

COMP3331: Networks and Applications

Haeohreum Kim (z5480978)

I. Introduction to the internet

A. What is the internet?

A ground up explanation.

At the *edge* of the internet, there are **billions** of connected devices. These devices are clients and servers, which have very specific meanings.

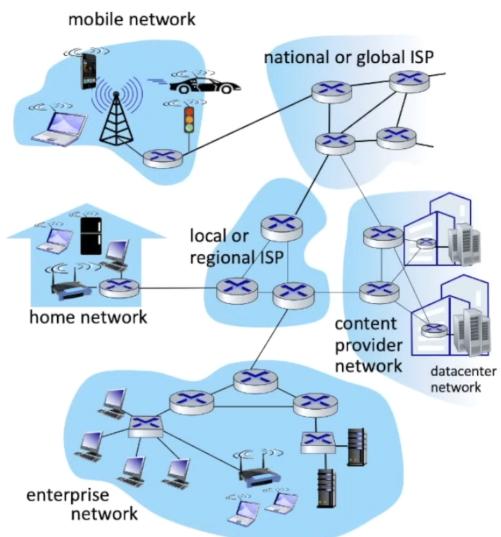
- Computers
- Refrigerators
- Gaming consoles
- Security cameras
- and so much more...

Going deeper, we find *packet switches* - these forward packets (chunks of data) across the network. There are two main types: **routers** and **switches**.

Between these packet switches, we have communication links - fiber, copper, radio and satellite. The transmission rate (the speed at which packets are transferred) between these links is what we call *bandwidth*.

And finally, we have this abstract idea of *networks*, which are managed by an organisation and are a collection of devices, routers and links.

Protocols are everywhere - and they control the receiving and sending of data across the network defined above. There must be standards that all these devices, routers and links agree on - being the Request for Comments (RFC).



A 'service' view. The internet, at the end of the day, is the infrastructure that provides services to applications. Furthermore, they provide a programming interface (API) to distributed applications:

- "Hooks" allow sending/receiving apps to connect to, using the Internet transport service
- Providing service options, analogous to postal services

B. What is an internet protocol?

Between humans, we have protocols. Languages themselves are specific protocols with rules, grammar, words and more so that we understand each other. Internet protocols are the same!

For devices to be able to communicate to each other, they need to have some agreed protocol of communication.

C. The network edge

The *network edge* refers to end points, like devices, modems and the devices that connect them. This includes *access networks*.

Access networks and physical media are generally the first point of contact for our devices to reach the 'Internet'. These can be residential (like our routers), institutional (like at university) or mobile (like 5G).

D. Wired access

Cable access networks. A cable access network connects multiple devices through a cable headend. Different households are split using *frequency division multiplexing (FDM)*, where different frequencies are used for different purposes.

Cable access networks are generally asymmetric - that is, the downstream (or download speed) is faster than the upstream (or upload speed). This is because most people are larger consumers than producers of data.

Digital Subscriber Line (DSL). uses existing telephone lines to a central office. DSL is also asymmetric. Data over the DSL line goes to the Internet, whereas voice goes to the telephone net.

The home network is generally powered by one of the above two - which is then connected to a router which distributes connections either wired (through Ethernet) or wireless.

E. Wireless access networks

Wireless Local Area Networks (WLANs). generally have a smaller proximity (10-100m), with either 11, 54 or 450 Mbps transmission rates.

Wide-area cellular access networks. provided by mobile, cellular network operators (10's km). 10's Mbps (but with 5G has increased this to gigabyte speeds).

F. Packets and hosts

A host has some data it wants to send. The host breaks up the data into chunks called *packets*, of length L bits. The packet will have some extra data called a *packet header*.

Overhead.

We call *overhead* the percentage of data in the packet that is not payload. That is if 40 bits of a 1000 bit packet payload were headers, then we have a 4% overhead.

$$\frac{\text{Amount of data that is headers}}{\text{Amount of payload data}}$$

The host transmits this packet through some medium at a transmission rate R .

Of course, the time taken is then $\frac{L}{R}$.

G. Physical media

Some important terminology and concepts

- Bit: A partition of data that propagates between transmitter/receiver pairs
- Physical link: what lies between transmitter and receiver
- Guided media: signals propagate in solid media: copper, fiber, coax
- Unguided media: signals propagate freely: e.g. radio
- Twisted pair (TP): two insulated copper pairs in a wire - 100 Mbps to 10 Gbps speed
- Coaxial cable: two concentric copper conductors, bidirectional, 100's Mbps per channel
- Fiber optic cable: glass fibers carrying light pulses, with each pulse being a bit. High speed and low error.
- Wireless radio: signal carried in various bands - and is 'broadcast'; any receiver can technically receive it. There are challenges with propagating waves.

H. Network core

I. Packet switching

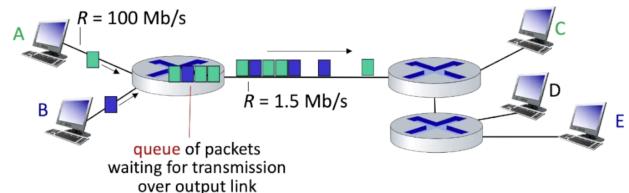
Hosts break application-layer messages into packets. These packets are then forwarded from one router to the next along a path until it reaches the destination.

The network core has two main functions:

- 1) Forwarding: forwarding is a local action - it moves arriving packets from a router's input link to the appropriate output link.
- 2) Routing: routing is a global action - which determines the source-destination paths taken by packets, which requires *routing algorithms*.

So routing requires knowledge beyond the local network - it needs to find a path; forwarding is done without the knowledge beyond the device the packet is currently at. Intermediate routers along a route are called **hop routers**. Packets **store and forward** packets by waiting for their entire transmission, and then forwarding them onto the next destination.

When multiple devices try to transmit packets through an access network at a faster rate than its transmission rate, then **queuing** will occur. A problem occurs when packets are queued and router's memory fill up. In this case, packets are dropped, causing the problem of *packet loss*.



J. Circuit switching

Resources are reserved through a 'call' - a path is determined and then that path is reserved. This creates a direct link from source-destination; there's no loss, and no delay. However, since resources are entirely reserved for some source-destination, resources are largely wasted.

1) **Frequency Division Multiplexing (FDM):** Optical, electromagnetic frequencies are divided into narrow frequency bands. Each call is allocated its own band, and can transmit at the max rate of that band.

2) **Time Division Multiplexing (TDM):** Time is divided into slots - each call is allocated a periodic slot, and can transmit at maximum rate (wide band) for the designated time slot.

K. Packet switching v.s circuit switching

Consider a router that routes N users. They all use 100 Mb/s when active, but are only active 10% of the time. The

router has a 1 Gbps link to the internet. With 100 Mb/s, we need 10 users to fill the entire link, which means we need to find the probability that more than 10 users are on the network at the same time. This is given by:

$$P(\text{Network overloaded}) = 1 - \sum_{i=1}^{10} \binom{N}{i} \left(\frac{1}{10}\right)^i \left(\frac{9}{10}\right)^{N-i}$$

The probability ends up being 0.0004. This means it's extremely unlikely, and was a major argument for the usage of packet switching.

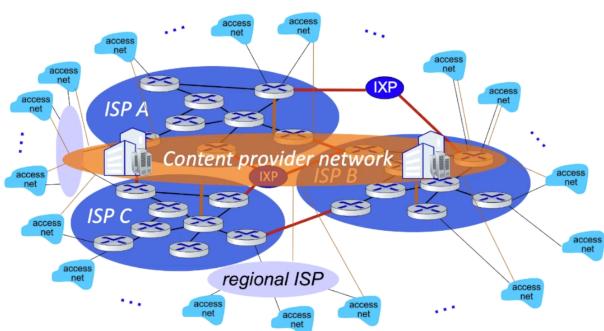
So is packet switching a slam dunk win?

- Packet-switching is great for bursty data; so data that is sent at an irregular interval
- Congestion is certainly possible
- Can we provide a consistent stream from source-destination like circuit switching using packet-switching?

Having direct end-to-end paths between every single access network is not feasible. Rather, we create smaller networks. On the internet, hosts connect to Internet Service Providers (ISPs). ISPs connect multiple access networks into a shared network.

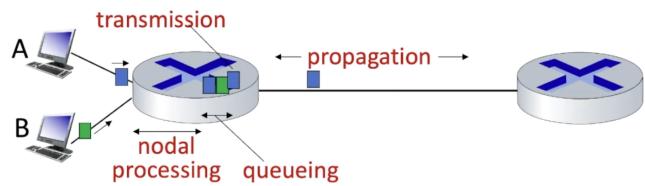
Of course, there are multiple ISPs. ISPs peer through each other at points called *Internet Exchange Points* (IXP). Different companies that provide different ISPs to different access networks must still be able to communicate across each other, so they create agreements to have peering locations.

Content provider networks, like Google and Facebook, are private networks that connect its data centres to the Internet, bypassing ISPs. They do this with *Content Delivery Networks* (CDN), which are edge servers for customers, but also have agreements with ISPs to connect to IXPs.



L. Network performance

M. The four sources of packet delay



Packet delay is defined by:

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

- d_{proc} : processing delay - forwarding, error checking, i microsecs
- d_{queue} : time waiting at output link for transmission
- d_{trans} : defined by L/R for L amount of bits at rate R .
- d_{prop} : length of the physical link, divided by the propagation speed.

Consider the following formula for *traffic intensity*:

$$t = \frac{L \cdot a}{R}$$

where L is the length of the packet, a is the average arrival rate of packets, and R is the bandwidth. We see as 1 is the average queue time, and < 0 as smaller queueing delays. Where $aL > R$, queueing occurs.

N. Tracking real delays in transmission (traceroute)

Using the tool traceroute allows us to find the route of a packet, as well as the delay of a packet along its route for each hop-router.

O. Throughput

Definition. Throughput is the rate at which bits are being sent from sender to receiver. It is given as a *rate* (bits/sec) at which bits are being sent from sender to receiver:

- instantaneous: rate at given point in time
 - average: rate over longer period of time
- $$\frac{\text{number of bits sent}}{\text{total time taken}}$$

You must consider the bottlenecks within the network - and find where the transmission rate is minimised. Let's consider a situation where some packets are being sent through two links, with rates R_S and R_C . The throughput will be given by $\min(R_S, R_C)$.

What if we have 10 devices going through their own respective links R_{S_i} , into one shared link to the destination R_d ? Then our throughput is $\min(R_{S_i}, R_d)/10$.

II. Application layer

The application layer is how users and software access the network.

Before we step into the application layer, what is a layer? Each layer *implements* a service. Of course, these layers interact with each other to provide the whole experience when we use the internet.

But why do we layer?

