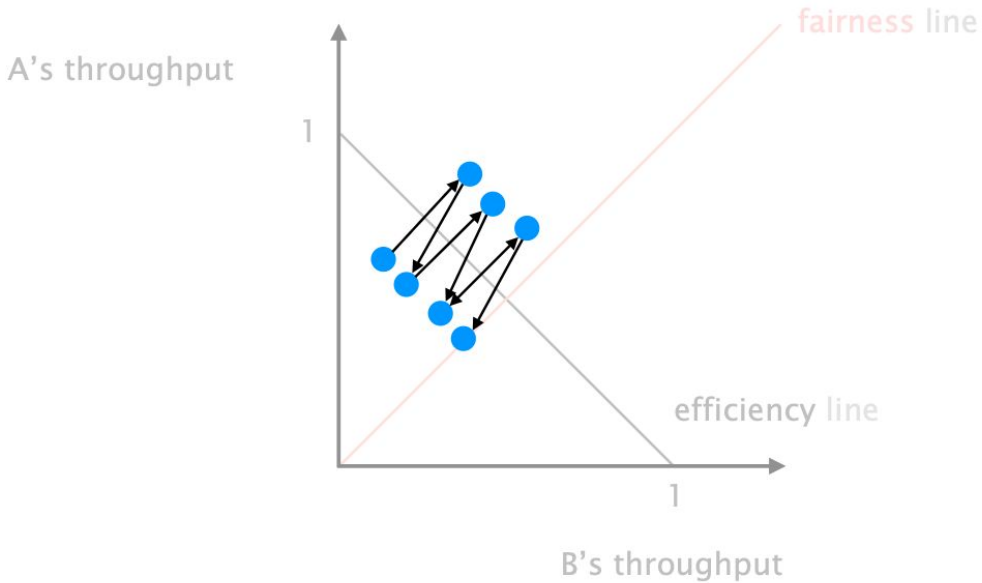
solved using a **receiving window**

Congestion control

prevents a set of senders from overloading the network

solved using a **“congestion” window**

AIMD converge to fairness and efficiency, it then fluctuates around the optimum (in a stable way)



AIMD converge to fairness and efficiency, it then fluctuates around the optimum (in a stable way)