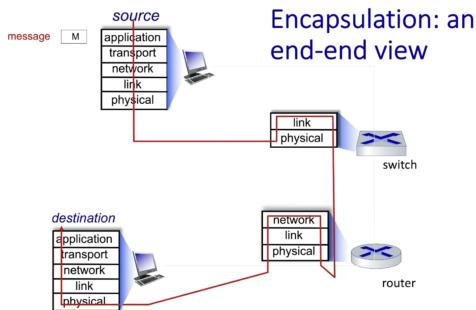
- Explicit structure allows identification and relationship of system's pieces
- Modularisation eases maintenance, and the updating of the system

Remember *shotgun surgery* from COMP2511. That's what we want to avoid! **What layers does the internet have?**

- Application: supports network applications
- Transport: Process-process data transfer
- Network: routing of datagrams from source to destination
- Link: data transfer between neighboring network elements
- Physical: physically transfers the bits between the links

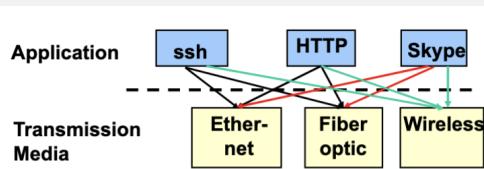
Definition. The application layer exchanges messages to implement some application service using services of the transport layer

As data exchanges from application layer to application layer, it of course first goes down the layers of one source. As the data passes down, headers are added at each layer (besides the physical); we call this *encapsulation*.



The importance of encapsulation.

Consider the alternative to layering. If there were no layering principle in networking, every single application would have to be reimplemented for every network technology.



Layering implements ways to pass any application through the same layer, by the usage of *headers*. This is the cost of layers.

Hosts, routers and switches; who implements what?.

Across the internet, there are three main participants: **hosts**, **routers** and **switches**. Who must implement what layers?

- Hosts
 - Hosts have applications. Application layer.
 - Applications must send segmented data. Transport layer.
 - Transport layer must send packets. Network layer.
 - Network layer must send bits. Passes to data link.
 - Data link passes to physical.
- So hosts must implement all layers.
- Routers
 - Bits arrive on wire to routers. Physical layer.
 - Routers have local delivery (forwarding) and need framing. Link layer.
 - Routers participate in routing. Network necessary.
 - Routers don't need to segment. No transport.
 - Routers don't have applications. No application.
- Switches
 - Bits arrive on wire to switches. Physical layer.
 - Switches pass packets within a LAN. Link layer.
 - Switches don't route. No network.
 - Switches don't need to segment. No transport.
 - Switches don't have applications. No application.

A. Client-server paradigm

Server:

- Always-on host
- Permanent IP address
- often in data centers, for scaling

Client:

- contact, communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do *not* communicate directly with each other

Examples of client-server protocols: HTTP, IMAP, FTP

In a client-server architecture, processes on each of the client and server's machines are communicating with each other. That is the client process initiates contact, and the server process responds.

Sockets.

- processes send and receive messages to and from its socket
- Sockets are like doors; sending sockets send processes out the door, and receiving through it.
- There must be a sending and receiving layer.

Addressing processes.

- to receive messages, processes must have identifiers
- identifiers must have an **IP address** and **port number**
- but, some port numbers are for specific services.

Required transport services.

- Data integrity; how much data reliability do we need?
- Timing; how fast does the data transfer have to be?
- Throughput; how much minimum bandwidth do we need?
- Security; does our data have to be secure?

Transport protocol services.

- TCP: reliable data, connection oriented, flow and congestion controlled
- UDP: unreliable data, a lot less controlled than TCP. These services can however be built ontop of UDP.

B. Peer-peer architecture

- No always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
- peers are intermittently connected, and have dynamic IP addresses

Example: P2P file sharing (torrents!)

C. Web and HTTP

A little bit about the web, and URLs.

The World Wide Web is a *distributed database* of pages linked through Hypertext Transport Protocol (HTTP). Webpages, are defined by Uniform Resource Locators (URL).

protocol://host[:port]/dir/res

- protocol: http, ftp, https, etc.
- host: DNS name, IP address
- port: default to protocol's stand port (e.g http: 80)
- dir: hierarchical directory, reflecting file system
- res: resource in directory dir client wishes to access

HTTP: hypertext transfer protocol.

- Web's application layer protocol
- Applies the client-server paradigm
 - client: browser that requests, receives then displays Web objects
 - server: web server sends (using HTTP protocol) objects
- HTTP uses the TCP protocol
 - client initiates TCP connection to server, using port 80
 - server accepts TCP connection
 - HTTP messages are then transacted
 - TCP connection is closed
 - HTTP is stateless - server maintains no information about past requests

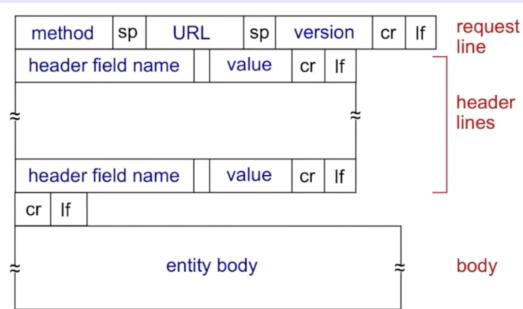
There are three flavours of HTTP connections

- Non-persistent*: : TCP connection opened, at most one object is sent, and then TCP connection is closed.
- Persistent*: : TCP connection opened, multiple object can be sent over single TCP connection, and then TCP connection is closed.
- Persistent with pipelining*: : TCP connection opened, multiple objects are requested concurrently over a single TCP connection, and then TCP connection is closed.

Round trip time (RTT): time for a small packet to travel from the client and back

By leaving the connection open, persistent connections can save RTTs for future messages, as a new 'acknowledgement' request is not required for the TCP connection - cutting the RTTs required by half.

Anatomy of a HTTP request message.



Do note that:

- The body is text, so each character uses a byte.
- That means, 256 uses 3 bytes.

The four main types of HTTP requests:

- GET: for sending data to server
- POST: for user input
- PUT: to upload new files to the server
- HEAD: request headers that would be returned if a GET request was sent instead

D. Cookies

Cookies. Web sites and client browsers use cookies to maintain some state between transactions. There are four components to use cookies

- cookie header line of HTTP response message (sent to client)
- cookie header line in next HTTP request message (sent to server from client)
- cookie file kept on user's host, managed by user's browser (state)
- back-end database at website

User states are of course extremely useful, and can be used for:

- authorisation
- shopping carts
- recommendations
- user session state

E. HTTP Performance Improvements

Web caches/Proxy Servers.

We'd like to limit repeat requests where possible, and to avoid going all the way to the host server.

- User configures browser to point to a local web cache.
- Browser sends all HTTP requests to the cache
 - if the objects in the cache, cache returns object to client
 - else, cache requests object to the origin server, and then forwards it along to client.

Essentially, proxy servers are a middle man between the client and server, that hopes to return cached information.

The origin server would tell the cache how long to cache the data for. The cache serves both the server and the client role.

Conditional GET.

- Web proxy/cache asks server whether or not the client-requested object has been updated (w.r.t If-modified-since or ETag)
- If it has been updated, then server transmits new version of object
- Otherwise, it returns the cached object

ETags and Conditional GET.

ETags, are essentially identifiers for a web object. Conditional GETs can also utilise ETags to check whether an object of the ETag x, exists, and if it does not exist, to fetch a new copy from the server.

HTTP/1.1 and HTTP/2.

HTTP/1.1 introduced multiple, pipelined GETs over a single TCP connection, which allowed for concurrent HTTP requests.

- The server responds to requests *in-order*
- Since it is first come, first serve, smaller objects that could be processed might wait behind large objects (*head-of-line (HOL) blocking*)
- Loss recovery (due to Go-Back-N) stalls object transmission.

HTTP/2 aims to decrease delay in multi-object HTTP requests. There is increased flexibility at **server** in sending objects to clients

- methods, status codes, etc. are unchanged
- transmission order of requested objects can be based on client-specific priority
- push unrequested objects to clients
- divide objects into frames, schedule frames to mitigate HOL blocking - so that large objects doesn't necessarily block small objects in queue.

F. E-mail

Three major components for email

- User agents (us)
- Mail servers (which send the emails around)
- Simple mail transfer protocol (the protocol used to send emails)

User agents, mail servers and SMTP protocol.

User agents

- Compose, edit and read mail messages
- Outgoing and incoming messages which are stored on the mail servers

Mail servers

- Mailbox - which contains incoming messages for user
- Message queue - which contains outgoing messages from a user

SMTP protocol

- Client: sending mail server
- Server: receiving mail server

Thus, for e-mails, there is server-to-server requests which are then relayed to clients.

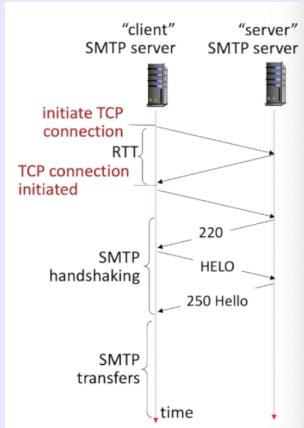
Alice sends an email to Bob.

- 1) Alice uses her user agent (for example, outlook) to compose an email message to

- bob@someschool.edu.
- 2) Alice's user agent sends the message to her mail server using SMTP or HTTP, and the message is placed in the message queue.
 - Whether the user agent uses SMTP or HTTP depends on the user agent
 - 3) Alice's mail server opens a TCP connection with Bob's mail server
 - 4) The mail server then sends Alice's message over the TCP connection
 - 5) Bob's mail server places the message in Bob's inbox, which he can then read.

SMTP RFC (5321).

- uses TCP to reliably transfer email messages from client to server
- **uses port 25**
- Of course, there is the usual SYN, SYN-ACK then ACK to establish the TCP connection.
- Three phases of transfer for SMTP
 - An SMTP handshake (which establishes and SMTP connection)
 - SMTP transfer of messages
 - SMTP closure
- Uses persistent connections



Mail message format.

Header

- To:
- From:
- Subject:

A blank line

Body: the "message", 7-bit ASCII

But how are the emails actually accessed from a user agent? There is of course a protocol for this too!

Internet Mail Access Protocol (IMAP).

IMAP defines how users interact with mail servers to access email messages. Commands are sent using

a TCP connection using ASCII commands, such as

- LOGIN username password to authenticate a user.
- SELECT INBOX to open the inbox.
- FETCH 1 BODY[] to retrieve message 1.
- SEARCH SUBJECT "Meeting" to search emails with "Meeting" in the Subject/
- LOGOUT to close the session.

G. Domain Name System (DNS)

DNS and its function.

Hosts and routers will have at least two identifiers:

- 1) A human readable name (such as google.com.au)
- 2) IP address (32 bit) - used to address datagrams (packets)

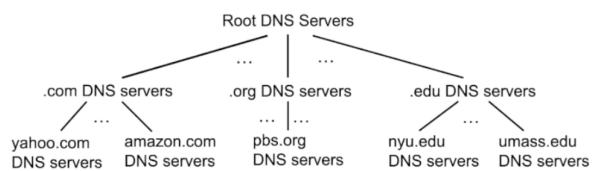
But how do we translate between addresses and names? We use DNS!