Intuition

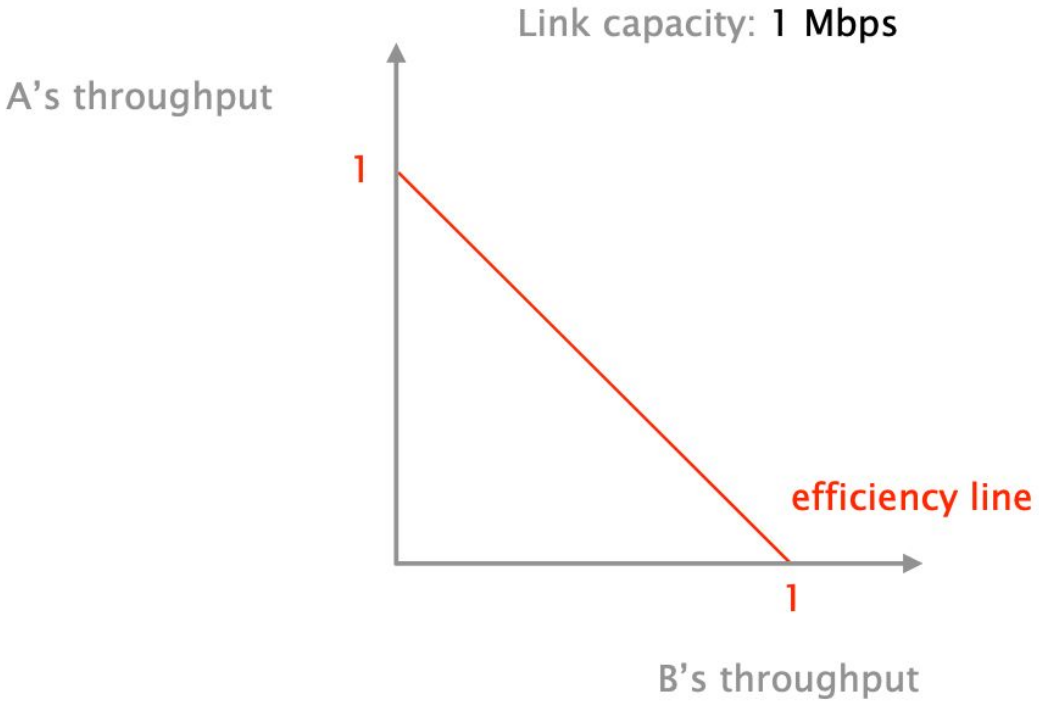
During increase,

both flows gain bandwidth at the same rate

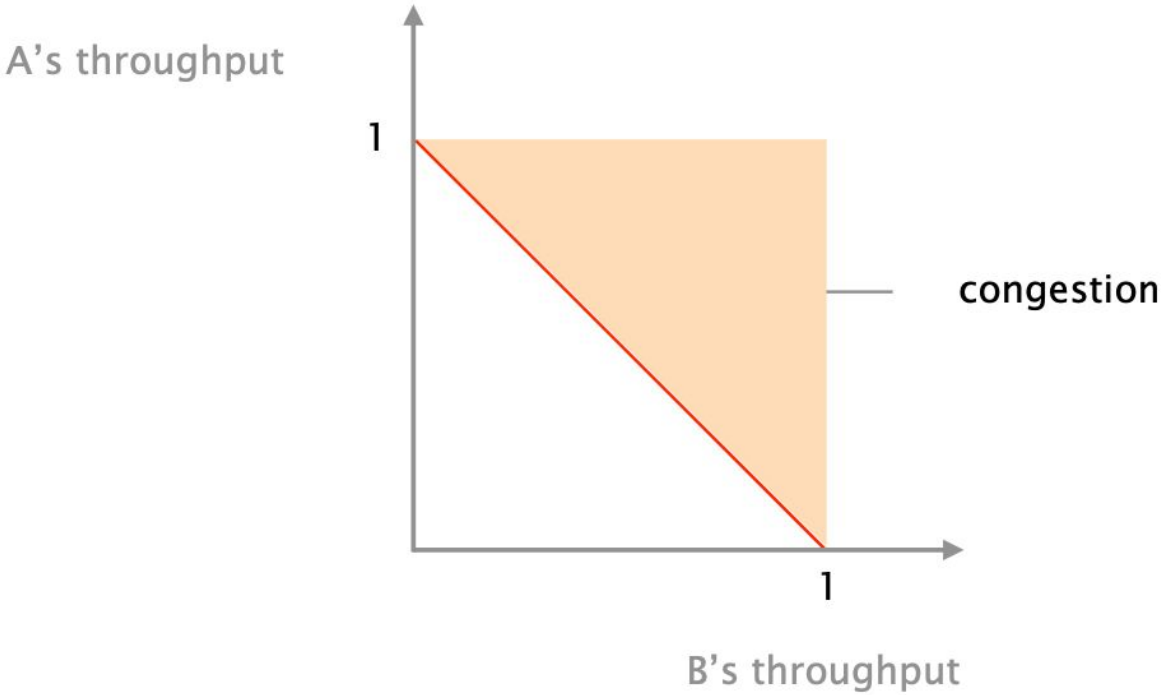
During decrease,

the faster flow releases more

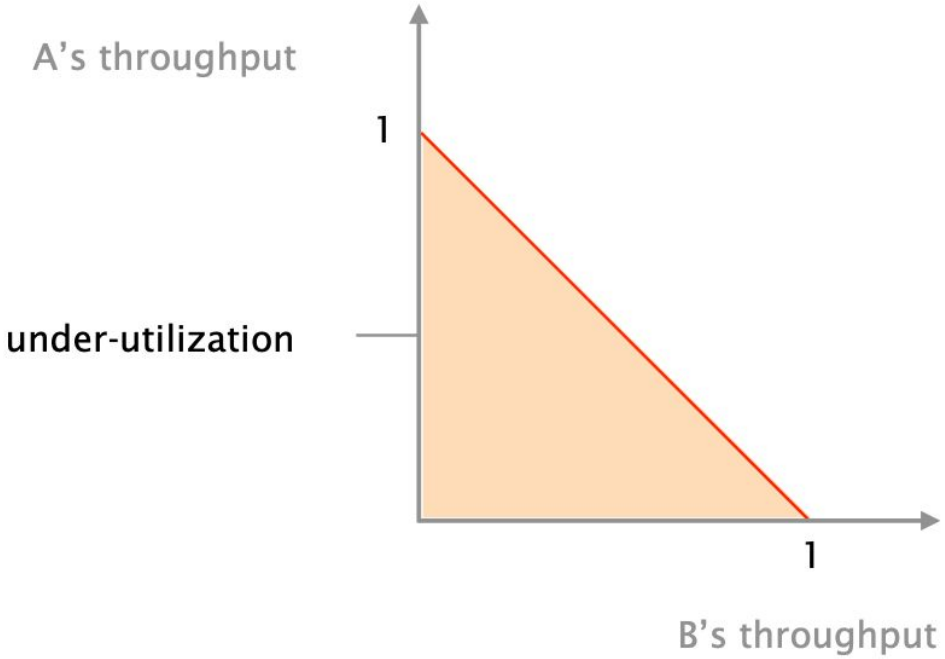
The goal of congestion control is to bring the system as close as possible to this line, and stay there



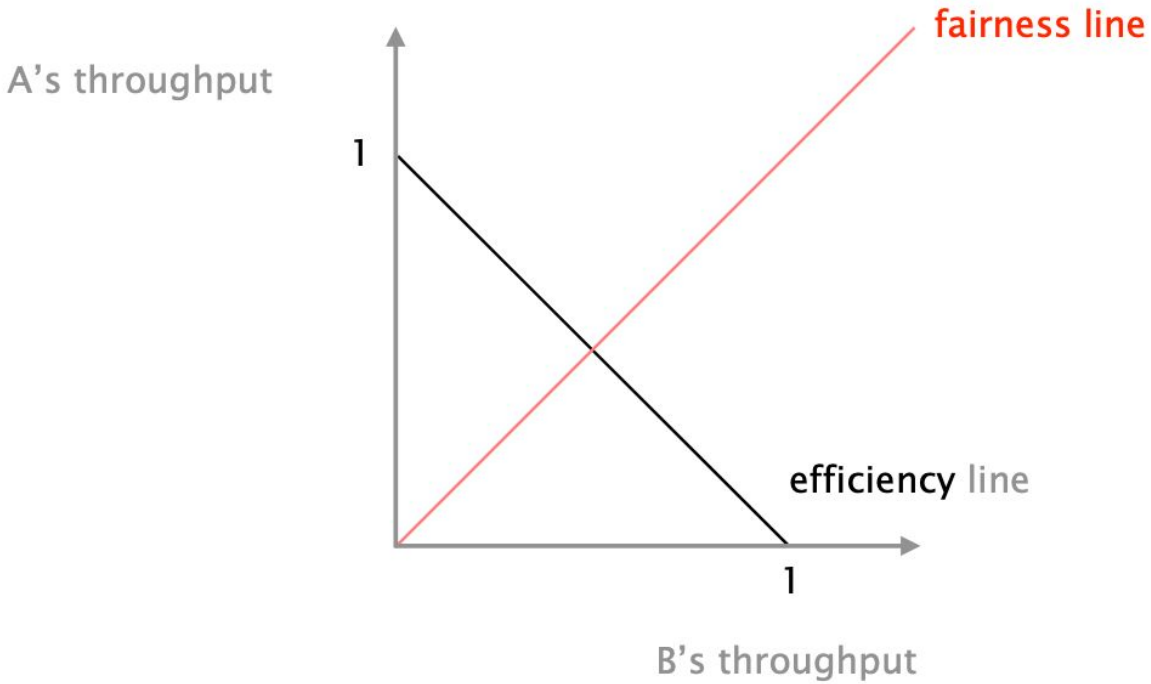
The goal of congestion control is to bring the system as close as possible to this line, and stay there



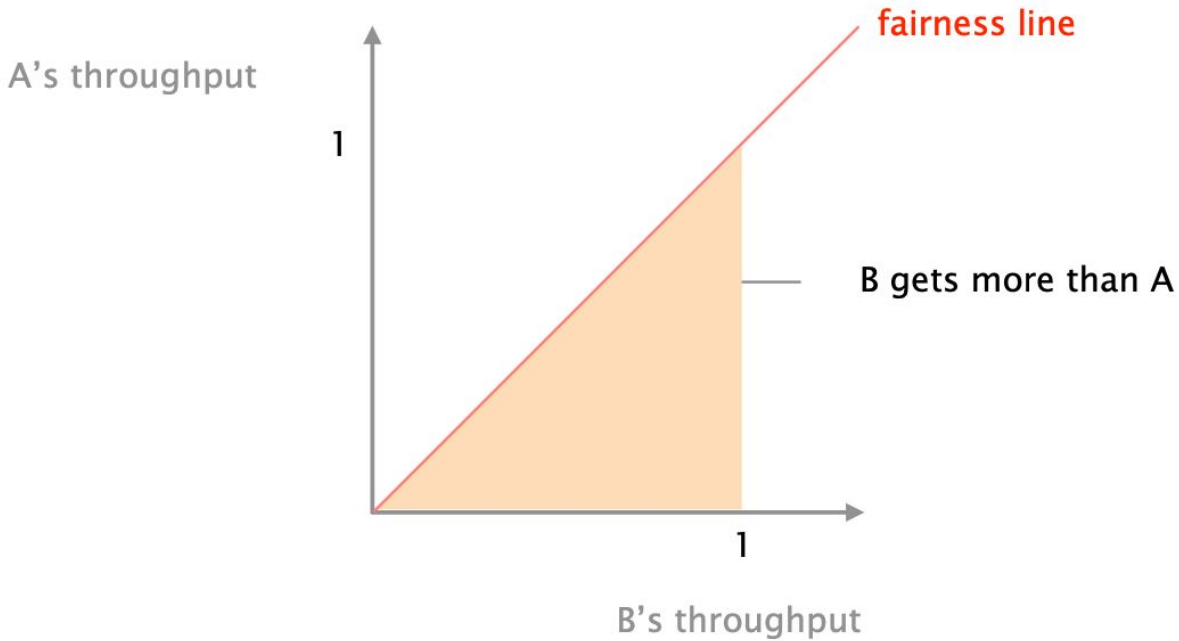
The goal of congestion control is to bring the system as close as possible to this line, and stay there



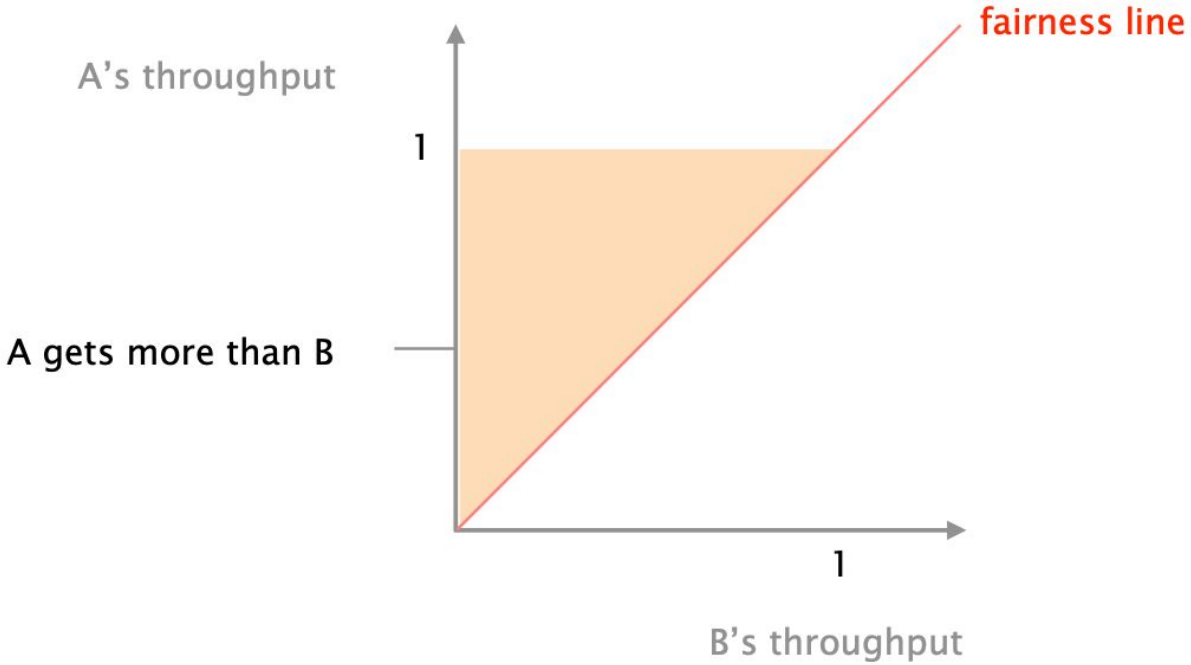
The system is fair whenever A and B have equal throughput, defining a fairness line where a = b



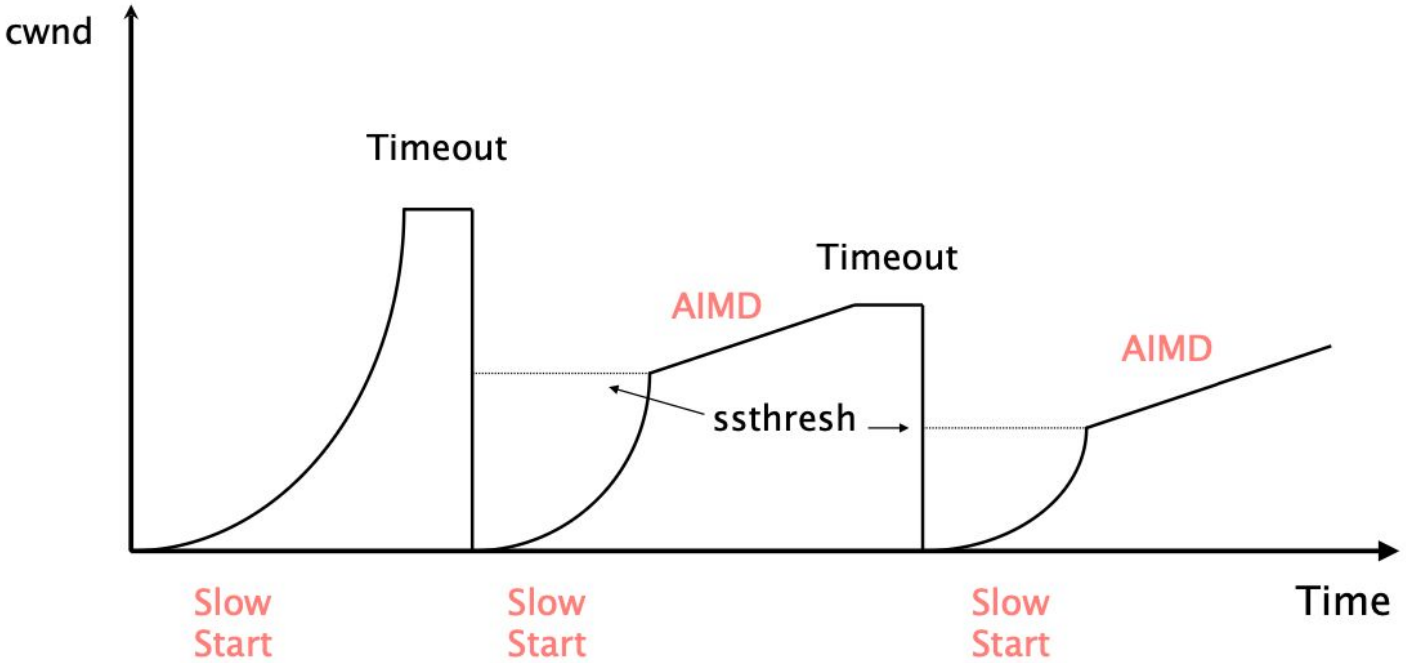
The system is fair whenever A and B have equal throughput, defining a fairness line where a = b