DNS is a distributed, hierarchical database implemented in hierarchy of many *name servers*. It's also an application-layer protocol, implemented in the network edge. It also has the following functions

- host aliasing - aliased names can also be translated.
- mail server aliasing - IP address of a mail server, associated with a domain
- load distribution - many IP addresses may be allocated to one name, which the DNS can distribute

So why distributed and not centralised?

- A central DNS server would be a single point of failure could bring down a network
- Intense traffic volume
- Distance from the server
- Maintenance
- Lack of scalability



If a client wants an address at www.amazon.com:

- 1) client requires root server to find .com DNS server
- 2) queries .com server to find amazon.com server
- 3) queries amazon.com to get IP address

Root DNS servers.

Official, contact-of-last-resort by name servers that can not resolve names. They forward clients to

proper name servers.

There are 13 logical root name servers world wide, but each server is replicated many times in each region.

Layers of the DNS tree.

- **Root name servers:** has information about TLD DNS servers.
- **Top-Level Domain (TLD) servers:** responsible for .com, .org, .cn, etc.
- **Authoritative DNS servers:** organisation's own DNS servers, providing authoritative hostname to IP mappings for organisations named hosts
- **Local DNS servers:** When a host makes a DNS query, it is sent to its local DNS query, local DNS server returns reply answering:
 - recent name-to-address translation pairs (which might be out of date)
 - forwarding request to the DNS hierarchy above

There are two types of querying:

- 1) Iterative, where the local DNS server sequentially requests down the DNS hierarchy, until it finds the address
- 2) Recursive, where the requests are placed on the sequential servers (for example, root goes to TLD directly, rather than going back to local).

Caching DNS information.

- Once any **server** or **host** learns of DNS information regarding name servers, they cache this information.
- There is a TTL (time to live) on these caches. This means that they are not utilised forever.
- Thereby once a DNS request is completed, they will not be required to be repeated in the near future.

DNS records. DNS servers store records in the form resource records

(name, value, type, ttl)

Depending on the type, these keys hold different values. ttl refers to Time to Live, which holds how long the record should be held in cache before being discarded.

Common types

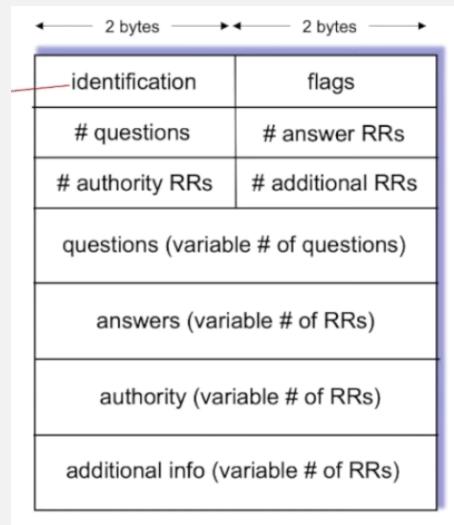
- A: name is hostname, value is IP address
- NS: name is domain (e.g. foo.com), value is hostname of authoritative name server
- CNAME: name is an alias, value is canonical name

- MX: value is the name of the mailserver associated with name

Getting your info into the DNS. example: new startup 'Network Utopia'

- register name networkutopia.com at DNS registrars
 - provide names, IP addresses of authoritative name server
 - registrar inserts NS, A RRs into .com TLD server
- create authoritative server locally

DNS protocol messages. DNS query and reply messages both have the same format.



- identification header is the same for reply and query, to identify which reply belongs to which query
- flag contains which type of query/reply it is
- number of questions and answers

DNS Security.

DDoS attacks

- bombard root servers with traffic
 - traffic filtering
- bombard TLD servers

Re-direct attacks

- man-in-middle: intercept DNS queries
- DNS poisoning: send bogus replies to DNS server, which caches
 - preventable by only caching from authoritative name servers.

Exploit DNS for DDoS

- send queries with spoofed source address: target IP
- requires amplification

H. Peer-to-peer applications

Peer-to-peer architecture.

- no always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
- peers are intermittently connected and change IP addresses

Why peer-to-peer?

How much time to distribute file (size F) from one server to N peers? where μ_s is the bandwidth of the server, and d_i is the individual download rates of the clients, we have the throughput to be

$$\text{download time} \geq \max \left\{ \frac{NF}{\mu_s}, \frac{F}{d_{min}} \right\}$$

Note that this grows linearly with N . In the peer-to-peer model, each client must download a file copy $\frac{F}{d_{min}}$, but as more clients begin to share bandwidth, we have that

$$\text{download time} \geq \max \left\{ \frac{F}{\mu_s}, \frac{F}{d_{min}}, \frac{NF}{\mu_s + \sum_{u_i}} \right\}$$

where u_i is the upload speed of the clients. So more clients, lower download time. $\frac{NF}{\mu_s}$ is the initial time for all the clients to download the file.

BitTorrent.

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

A *torrent* is a group of peers exchanging chunks of a file. A *tracker* tracks peers participating in a torrent. So it's not a pure peer-to-peer architecture, as there is an always-on server.

If a new user wants to join a torrent, then the user contacts the tracker. The tracker gives the list of peers in the torrent. They initially have no chunks, but as they accumulate them, they also begin to upload them.

BitTorrent: requesting and sending file chunks.

Consider two participants, *Alice* and *Bob*.

- Requesting chunks
 - at any given time, different peers have different subsets of file chunks.
 - periodically, Alice asks each peer for a list of chunks they have.
 - Alice requests missing chunks, *rarest first*, to ensure these chunks have copies.
- Sending chunks: *tit for tat*
 - Alice sends chunks to those four peers cur-

ently sending her chunks at **highest rate**.

- * other peers are choked by Alice (Alice does not send them chunks)
- * re-evaluate top 4 every 10 seconds
- Every 30 seconds, Alice randomly chooses another peer, and starts sending chunks
 - * optimistically unchoke this peer
 - * new chosen peer may join top 4

I. Video streaming and content distribution networks (CDNs)

Videos.

- video: sequence of images displayed at a constant rate
- digital image: an array of examples
- coding: use redundancy within and between images to decrease # bits used to encode image

Two main encodings

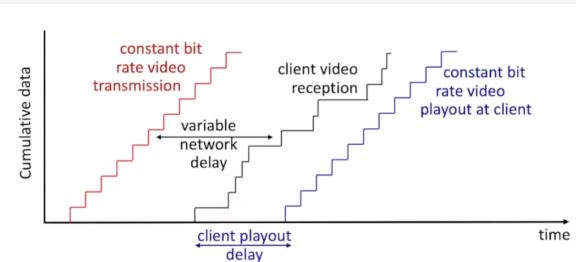
- CBR (constant bit rate): video encoding rate fixed
- VBR (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes.

But the amount of available bandwidth will vary over time, and we'll have to adapt to that. We also have to deal with packet loss - delay due to congestion will delay (causing buffering).

Challenges on the client side.

- continuous playout constraint: during client video playout, playout timing must match original timing.
 - but network delays are variable (jitter), so we'll need a **buffer** to match continuous playout constraints

So how is the continuous playout constraint solved?



Dynamic, Adaptive Streaming over HTTP (DASH).

Server:

- divides video file into multiple chunks
- each chunk encoded at different rates
- manifest file: provides URLs for different

chunks

Client:

- periodically estimate server-to-client bandwidth
- consults **manifest file**:
 - chooses maximum coding rate sustainable at current bandwidth
 - can choose different coding rates for different points in time

But how do we stream all of this content to hundreds of thousands of simultaneous viewers? One option would be a *single, large "mega-server"*. Of course, this suffers from congestion, single point of failure, as well as distance issues to some clients.

Content distribution networks (CDN). Content Distribution Networks (CDNs), store/serve multiple copies of videos at multiple geographically distributed sites. There are two types of CDNs

- 1) enter deep: push CDN servers deep into many access networks (at the network edge)
- 2) bring home: smaller number of CDN servers near access nets (deeper into the network)

So how is a video streamed using CDNs?

- Let's say Bob wishes to stream a video on Netflix
- 1) Bob selects the video, which in turn creates a DNS request to find the CDN server.
 - 2) Then, we request the CDN server the DNS query returned for a **manifest** file.
 - This begins the process of video streaming
 - we now have the URLs for the chunks of video.
 - 3) If at any point, the CDN server becomes congested - it may redirect Bob to a new CDN server.

III. Transport Layer

The transport layer is all about how we segment and prepare data for delivery.

- Providing logical communication between application processes running on different hosts
- Transport protocols actions in end systems (on the network edge)
- Two transport protocols available to internet apps; TCP and UDP

The distinction between the *network layer* and *transport layer* - is that the network layer provides logical communication between *hosts* - whereas the transport layer provides logical communication between *processes*.

A. Transport layer multiplexing and demultiplexing

Multiplexing and demultiplexing.

Multiplexing and demultiplexing is how the transport layer keeps track of which segments of data belong to which application (sockets), by using port numbers contained in the header.

These headers are added (and interpreted) at the transport layer, and contain the source and destination ports.

How demultiplexing works.

- 1) the host receives IP datagrams
- 2) each datagram has source IP address, destination IP address
- 3) each datagram carries one transport layer segment
- 4) each segment has source, destination port number
- 5) the host uses IP addresses and port numbers to direct segment into socket

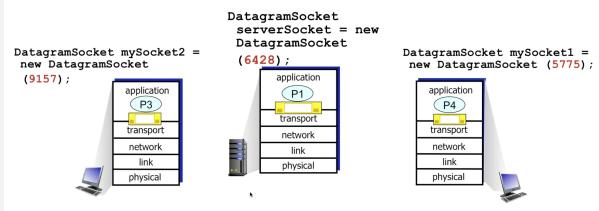
UDP Connectionless Demultiplexing.

When creating a UDP socket, you have to specify a host-local port number. When creating a datagram to send into a UDP socket, you must specify:

- destination IP address
- destination port number

When the receiving host receives UDP segment, it uses the destination port number in the segment into the respective socket.

UDP datagrams with the same destination port number but with different source IP addresses and port numbers will be directed to the same socket.



Both `mySocket1` and `mySocket2`'s datagrams will be sent to the same socket, on port 6428 on `serverSocket`.

TCP Connection-oriented demultiplexing.

TCP sockets use 4-tuples:

- source IP address
- source port number
- destination IP address

- destination port number

Since TCP sockets have a distinct connection between sender and receiver, it requires to know all four of the above values to send it to the respective sockets.

Therefore, if many processes use the same port; under a TCP connection, we can ensure that packets can be sent to specific sockets held by processes on the server.