The system is fair whenever A and B have equal throughput, defining a fairness line where a = b



The congestion window of a TCP session typically undergoes multiple cycles of slow-start/AIMD



Detecting losses can be done using ACKs or timeouts, the two signal differ in their degree of severity

duplicated ACKs

mild congestion signal

packets are still making it

timeout

severe congestion signal

multiple consequent losses

TCP CUBIC

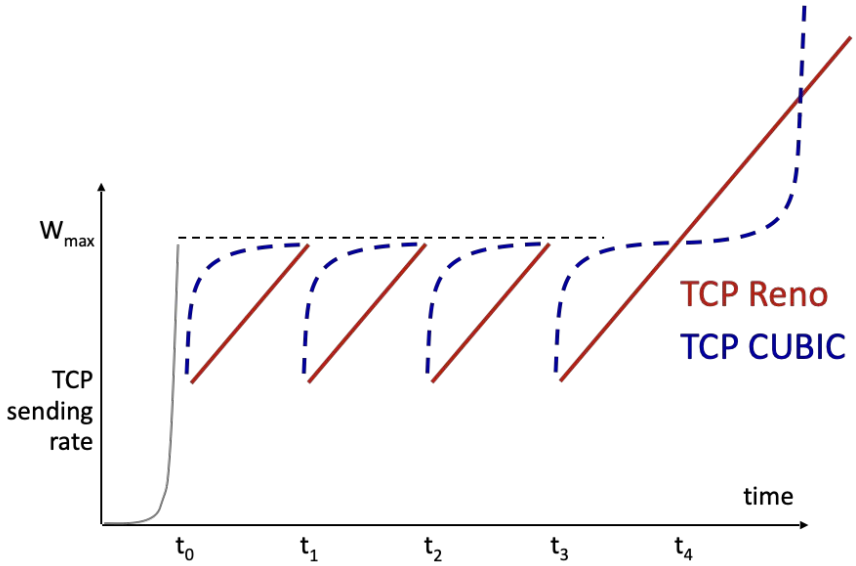
K: point in time when TCP window size will reach Wmax

- K itself is tuneable

increase W as a function of the cube of the distance between current time and K

- larger increases when further away from K
- smaller increases (cautious) when nearer K

TCP CUBIC default in Linux, most popular TCP for popular Web servers



TCP over “long, fat pipes”

example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput

throughput in terms of segment loss probability, L [Mathis 1997]:

$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT}\sqrt{L}}$$

to achieve 10 Gbps throughput, need a loss rate of $L = 2 \cdot 10^{-10}$ **YIKES**

versions of TCP for long, high-speed scenarios

TCP Throughput (Mbps)	RTTs between losses	cwnd	Packet Loss Rate P
1	5.5	8.3	0.02
10	55	83	0.0002
100	555	833	2×10^{-6}
1000	5555	8333	2×10^{-8}
10,000	55555	83333	2×10^{-10}

Leaving the transport layer for application layer

- DNS uses UDP or TCP
- Special protocol - not simply an application, it’s a fundamental network protocol for making the Internet operate
- www.hawaii.edu ->
  - web3x-vip-www00.its.hawaii.edu ->
    - 128.171.133.5

How do you resolve a name into an IP?

In olden times (1980s)

- all host to address mappings were in a file called hosts.txt
- in /etc/hosts
- Had to download regularly
- *still useful for certain situations. /etc/hosts takes precedence
    - https://raw.githubusercontent.com/StevenBlack/hosts/master/hosts

Problem:

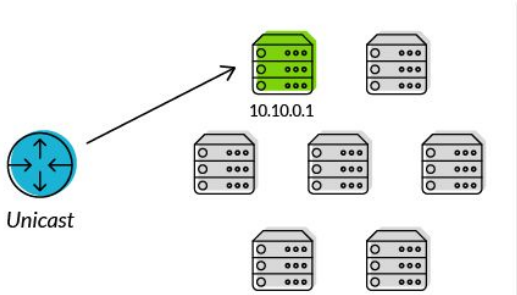
- Scalability in terms of
    - Management
    - Availability
    - Consistency

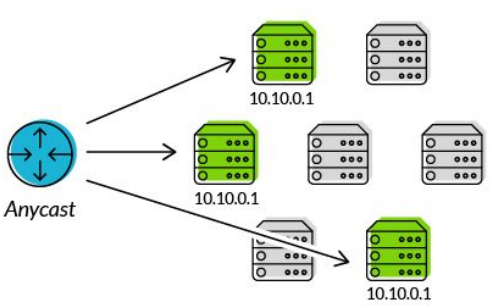
To scale, DNS adopt three intertwined hierarchies

naming structure	hierarchy of addresses https://ee.hawaii.edu/home/
Management	hierarchy of authority over names
Infrastructure	hierarchy of DNS servers

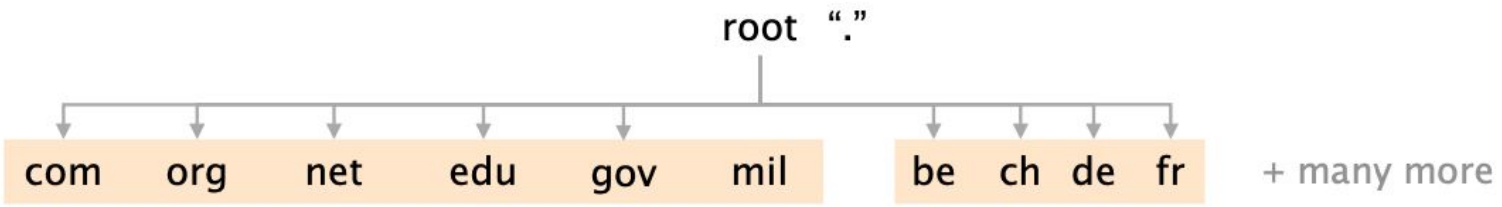
To scale root servers, operators rely on BGP anycast

- Routing finds shortest-paths
- If several locations announce the same prefix, then routing will deliver the packets to the “closest” location
- This enables seamless replications of resources





Top Level Domain (TLDs) sit below the root



Each root knows the address of all TLD servers

TLD and Authoritative DNS servers

- Top-level domain (TLD) servers
  - Generic domains (e.g., com, org, edu)
  - Country domains (e.g., uk, fr, cn, jp)
  - Special domains (e.g., arpa)
  - Typically managed professionally
    - Network Solutions maintains servers for “com”
    - Educause maintains servers for “edu”
- Authoritative DNS servers
  - Provide public records for hosts at an organization
  - For the organization’s servers (e.g., Web and mail)
  - Can be maintained locally or by a service provider

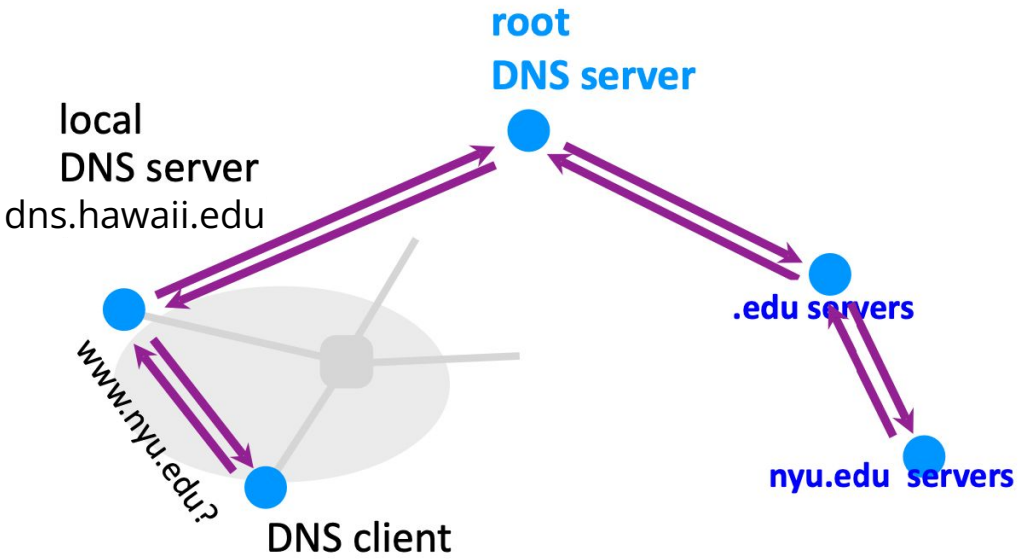
How does it work? Recursive vs Iterative Queries

Recursive query

- Ask each server to get answer for you (most common)

Iterative query

- Ask server who to ask next



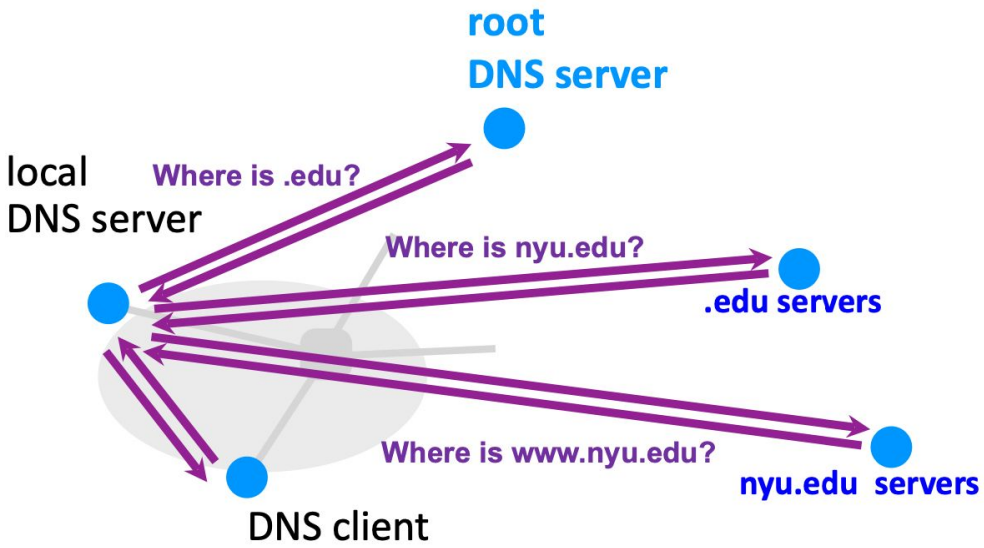
How does it work? Recursive vs Iterative Queries

Recursive query

- Ask each server to get answer for you (most common)

Iterative query

- Ask server who to ask next



DNS caching

- Performing all these queries takes time
  - And all this before actual communication takes place
  - E.g., 1-second latency before starting Web download
- Caching can greatly reduce overhead
  - The top-level servers very rarely change
  - Popular sites (e.g., www.cnn.com) visited often
  - Local DNS server often has the information cached
- How DNS caching works
  - DNS servers cache responses to queries
  - Responses include a “time to live” (TTL) field
  - Server deletes cached entry after TTL expires (OR SO THEY SAY)

DNS wrap up

- Full distributed, hierarchical system
  - Root -> TLD -> Authoritative
  - Anycast
- Underpins the modern Internet
  - Old way (/etc/hosts) couldn’t scale
- Recursive vs. Iterative modes
- Many different types of records
  - A / AAAA - used to translate domain to IP
  - PTR - reverse of A / AAAA
  - CNAME - name redirection
- Lots of work on DNS privacy / security these days

The web: basic requirements

- Something to represent content with links: **HTML**
- Client program to access/navigate/display content (e.g. HTML): **Web browser**
- A way to reference content: **URLs**
  - It's how you link/embed content to/in other content across a network
  - First general "handle" for arbitrary Internet content
  - Not just naming a host/processes (address/port)
- Something to host content: **Web servers**
- A protocol to get content from server to client: **HTTP**
  - Turns web URLs into TCP connections

Request headers are variable length but still human readable

Uses

Authorization info

Acceptable document types/encoding

From (user email)

Host (identify the server to which the request is sent)

Why would you need this? You're already connected?

Remember our DNS discussion about multiple names mapping to a single IP address - known as *virtual hosting*.