The welcoming socket.

In TCP connections, a handshake is required to establish the connection. This handshake is done with a **welcoming socket** at the server, which is separate to the actual socket that the client will connect to.

Its main purpose is to listen for incoming connections.

B. Connectionless transport: UDP

UDP: User Datagram Protocol.

- connectionless; no handshakes with sender/receiver
- UDP segments are handled independently
- no connection establishment - saves an RTT
- simple, smaller header size
- no congestion control (so no intrinsic buffering of requests)

Thus, UDP is used for data that needs to be transferred quickly, where packet loss isn't critical; like media streaming.

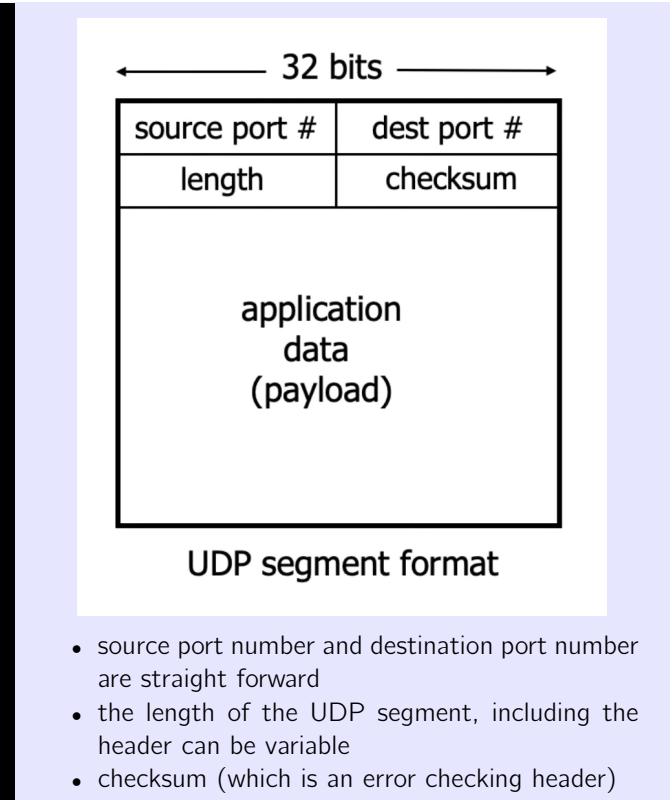
On the sender side

- A sender passes an application-layer message
- UDP segment header field values are identified
- UDP segment is created, and sent to IP

On the receiver side

- receive segment from IP
- checks UDP checksum header value
- extracts application layer message
- demultiplex message up to application socket

UDP segment.



UDP segment format

- source port number and destination port number are straight forward
- the length of the UDP segment, including the header can be variable
- checksum (which is an error checking header)

Checksum.

UDP checksum is aimed at detecting errors (or flipped bits) in a transmitted segment. Checksum works by sending numbers, and the sum of these numbers. If the sum of these numbers don't equal to the transmitted sum, then there has been an error while transmitting.

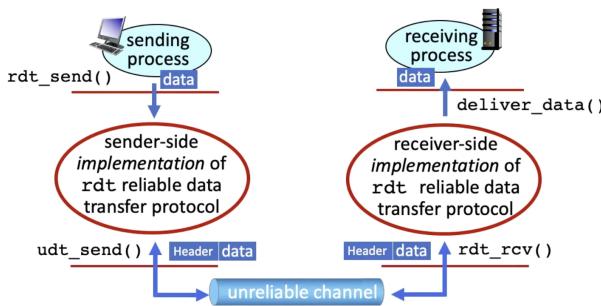
wraparound	1	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1
	1	1	0	1	1	0	1	1	1	0	1	1
	sum	1	0	1	1	1	0	1	1	1	1	0
	checksum	0	1	0	0	1	0	0	0	1	0	0

- Begin by doing a normal bit addition - if there is a wrap around, add it to the back once more
- The check sum is then the complement of this final sum

Do note that checksum isn't a perfect error check - you can convince yourself of this by considering flipping the first two bits of the two numbers in the example. The checksum would still be the same!

C. Principles of Reliable Data Transfer

How do we reliably send packets from a sender to receiver - ensuring they are sent properly? This mechanism is of course implemented in the transport layer, over an *unreliable channel*.



So how do we develop a **Reliable Data Transfer** (rdt) protocol?

- consider only unidirectional data transfer
 - but control info will flow on both directions
- use finite state machines (FSM) to specify sender, receiver

Let us try to make finite state machines for differing levels of contingencies.

rdt1.0: reliable transfer over a reliable channel.

Since the underlying channel is reliable, the sender and receiver don't have to do much besides send and receive the data. They can be sure that the packets are sent and received.

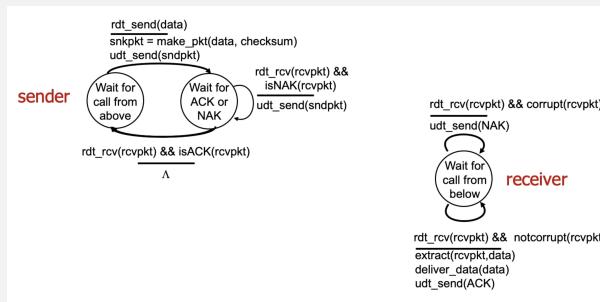


rdt2.0: channel with bit errors.

Now, the underlying channel is unreliable and may flip bits in packets. We can use checksum to detect bit errors. How do we recover from errors?

- acknowledgements (ACKs): receiver explicitly tells sender that packet received is OK
- negative acknowledgements (NAKs): receiver explicitly tells sender that packet had errors
- sender retransmits the packet on a receipt of a NAK

Now the sender sends one packet, then waits for the receiver's response.



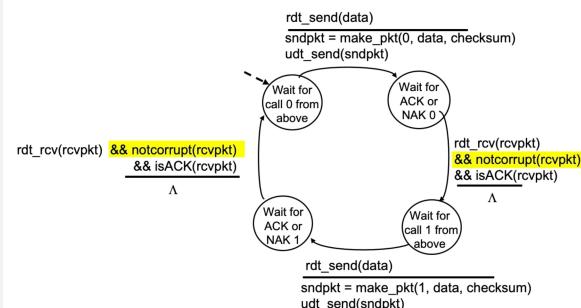
There should be one obvious question that arises.

What happens if the acknowledgements are corrupted or dropped?

rdt2.1: handling corrupted ACKS and NAKS.

Here's how rdt2.0 can be extended to deal with corrupted acknowledgements

- sender retransmits current packet if ACK/NAK is corrupted
- sender adds sequence number to each packet (so same seq. number for same packet)
- receiver discards duplicate packet



rdt2.2: NAK-free protocol.

Same functionality as rdt2.1, but implemented using ACKs only.

- Instead of a NAK, receive sends an ACK of the last packet # received.
- Therefore, for the sender, a duplicate ACK indicates a loss at # + 1.
- TCP uses this NAK free approach.

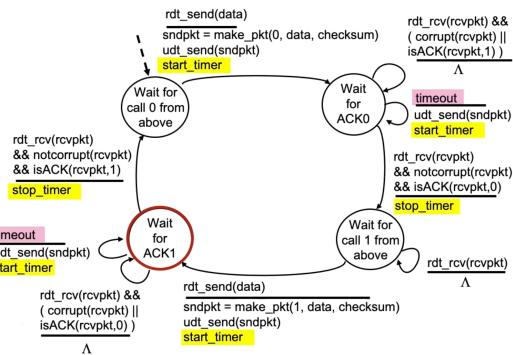
rdt3.0: channels with errors and loss.

In reality, channels can also lose packets.

- Checksum, sequence #s, and etc can be useful, but they won't solve this problem.
- What do humans do? If someone randomly stopped talking, we would ask them to acknowledge what we said.
- So the **motivation** is to wait a 'reasonable' amount of time for an ACK to come back, and if it doesn't, to take appropriate action.

Approach:

- 1) sender waits a reasonable time for an ACK
- 2) retransmit if no ACK is received in this time
- 3) if packet (or ACK) delayed, but not lost:
 - a) retransmission will be a duplicate, but the seq #'s can deal with this
 - b) receivers must specify sequence # of packet being ACKed
- 4) use a countdown timer to interrupt after a reasonable amount of time



Utilisation.

Utilisation can be described by

- 1) How much of the available bandwidth is being used?
- 2) How much time in an RTT is spent sending data?

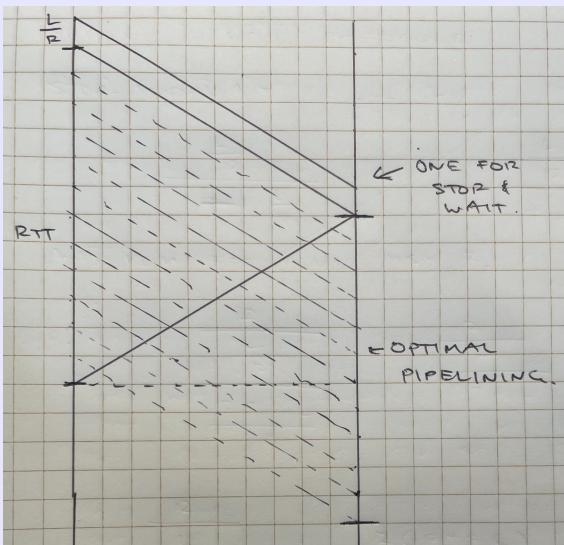
The first can just be found by

$$\frac{\text{total bandwidth used}}{\text{total bandwidth available}}$$

Whereas the second one can be found by

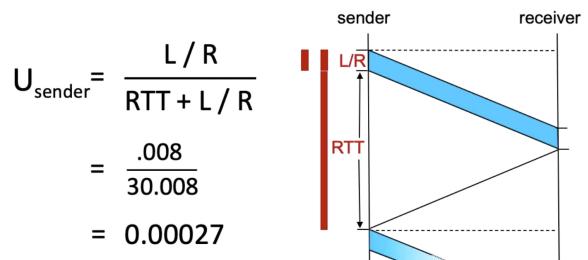
$$\frac{NL}{R} + RTT$$

where N is the size of pipelining window, L is the size of packets, R is the transmission rate of the link and RTT is propagation delay RTT.



Performance of rdt3.0:

- U_{sender} : utilisation
- 1 Gbps link, 15 ms prop. delay, 8000 bit packet
- $D_{\text{trans}} = L/R = 8000/10^9 = 8 \text{ microseconds}$



That's very low utilisation. To increase the utilisation of the link, we can allow multiple packets to be sent concurrently while waiting for their acknowledgements.

D. Pipelining for reliable data transfer protocols

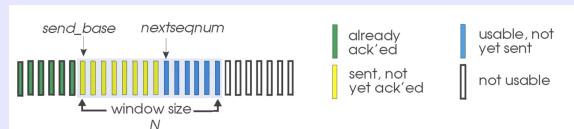
Components of a window.