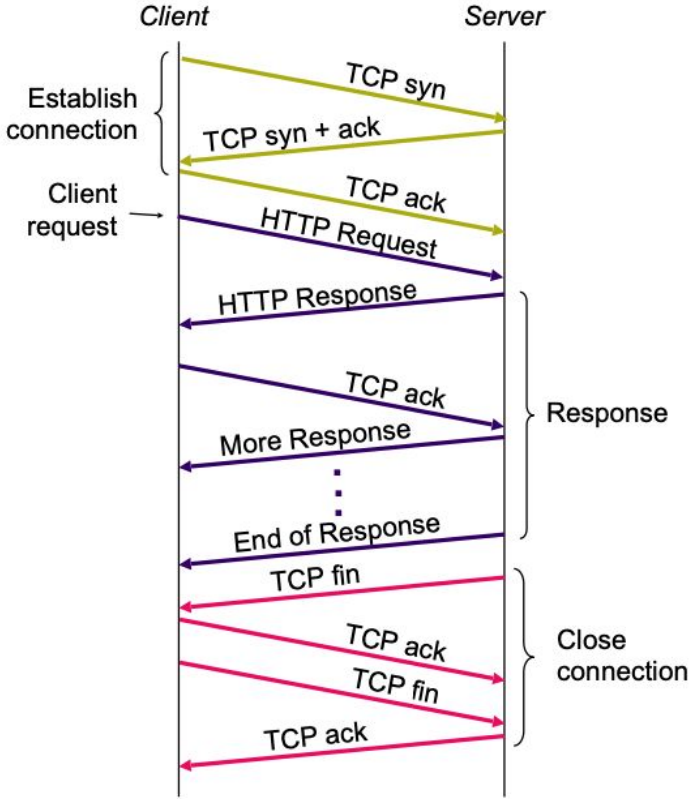
HyperText Transfer Protocol (HTTP)

- (Simple HTTP 1.0 “GET” request)

- Client creates TCP connection (port 80)
- Client sends request

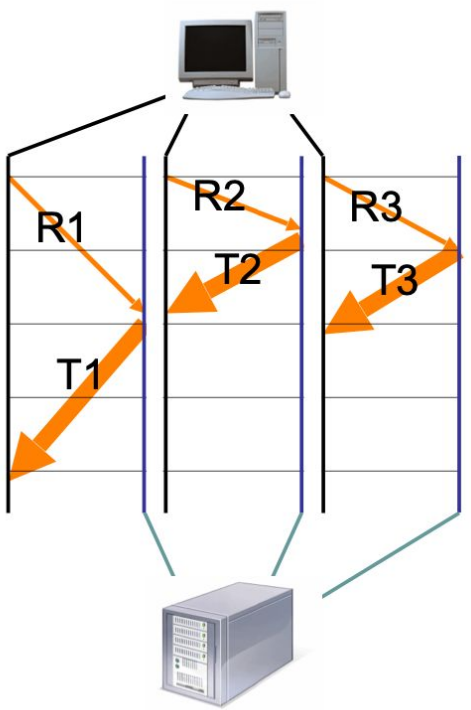
- Server sends response packets
- Client ACKs them

- Server closes connection



One solution to that problem is to use multiple TCP connections in parallel

User	Happy!
Content provider	Happy!
Network operator	Not Happy!
	Why?



Caching leverages the fact that highly popular content largely overlaps

Just think of how many times
you request the  logo
per day

vs

how often it *actually* changes

Caching it saves time for your browser
and decrease network and server load

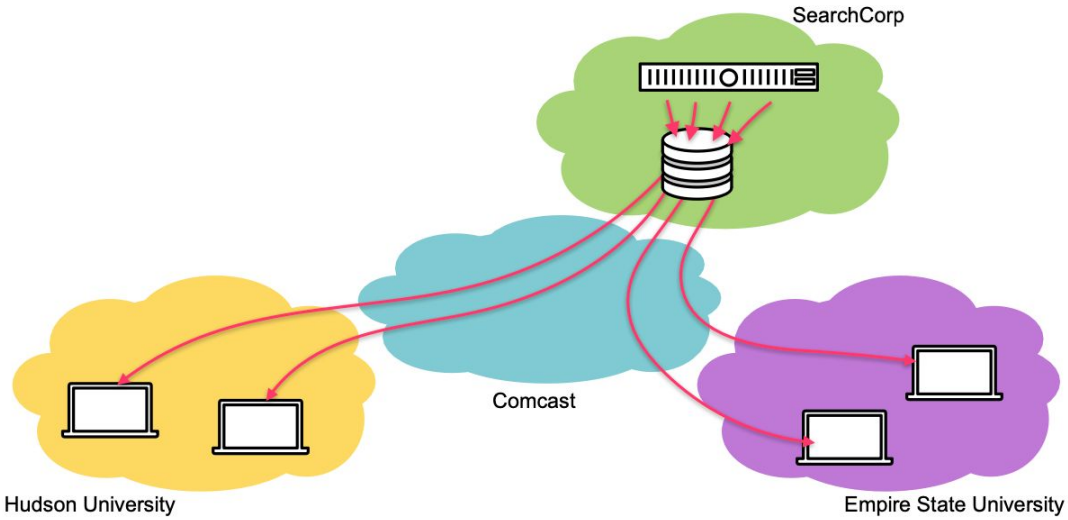
Caching can be (and is) performed at different locations

client	browser cache
close to the client	forward proxy Content Distribution Network (CDN)
close to the destination	reverse proxy

HTTP Caching

Reverse proxies

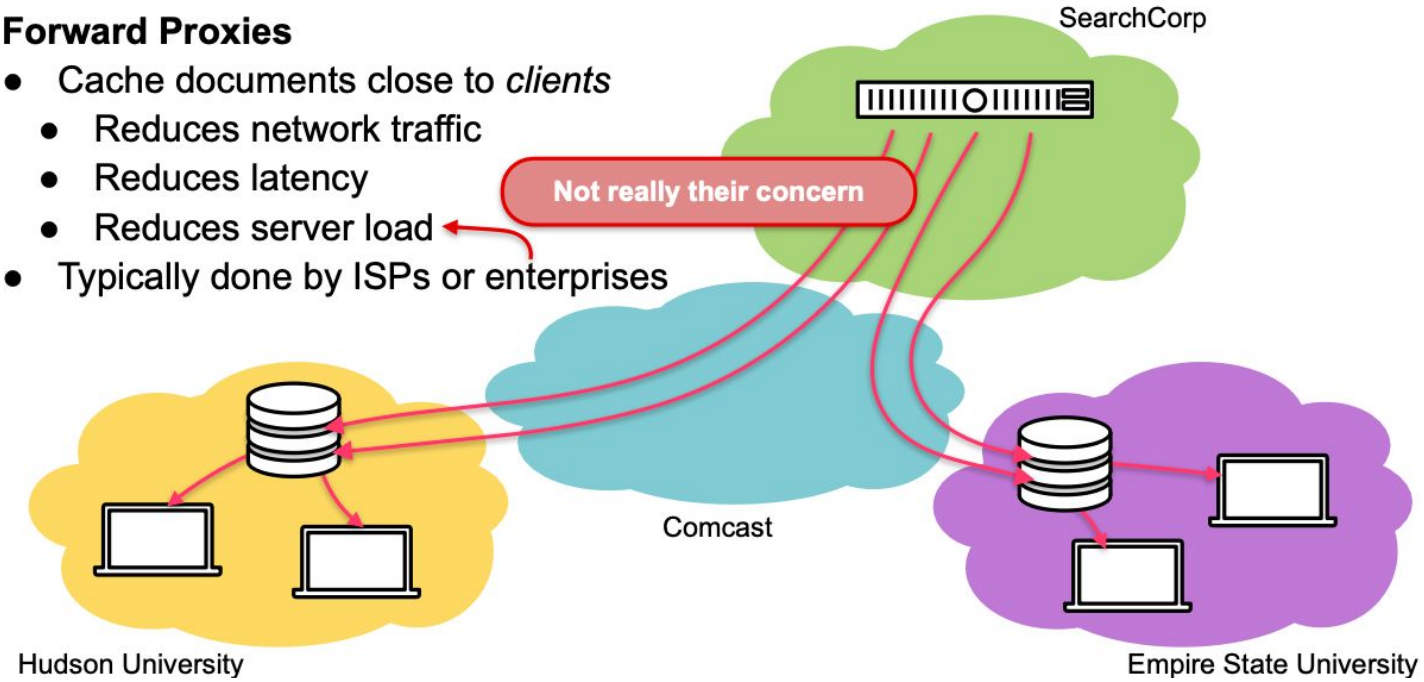
- Cache documents close to servers
    - Reduces server load
- Typically done by content provider