- SendBase/RecvBase: the sequence number of the oldest unacknowledged packet
- nextSeqNum: the sequence number of the next packet
- CongWin/cwnd: the congestion window (congestion control)
- rwnd: the receiver window (flow control)

Go-Back-N.

Sending side

- Has a window of N transmitted packets, that are unacknowledged
- Has a k-bit seq # in the packet header
- *cumulative ACK*: when an ACK of seq # n is received, **all** of the packets until n will be acknowledged.
- There is a timer for the oldest in-flight packet.
 - if the receiver receives out of order, the packet is discarded and acknowledgment of the most recent in order packet is sent
 - if packet k out of n packets is timed out, then packets $k \rightarrow n$ must be resent (this is the cumulative idea).



Receiving side

- ACK-only: always sends ACK for correctly-received packet so far, with *highest* in-order seq #.
 - may generate duplicate ACKs
 - need only remember `rcv_base`, or next in-order seq #.
- on receipt of out-of-order packet:
 - discard or put in buffer
 - send ACK for highest seq # received so far

Convincing myself of Go-Back-N.

Consider sending four packets 0, 1, 2, 3 to a server. The server acknowledges all of these packets, but the acknowledgements for 0, 1 and 2 are lost. The acknowledgement for 3 arrives - and in Go-Back-N, we say **all of these packets are sent**.

We can do this as we know the server would *not* send an acknowledgement for 3, if it had not already received 0, 1 and 2.

Selective repeat.

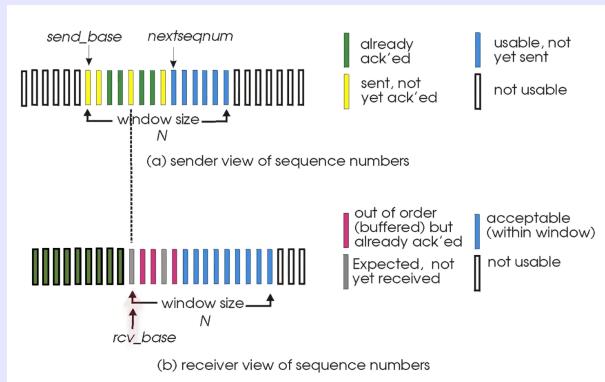
Receiving side

- receiver *individually* acknowledges all correctly received packets
- so packets must be buffered as needed, to guarantee in order delivery

Sending side

- sender times-out/retransmits individually for unacknowledged packets
- so the sender must time every unacknowledged packet, as there is no cumulative idea like in Go-Back-N
- The window is only advanced when the last packet to be received is acknowledged.

Sliding window defined by the smallest acknowledged k -th packet, up till the $k + n + 1$ -th packet, which defines the maximum amount of packets allowed to be sent at one time.



Maximum window sizes for Go-Back-N and Selective Repeat.

For window size W and number of possible sequence number $|S|$

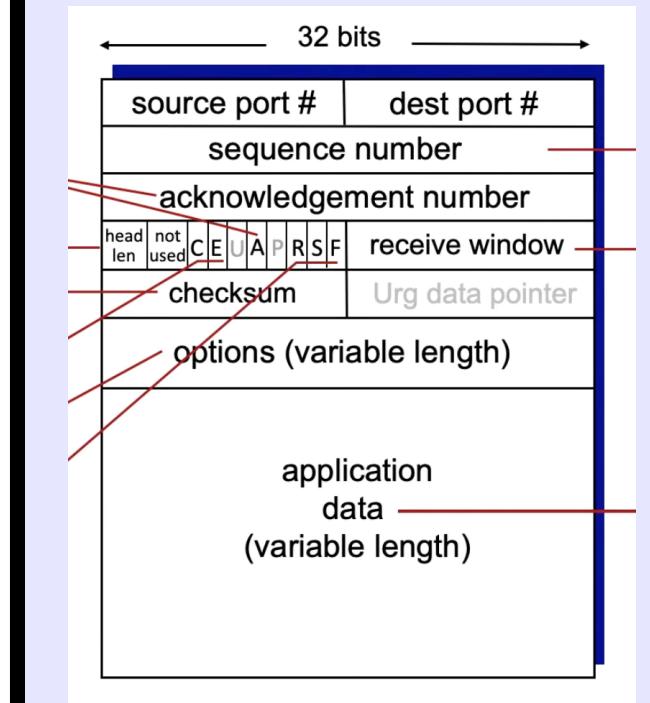
- For Go-Back-N, $W < |S|$.
 - For $[0, 1, 2, 3]$, if all ACKs are lost, then 0 will be sent as a resubmission, but the receiver believes it is a new packet.
- For Selective Repeat, $W \leq \frac{|S|}{2}$.
 - If W packets are lost, with $|S| \leq 2|W|$, retransmissions are interpreted as new data.

- The window may wrap back around to ACK lost packets - which causes confusion.

E. TCP Reliability

TCP Overview.

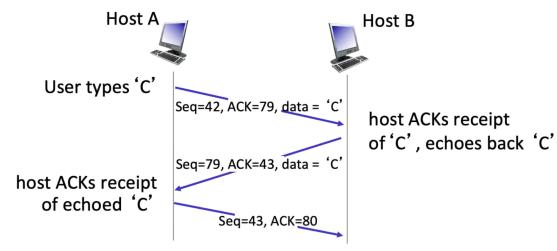
- point-to-point: one sender, one receiver
- reliable, in-order *byte stream*: no "message boundaries"
- full duplex data: bi-directional data flow
- cumulative ACKs: so uses Go-Back-N
- pipelining: tcp congestion and flow control set window size
- connection-oriented: so a handshake occurs to initialise sender-receiver state before data exchange
- flow controlled: sender will not overwhelm receiver



TCP Sequence Numbers.

A *byte stream* is simply a sequence of bytes - that is TCP sends *byte/stream-oriented* data that can be read by the receiver. UDP uses *message-oriented* which means when one message is sent, it will be sent as one packet.

- Sequence numbers: represents byte stream 'number' of first byte in segment's data.
- Acknowledgements: represents the seq # of the next byte expected from the other side
- What to do with out of order segments? TCP doesn't give a standard reasoning, so is up to the implementation



Note in the above statement, that the acknowledgement statement corresponds to the next byte that the *receiver* should **expect** from the sender.

SYNs, FINs and payload-less packets in sequence numbers.

- SYNs, SYNACKs and FINs will use up a sequence number, even though there is no data sent within these packets.
- Otherwise, ACKs with no data do not use up a sequence number.
 - This doesn't mean an ACK with no data always uses the previous sequence number.
 - If the previous sequence number used (e.g. FIN at 4777), then the next ACK should be at 4778.
 - If there were subsequent payload-less ACKs, they would all be at 4778.

TCP Maximum Segment Size.

- IP Packet (the packet actually sent across)
 - No bigger than Maximum Transmission Unit (MTU) of link layer
 - E.g up to 1500 bytes with Ethernet
- TCP packet
 - IP packet with a TCP header and data inside
 - TCP header \geq 20 bytes long
- TCP **segment**
 - No more than Maximum Segment Size (MSS) bytes
 - E.g up to 1460 consecutive bytes from the stream
 - MSS = MTU - 20 (Min. IP Header) - 20 (min. TCP header)

TCP Acknowledgements, Sequence Numbers and Piggybacking.

- Sequence numbers in the TCP protocol begin with a random Initial Sequence Number (ISN).
 - This prevents malicious behaviour, as well as confusion for back-to-back connections.
- ACKs returns the **next sequence number** the receiver expects.

- Many TCP connections have duplex (double-sided) sending behaviour
 - To reduce delay, ACKs can be sent alongside payload
 - This combines two messages into one

TCP Go-Back-N + Selective Repeat.

TCP combines the ideas of Go-Back-N and Selective Repeat

- TCP will *typically* use cumulative acknowledgements like Go-Back-N
- TCP also uses a sliding window to control how much unACKed data can be "in flight"
- However, TCP also **buffers out-of-order segments**, using Selective Acknowledgements SACK.
- This means the sender can retransmit only the lost packet, while combining the ideas of cumulative ACK.

TCP round trip time, timeout.

How do we set the timeout value? Clearly these should depend on the round trip time, but if we do this too fast - we may do unnecessary transmissions, or too long, and react too slow.

$$RTT_{estimate} = (1 - \alpha) \cdot RTT_{estimate} + \alpha \cdot RTT_{new}$$

Where $\alpha \in [0, 1]$. A standard value for $\alpha = 0.125$. The equation above denotes an exponential weighted moving average, where new samples are given more weight over older ones.

Now to find the actual timeout interval, we use

$$\text{Timeout} = RTT_{estimate} + 4 \cdot \text{dev}(RTT_{estimate})$$

And to calculate the deviation

$$\text{DevRTT} = (1 - \beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

TCP Sender to Receiver Flow.

- 1) create segment with seq #
- 2) seq # is byte-stream number of first data byte in segment
- 3) start timer if not already running
 - think of timer as for older unACKed segment
 - expiration interval: TimeOutInterval

Receiver Scenarios

- 1) arrival of in-order segment with expected # arrives. all data up to expected seq # already acked.
 - a) wait up to 500ms for another segment.
 - b) if no next segment, send ACK.
 - c) if another segment, then send a cumulative ACK.

- 2) arrival of in-order segment with expected seq #. one other segment has ACK pending
 - a) immediately send single cumulative ACK, acking both in-order segments.
- 3) arrival of out-of-order segment, higher than expected seq #
 - a) immediately send duplicate ACK
 - b) ACK should contain # of next expected byte
- 4) arrival of segment that partially or completely fills gap
 - a) immediately send ACK, provided that segment starts at lower end of gap.

TCP fast retransmit.

If multiple consecutive ACKs from the receiver come back with the same sequence number, the sender can know that there has been a missing segment.

- 1) If there is a receipt of three duplicate ACKs, this indicates 3 segments have been received after a missing segment
- 2) So retransmit the lost segment (as detailed by ACK #) immediately.

F. TCP Flow Control

What happens if network layer delivers data faster than application layer removes data from socket buffers? this is what flow control tries to achieve - as buffered data at the application layer isn't very helpful.

The basic idea.

- TCP receivers 'advertised' free buffer space in `rwnd` field in TCP header item sender limits amount of unAcked ('in-flight') data to received `rwnd`.
- guarantees receive buffer will not overflow.

G. TCP connection management

TCP Connection Management.

Before exchanging data, sender and receiver must 'handshake'

- agree to establish connection
- agree on connection parameters (e.g starting seq #'s, ...)

Why use a 3-way handshake?

- 1) TCP is fully bidirectional - which means both sides can send and receive
- 2) This means both sides need to establish an initial sequence number.
- 3) The initial request alerts the initiated of the ISN of the initiator.

- 4) The acknowledgment of the initial request alerts the initiator of the initiated's ISN, and acknowledges the ISN of the initiator.
- 5) The final acknowledgment acknowledges the initiated's ISN.

The below is the detailed process of the 3-way handshake

- 1) The client sends the server a SYN message, with `SYNbit=1` and `seq=x`
- 2) The server sends the client a SYNACK message, with `seq=y`, `ACKbit=1` and `ACKnum=x+1`
- 3) The client sends the server a ACK message, with `ACKbit=1` and `ACKnum=y+1`.

How to close a TCP connection?

- 1) Closer sends packet with FIN bit = 1
- 2) Closee responds with ACK bit = 1 and FIN bit = 1. These may be separated into two different messages.
- 3) Closer responds with ACK bit = 1. Closee is now closed - closer remains open for any remaining packets from the closee.

Abrupt termination

- 1) Rather than a FIN, an abrupt termination is declared by the closer sending a RST packet.
- 2) The closee does not respond with an ACK - so it is not sent reliably.
- 3) Rather, if the closee sends another packet to A, another RST is sent.
- 4) Occurs due to **unexpected and invalid packets** as well as network or system loss.

TCP SYN Attacks and Cookies.

SYN Attacks

- An attacker creates a fake SYN packet, with the IP address of victim host
- Victim receives TCP connection, allocates buffers, creates variables, etc.
- This uses up resources on the victim's machine.
- The attacker never ACKs; yet even during this timeout, the victim has used resources.
- The attacker sends multiple of these ACKs, overwhelming the victim.

SYN Cookies

- On receipt of SYN, server does not create a connection state.
- Creation of ISN, that is a hash of source & dest IP address, and port number of SYN packet.
 - Responds with SYN ACK containing ISN.
 - The server need not store this.
- If original SYN was genuine, then a connection state will be instantiated if the ACK returns ISN + 1
- Otherwise, a fake SYN, and no state is created.

H. Congestion control

Problem statement.

Too many sources sending too much data too fast for the **network** to handle.

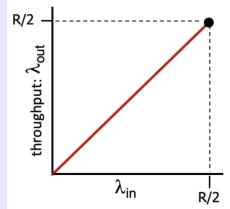
How congestion occurs.

Consider two hosts sending through a shared link of capacity R to a router R , which links to two separate servers of shared link R . The two hosts send to separate servers.

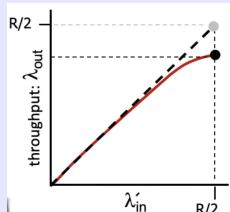
- λ_{in} : the arrival rate of data from a host
- λ_{out} : the throughput that the server receives

With buffers at R , and queuing delays as $\lambda_{in} \rightarrow \frac{R}{2}$, we may see retransmissions from the hosts to account for buffer overflows. Call this adjusted arrival rate λ'_{in} .

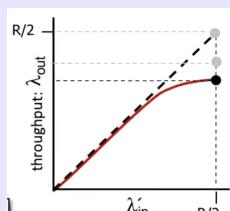
Imagine that somehow the sender knows how much space is available in the buffer, and thus no packets are dropped. In this case, $\lambda_{in} = \lambda_{out}$.



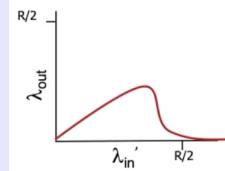
Now imagine the sender can perfectly know when a packet is dropped by the buffer. In this case, $\lambda'_{in} \geq \lambda_{out}$, as packets are dropped and do not reach the servers.



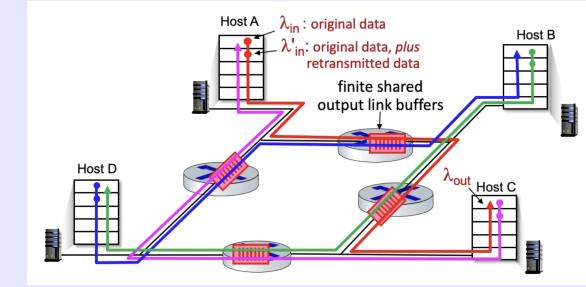
Now imagine the sender *does not* know whether a packet has been dropped or not, which require retransmissions. However, sender times can time out prematurely, sending two copies, both of which are delivered.



As we add more hops inbetween our hosts and servers, any upstream transmission capacity and buffering that was used for the packet gets wasted - and so congestion becomes worse and worse.



We call the above phenomena **congestion collapse**.



So how do we deal with this?

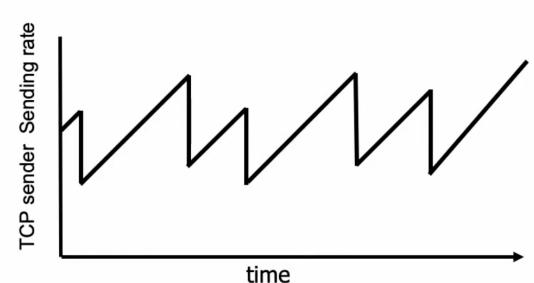
- 1) **End-end congestion control**: no explicit feedback from network, but infers congestion from loss and delay. This is the approach taken by original TCP.
- 2) **Network-assisted congestion control**: routers provide direct feedback to sending/receiving hosts with flows passing through congested routers. may indicate congestion or explicitly end rate.

I. TCP Congestion Control

Additive Increase Multiplicative Decrease (AIMD).

Approach: senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss events. Specifically:

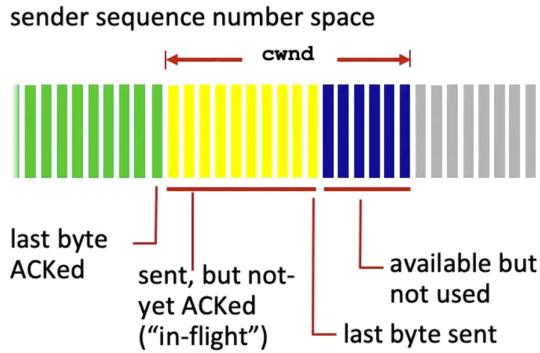
- **Additive increase**: increases rate by 1 maximum segment size every RTT until loss detected
- **Multiplicative decrease**: cut sending rate in half at each loss event



Multiplicative decreases are signalled by:

- Cut in half on loss detected by triple duplicate ACK
- Cut to 1 maximum segment size when loss detected by timeout

cwnd (congestion window) and TCP.



- Recall $cwnd$ (congestion window), which is dynamically adjusted by senders.
- The TCP sender limits transmission such that:

$$\text{LastByteSent} - \text{LastByteAcked} \leq cwnd$$

- Then, TCP rate is given by

$$\text{TCP rate} = \frac{cwnd}{RTT} \text{ bytes/sec}$$

TCP slow start.

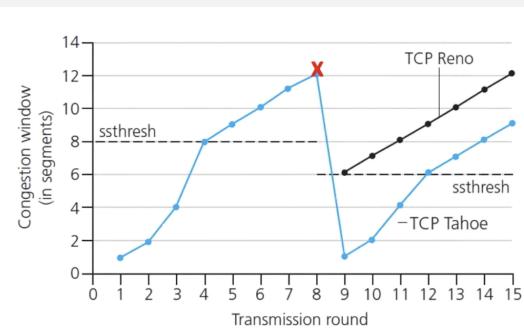
- When connection begins, increase rate exponentially until first loss event
 - initially $cwnd = 1$ MSS
 - double $cwnd$ every RTT
 - done by incrementing $cwnd$ for every ACK received
- so sending rate ramps up exponentially.

So how do we join these two ideas of AIMD and slow start together? More formally put:

When should we swap from exponential increasing sending rates to linear increasing?

Transitioning from slow start to AIMD.

We switch from slow start exponential growth to AIMD when $cwnd$ (or the amount of bytes that are able to be in flight) reaches half of its value before the last timeout.



A generic TCP congestion control algorithm.

$cwnd$: congestion window

$ssthresh$: slow start threshold

- If $cwnd < ssthresh$; then slow start (exponential increase).
 - If this is the beginning of the flow - slow start until congestion event
 - Otherwise, slow start until $ssthresh$.
- Else, additive increase (linear increase)
- If timeout occurs, then reduce $cwnd$ to x segments
- If 3 duplicate ACKs occur, then reduce $cwnd$ to y segments

Note x can equal y . The slow start threshold is set to half of the $cwnd$ value before the last timeout.

$$ssthresh = \frac{cwnd_{\text{before timeout}}}{2}$$

TCP Tahoe and Reno.

TCP Tahoe and Reno are different implementations of TCP congestion control.

TCP Tahoe.

- On timeout: $cwnd = 1$
- On triple duplicate ACK: $cwnd = 1$

TCP Reno.

- On timeout: $cwnd = 1$
- On triple duplicate ACK: $cwnd = \frac{cwnd}{2}$

Delay-based TCP congestion control.

TCP increases the sending rate until packet loss occurs at some router's output; we refer to this as the **bottleneck link**.

Consider the link can only send a specific amount of bits per second. Clearly there is no point of continually sending more bits through this link if it cannot handle the traffic.

We can measure the round trip time. We can also consider the # of bytes sent in the last RTT interval. The minimum RTT value, is the *uncongested path*. Consider:

$$T_{\text{measured}} = \frac{\# \text{ of bytes in last RTT interval}}{RTT_{\text{measured}}}$$

The uncongested throughput with congestion window $cwnd$ is then

$$T_{\text{ideal}} = \frac{cwnd}{RTT_{\text{min}}}$$

Thus

- If T_{measured} close to T_{ideal} , increase $cwnd$ linearly.

- If $T_{measured}$ far below T_{ideal} , decrease cwnd linearly.

Explicit congestion notification (ECN).

TCP deployments often implement *network-assisted* congestion control:

- two bits in IP header (TOS field) marked by network router to indicate congestion
- congestion indication carried to destination
- destination sets ECE bit on ACK segment to notify sender of congestion
- involves both IP (IP header ECN bit marking) and TCP (TCP header C,E bit marking)

TCP fairness.

Fairness goal: if K TCP sessions share some bottleneck link of bandwidth R , each should have average rate of R/K .

Consider AIMD.

- 1) If connection 1 starts with more throughput than connection 2, both linearly increase.
- 2) However, connection 1 will reach a sending rate of R much faster - which then leads to a halving of the sending rate
- 3) While this happens, connection 2 begins to grow, until it also halves.
- 4) Therefore, the two connections converge to fairness

IV. Network layer

The network layer has two main jobs - to forward datagrams within routers, and to **route end-to-end paths**.

Responsibilities of the network layer.