HTTP Caching

Forward Proxies

- Cache documents close to *clients*
  - Reduces network traffic
  - Reduces latency
  - Reduces server load
- Typically done by ISPs or enterprises



Content Delivery Networks

- Replication is a huge benefit to availability, scalability, and performance
  - We saw this with DNS
  - Can spread the load
  - Places content closer to clients (less latency)

  - Caching is a form of opportunistic replication
    - .. but what if a given organization doesn't have a forward proxy?
    - .. what if content provider and wants its content always replicated?

    - Idea: Caching and replication as a service — "CDNs 1.0"

CDNs “1.0”

- Large-scale distributed storage infrastructure
  - (Usually) administered by one entity
  - e.g., Akamai has 275,000+ servers in 136 countries
- Any server can host content for the many clients of the CDN (virtual hosting)

How do you get content onto the CDN servers?

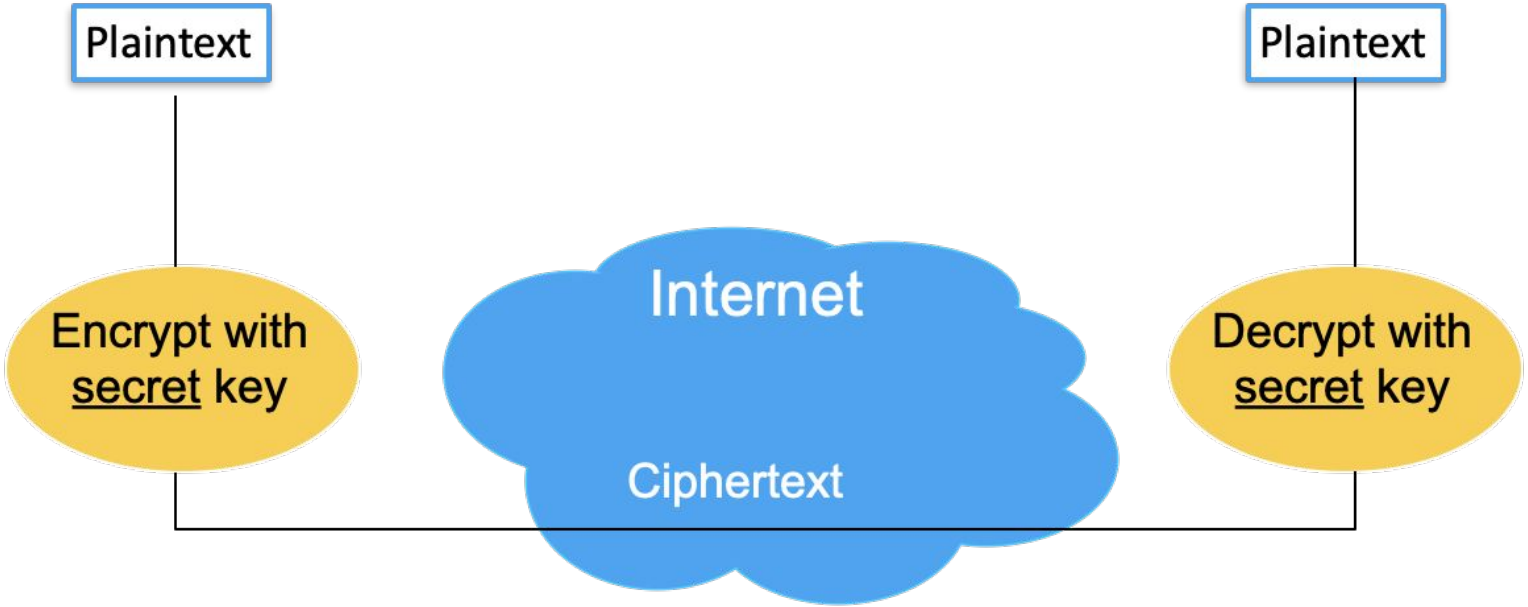
- Pull
  - Akamai servers act like a cache
  - Content provider gives CDN “origin” URL
  - When a client requests from Akamai
    - .. if cached, serve it
    - .. if not cached, request (“pull”) from origin, cache it, serve it
- Push
  - Akamai servers just act like normal servers
  - Content provider uploads content to CDN (“pushes” their content)
  - When a client requests from Akamai, just serve like any web server
- Various tradeoffs
  - Short version: pull is less work for content provider but push gives more control

Basic Requirements for Secure Communication

- **Availability**: Will the network deliver data?
    - Infrastructure compromise, DDoS
- **Authentication**: Who is this actor?
    - Spoofing, phishing
- **Integrity**: Do messages arrive in original form?
- **Confidentiality**: Can adversary read the data?
    - Sniffing, man-in-the-middle
- **Provenance**: Who is responsible for this data?
    - Forging responses, denying responsibility
    - Not who sent the data, but who created it