- transport segment from sending to receiving host
 - sender: encapsulates segments into datagrams, passes to link layer
 - receiver: delivers segments to transport layer protocol
- routers:
 - examines header fields in all IP datagrams passing through it
 - moves datagrams from input ports to output ports to transfer datagrams along end-end path

Two main segments in the network layer:

1) Data plane

- local, per-router function
- determines how datagram arriving on router input port is forwarded to router output port

2) Control plane

- network-wide logic
- determines how datagram is routed among routers along end-end paths
 - traditional routing algorithms: implemented in routers
 - software-defined networking (SDN): implemented in remote servers

Internet 'best effort' service model.

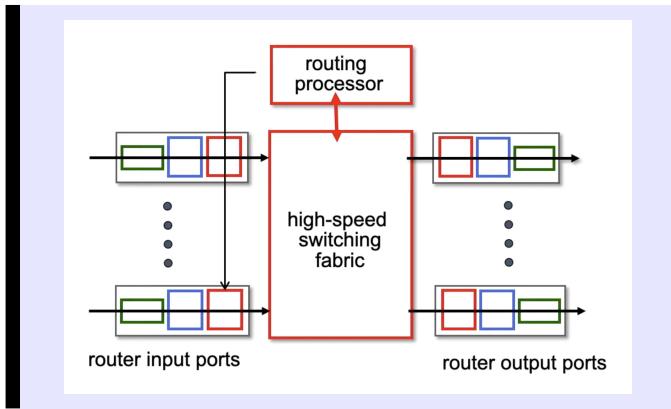
No guarantees on

- 1) successful datagram delivery to destination
- 2) timing or order of delivery
- 3) bandwidth available to end-end flow

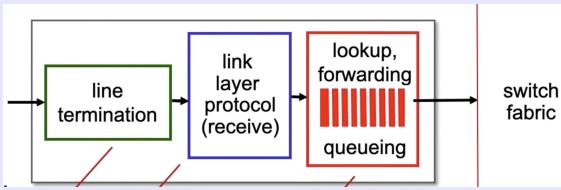
So... basically no guarantees at all.

A. What's inside a router?

High level diagram of router.



Input port functions.



- Line termination, is implemented by physical layer (datagrams passed through copper, for e.g)
- Link layer receives from the physical layer
- Lookup, forwarding and queueing is the *network* layer, which decides which port to output from.
 - **destination-based forwarding:** forward based only on destination IP address
 - **generalised forwarding:** forward based on any set of header field values

Functions of a router.

- Routers have *interfaces*, which connect routers to the physical link.
 - Each of these interfaces have their own IP addresses.
 - These interfaces cover a *range* of IP addresses (a subnet)
- Before sending through a interface, the router will then approach the link layer to find the next local hop (using MAC addresses).

B. Forwarding

Intro to destination-based forwarding.

Destinations are represented by a 32-bit number, and thus have 4+ billion possible destination addresses. We certainly can't have a table entry for every single unique destination. Rather, the addresses are aggregated into ranges.

forwarding table	
Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

But what happens if these ranges don't divvy nicely? For example, what if a certain range of the link interface 0 range should in fact go to range 3? We could add a new entry; but rather, we use something called *longest prefix matching*.

Longest prefix matching.

Motivation. Work with address prefixes, instead of ranges.

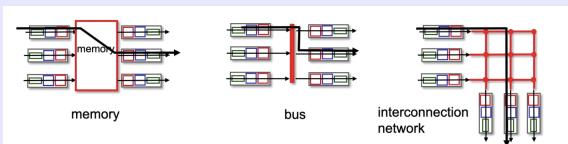
Destination Address Range	Link interface
11001000 00010111 00010*** *****	0
11001000 00010111 00011000 *****	1
11001000 00010111 00011*** *****	2
otherwise	3

When looking for forwarding table entry for given destination address, use **longest address** prefix that matches destination address.

- Longest prefix matching couples well with addressing.
- Longest prefix matching: often performed using ternary content addressable memories (TCAMs)

Switching fabrics.

- transfer packet from input link to appropriate output link
- **switching rate:** rate at which packets can be transferred from inputs to outputs
 - often measured as multiple of input/output line rate R
 - N inputs: switching rate NR desirable



Three types of switching:

- 1) Switching via memory:
 - Traditional computers with switching under direct control of GPU
 - packet copied to system's memory
 - speed limited by memory bandwidth
- 2) Switching via bus:

- Datagram from input port memory to output port memory via a shared bus
 - *bus contention*: switching speed limited by bus bandwidth
- 3) Switching via interconnection network
- Crossbar switches (n inputs to n outputs via n^2 switches)
 - **Multistage switch**: $n \times n$ switch from multiple stages of smaller switches
 - **Exploiting parallelism**:
 - Break up datagram into fixed length cells on entry
 - switch cells through the fabric, reassemble datagram on exit

C. Packet buffering and queueing

Input port queuing.

- If switch fabric slower than input ports combined, then queueing may occur at input queues
 - queueing delay and loss due to input buffer overflow
- **Head-of-the-Line (HOL) blocking**: queued datagram at front of queue prevents others in queue from moving forward

Output port queueing.

- If the switch fabric sends to an output port at a rate of NR , but the output port sends out packets at R , queueing and congestive packet loss can occur.
- **Drop policy**: which datagrams to drop if there is no buffer space?
- **Scheduling discipline**: chooses among queued datagrams for transmission

So how much buffering should a router have? Is more buffer space always better?

- RFC 3439 rule of thumb: average buffering equal to "typical" RTT (example 250ms) times output link capacity C
- More recent research recommends, with N flows:

$$\frac{RTT \cdot C}{\sqrt{N}}$$

- More buffer space means *more delays*, as more packets must be sent instead of dropping.
 - long RTTs: poor performance for realtim eapps
 - recall delay-based congestion control: "keep bottleneck link just full enough, but no fuller"

Buffer management.

First, abstract an output port like a queue.

- **drop**: which packet to add, drop when buffers are full

- tail drop: drop arriving packet
- priority: drop/remove on priority basis (for e.g, prioritise real time data, drop TCP end user datagrams)
- **marking**: which packets to mark to signal congestion (ECN, RED)

Packet scheduling.

Packet scheduling is the problem of deciding which packets should leave in which order.

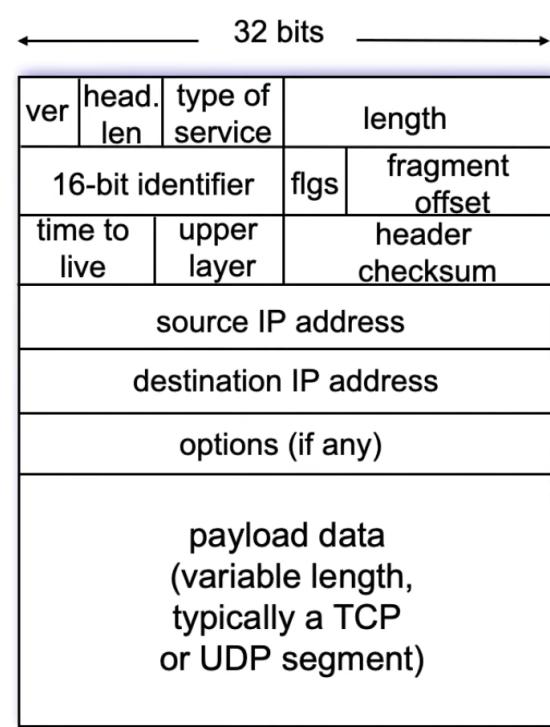
- 1) First come first serve (FCFS):
 - first packet to join is first packet to leave
- 2) Priority:
 - arriving traffic classified, queued by class
 - any header fields can be used for classification
 - FCFS within priority classes
- 3) Round robin:
 - arriving traffic classified, queued by class
 - any header fields can be used for classification
 - server cyclically and repeatedly scans class queues, sending one complete packet from each class
- 4) Weighted Fair Queuing (WFQ):
 - generalized round robin
 - each class i has weight w_i , and gets weighted amount of service in each cycle

$$\frac{w_i}{\sum_j w_j}$$

D. Internet protocol

We now know the how routers route to destination addresses. But how are these addresses formatted, and how do they work? This is controlled by the *internert protocol*.

IP Datagram format.



- ver: IP protocol version number
- header length: total number of bytes
- type of service
 - diffserv (0:5): different classes of service
 - ECN (6:7): congestion control
- length: total datagram length
- TTL: remaining max hops (decremented at each router) - so packets don't loop forever
- 16-bit identifier, flgs and fragment offset: used for fragmenting datagram into smaller chunks. doesn't exist in IPv6
- upper layer protocol: 6 = TCP, 17 = UDP (transport layer protocol)
- header checksum: header checksum for header contents. Must be done at every *router* due to TTL decrementing. Removed in IPv6
- source IP address, dest. IP address: 32-bit IP addresses
- options: timestamp, route taken, etc

Overall, 20 bytes of headers

IP addressing.

IP addresses: 32-bit identifier associated with each host or router *interface*.

interface: connection between host/router and physical link

- router's typically have multiple interfaces
- host typically has one or two interfaces (wired Ethernet, wireless 802.11)

IP addresses have the format:

dotted-decimal IP address notation:

223.1.1.1 = 11011111 00000001 00000001 00000001
 | | | |
 223 1 1 1

Every 8 bits represents a number - separated by decimals.

IP Fragmentation and Reassembly.

Network links have **maximum transmission units (MTU)** - which is the largest possible link-level frame. Therefore, large IP datagrams must be fragmented within net. Fragmented datagrams have the following components:

- length: describing the length of the fragment
- id: describing which datagram the fragment belongs to
- mf: 'more fragments', denotes more fragments exist - 1 if not the last fragment.
- offset: starting position of first byte in multiples of 8 bytes. if the first fragment's **payload** took up 1480 bytes, then offset for second datagram is $1480 / 8 = 185$.

Consider this pre-fragment datagram

	length	ID	MF flag	offset	
	=4000	=x	=0	=0	

With an MTU of 1500 bytes, and a minimum header size of 20 bytes per datagram, we split up into fragments of 1480, 1480, 1020.

	length	ID	MF flag	offset	
	=1500	=x	=1	=0	

	length	ID	MF flag	offset	
	=1500	=x	=1	=185	

	length	ID	MF flag	offset	
	=1040	=x	=0	=370	

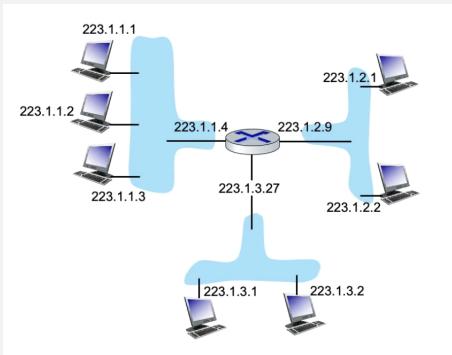
Network Masks, Broadcast Address and Network Address.

Network masks are a bit mask that identify the network portion of an address. They can be of arbitrary length (depending on the class or *CIDR* length).