Fundamental crypto: symmetric keys

Both the sender and the receiver use the same secret keys

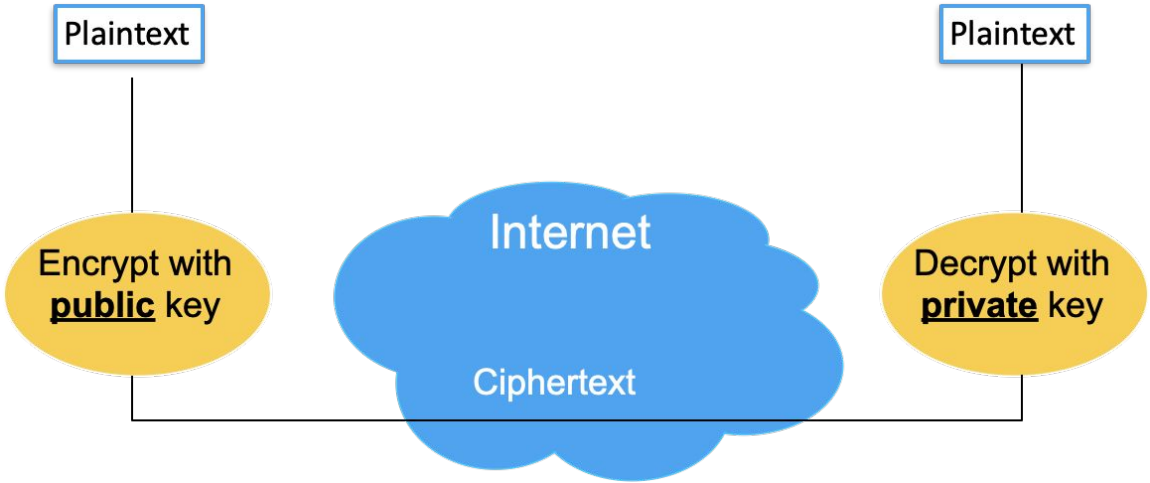


Fundamental crypto: asymmetric encryption (public key)

- Idea: use two different keys, one to encrypt (**e**) and one to decrypt (**d**)
  - A key pair
- Crucial property: knowing **e** does not give away **d**
- **e** can be public: everyone knows it
- If Alice wants to send to Bob, she fetches Bob's public key (say from Bob's home page) and encrypts with it
  - Alice can't decrypt what she's sending to Bob ...
  - ... but then, neither can anyone else (except Bob)

Public Key / Asymmetric Encryption

- Sender uses receiver's public key
  - Advertised to everyone
- Receiver uses complementary private key
  - Must be kept secret



Cryptographically Strong Hashes

- Hard to find collisions
  - Adversary can't find two inputs that produce same hash
  - Someone cannot alter message without modifying digest
  - Can succinctly refer to large objects

- Hard to invert
  - Given hash, adversary can't find input that produces it
  - Can refer obliquely to private objects (e.g., passwords)
    - Send hash of object rather than object itself

Putting It All Together: HTTPS

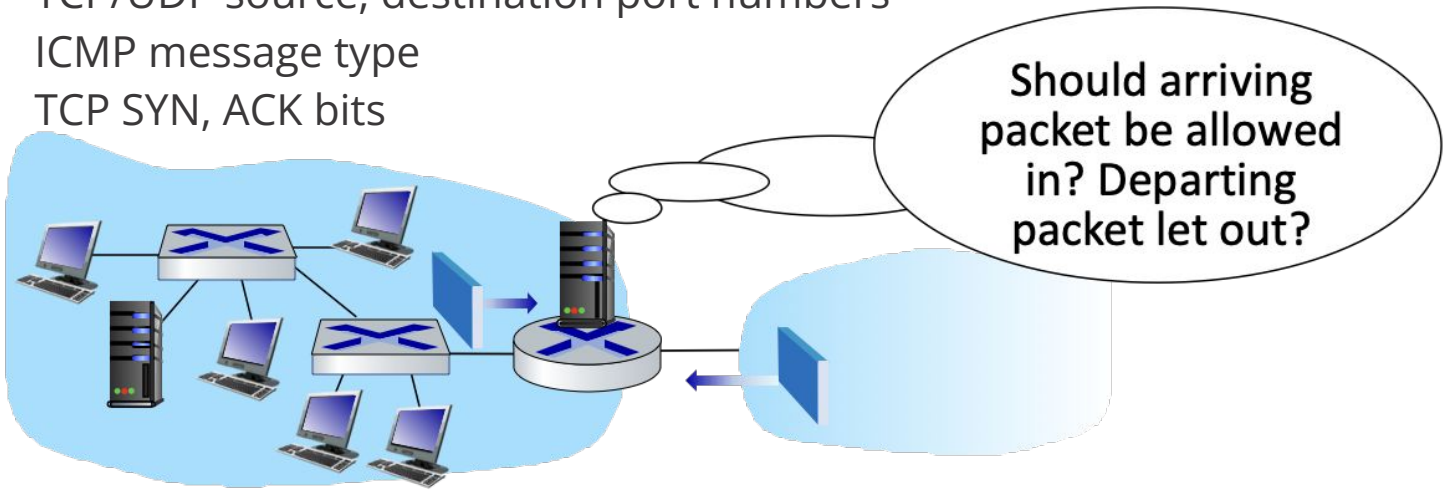
- Steps after clicking on https://www.amazon.com
- https = “Use HTTP over TLS”
  - SSL = Secure Socket Layer (older version)
  - TLS = Transport Layer Security
    - Successor to SSL, and compatible with it
  - RFC 4346, and many others
- Provides security layer (authentication, encryption) on top of transport layer
  - Fairly transparent to the app (once set up)

Firewalls

- prevent denial of service attacks:
  - SYN flooding: attacker establishes many bogus TCP connections, no resources left for “real” connections
- prevent illegal modification/access of internal data
  - e.g., attacker replaces CIA’s homepage with something else
- allow only authorized access to inside network
  - set of authenticated users/hosts
- three types of firewalls:
  - stateless packet filters
  - stateful packet filters
  - application gateways

Stateless Packet Filtering

- internal network connected to Internet via router firewall
- filters packet-by-packet, decision to forward/drop packet based on:
  - source IP address, destination IP address
  - TCP/UDP source, destination port numbers
  - ICMP message type
  - TCP SYN, ACK bits



Stateful Packet Filtering

- stateless packet filter: heavy handed tool
  - admits packets that “make no sense,” e.g., dest port = 80, ACK bit set, even though no TCP connection established

- stateful packet filter: track status of every TCP connection
  - track connection setup (SYN), teardown (FIN): determine whether incoming, outgoing packets “makes sense”
  - timeout inactive connections at firewall: no longer admit packets

Application gateways

filter packets on application data as well as on IP/TCP/UDP fields.

example: allow select internal users to telnet outside

1. require all telnet users to telnet through gateway.
2. for authorized users, gateway sets up telnet connection to dest host
   a. gateway relays data between 2 connections
3. router filter blocks all telnet connections not originating from gateway