- The **broadcast address** is when the host portion (the complement of the network mask) is all 1's.
- The **network address** is when the host portion is all 0's.

An IP address of 233.1.1.2 with a network mask of 255.255.255.0 has a network address of 233.1.1.0 and a broadcast address of 233.1.1.255.

Subnets.



What is a subnet?

- device interfaces that can **physically reach other** without passing through an intervening router (that is, not using the network layer).

Specifically, IP addresses are defined by:

- subnet part: devices in same subnet have common higher order subnets
- host part: remaining lower order bits
- so hosts and routers in the same subnet will have common part.

Class-ful address and Classless Inter-Domain Routing (CIDR).

At the beginning of the internet, there were different classes of network/host address partitions that existed. These defined the sizes of the networks and the size of the host portion:

	0	8	16	24	31
Class A	0	<i>netid</i>		<i>hostid</i>	
Class B	10	<i>netid</i>		<i>hostid</i>	
Class C	110	<i>netid</i>		<i>hostid</i>	
Class D	1110		<i>multicast address</i>		
Class E	1111		<i>reserved for future use</i>		

However this mean that ISPs had to buy fixed sizes of network sizes. Rather, CIDR fixed this problem by creating arbitrary length network portions, with the format $a.b.c.d/x$, where x denoted the network portions length in bits.

200.23.16.0/23

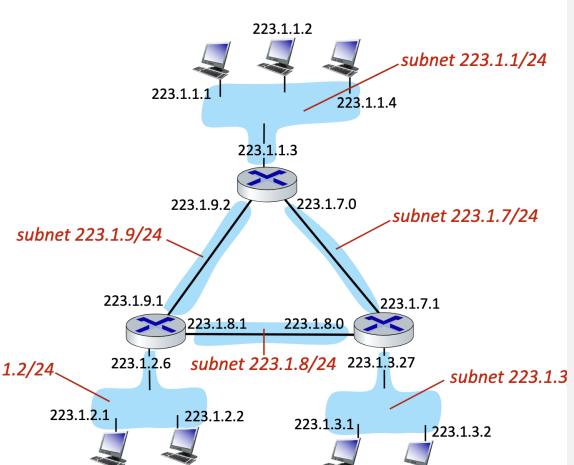
Reserved addresses in subnets.

There are two reserved addresses in networks

- The **network address**, which is the first address of the range

- Used to refer to the entire network
- The broadcast address, which is the last address of the range
 - Used to address packets to the entire network

Identifying subnets and creating forwarding tables.



Consider the forwarding table for the topmost router.

dest. network	next hop	# of hops	interface
223.1.1/24	-	0	223.1.1.3
223.1.9/24	-	0	223.1.9.2
223.1.7/24	-	0	223.1.7.0
223.1.8/24	223.1.9.1	1	223.1.9.2
223.1.2/24	223.1.9.1	2	223.1.9.2
223.1.3/24	223.1.7.1	2	223.1.7.0

Broadcast addresses. Broadcast addresses send a given packet to every address - hence called a broadcast address. In an IP address space, this address is the **last address**.

Consider a block of addresses from 144.4.64.0/26 to 144.4.64.63/26

In this case, the **broadcast address** is 144.4.64.63/26

Creating blocks of addresses. Consider an ISP has a block of addresses starting from 144.4.64.0/19. Let us translate this to binary:

1001000.00000100.010/00000.00000000

The / represents the beginning of the host portion.

Create 32 blocks of 64 IP addresses

- We first identify the size of our host portion.

We need 64 IP addresses in each block, so we need to solve $2^n = 64$ for n bits of host portion, $n = 6$.

- 2) Then, our CIDR notation is $/32 - 6 = /26$
- 3) We can then increment in increments of 63 (as it is inclusive) from our starting IP address to get the respective blocks.

Therefore, our blocks are:

144.4.64.0/26 to 144.4.64.63/26
 144.4.64.64/26 to 144.4.64.127/26
 144.4.64.128/26 to 144.4.64.191/26
 144.4.64.192/26 to 144.4.64.255/26
 ...
 144.4.71.0/26 etc.

To find the last range, you can consider how many blocks each 'group' has (so the .64 group has 4 blocks), and with 32 total blocks, that means there will be $8 - 1$ total increments.

Finding the start of subnet blocks.

You are likely to be asked to find the start of subnet blocks. Consider the following question

Suppose there is a block of address of the form 101.101.128.0/17. Split the addresses into equal blocks of four, and consider the subnet mask of the four subnets.

Finding the sizes of the four blocks is easy ($2^{15}/4$). From this, you can find the subnet mask needed (which is /19). Now you begin with:

101.101.128.0/19.

How do we find the start of the next block? Well there are $2^{15}/4 = 8192$ addresses per block. To see how much the third number should increase, we can divide 8192 by 256. That is $8192/256 = 32$, and thus the next subnet block starts at

101.101.160.0/19

If subnets are huge, then you might even divide by 2^{16} to get how many increments the second number increases.

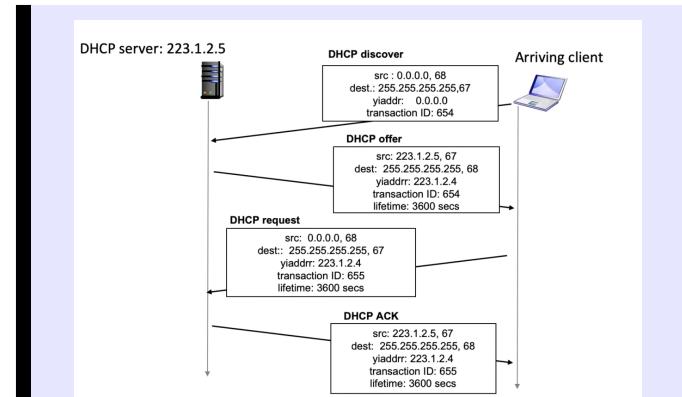
DHCP: Dynamic Host Configuration Protocol.

How do hosts get IP addresses?

Host dynamically obtains IP address from network server when it "joins" network

- can renew its lease on address in use
- allows reuse of addresses
- support for mobile users who join/leave network

DHCP Client-Server Example



- 1) DHCP servers listen on port 67 for incoming DHCP messages
- 2) The first message discovers a DHCP server at 255.255.255.67
- 3) The second message offers an IP address to the arriving client, in yiaddr
- 4) The third message requests the offered IP address
- 5) The fourth message acknowledges that request. For the specified lifetime, the IP address now belongs to the client.

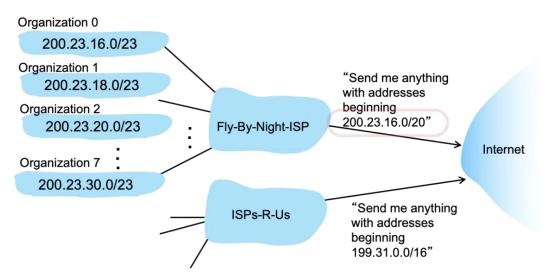
How do networks get IP addresses?.

Networks get an allocated portion of its provider ISP's address space.

How do ISPs get a block of IP addresses?.

- ICANN: Internet Corporation for Assigned Names and Numbers
 - allocates IP addresses, through 5 regional registries (RR) (who then allocate to local registries)
 - manages DNS root zone, including delegation of individual TLD (.com, .edu, ...) management
- Are there enough 32-bit IP addresses?
 - ICANN allocated last chunk of IPv4 addresses to RRs in 2011.
 - NAT (discovered next) helps IPv4 address space exhaustion
 - IPv6 has 128-bit address space

Hierarchical addressing.



The way we have defined IP addresses, networks and subnets, now allows for us to set a hierarchical structure to the internet. While this looks, neat, what happens when one of the organisations wishes to change ISPs?

- Consider a scenario where Organisation 1, with 200.23.18.0/23 wishes to switch to ISPs-R-Us
- Now, if it switches to ISPs-R-Us, we can add that to the condition which ISPs-R-Us sends to the internet
- But now both Fly-By-Night-ISP and ISPs-R-Us have the proper subnet mask for Organisation 1.
- **However**, longest prefix matching fixes this - since ISPs-R-Us has provided a longer subnet, traffic from the internet will come through to ISPs-R-Us.

E. NAT (Network Address Translation)

NAT. All devices in local network share just one IPv4 address as far as outside world is concerned. All datagrams leaving the local network, will use the same 32 bit IP address.

Further details of NAT.

- All devices in local network have 32-bit addresses in a "private" IP address space (10/8, 172.16/12, 192.168/16 prefixes) that can only be used in local network
- advantages:
 - just one IP address needed from provider ISP for *all* devices
 - can change addresses of host in local network without notifying outside world
 - can change ISP without changing addresses of devices in local network
 - security: devices inside local network not directly addressable

Implementation of NAT.

A NAT router must

- outgoing datagrams: replace (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
- remember (in NAT translation table) every

- (source IP address, port #) to (NAT IP address, new port #) pair
- incoming datagrams: replace (NAT IP address, new port #) in destination fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

Some controversies regarding NAT:

- routers "should" only process up to layer 3 (breaks encapsulation)
- address "shortage" should be solved by IPv6
- violates end-to-end argument (port # manipulation by network-layer device)
- NAT traversal: what if the client wants to connect to server behind NAT?

TCP Connections with NAT networks.

Consider Bob wants to create a TCP connection with Alice, who is behind a NAT router. She has yet to send a packet out to the network.

- Bob attempts to send to some given IP address and port
- Once the connection request reaches the router, the router has no idea where to send this packet
- Thereby the router drops the packet

In NAT routers, only outbound connections work (from Alice to Bob). To fix this issue, there is a concept of *connection reversal*, where another user who is connected to Alice connects the two by allowing Bob to request Alice to connect with him.

IPv6 and its motivations.

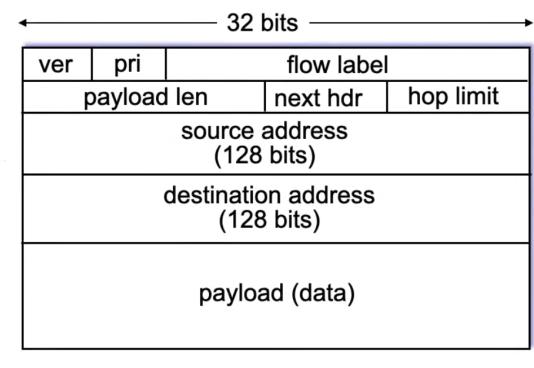
32-bit IPv4 address spaces (IPv4) would be completely allocated soon, so a larger space was required.

- Additionally, speed of processing and forwarding were improved by compressing headers
- Enable different network-layer treatment of *flows*
 - more akin to **circuits** than a datagram oriented design.

IPv6 is going through a transition state from IPv4.

IPv6 Datagram Format.

The below details the structure of an IPv6 datagram.



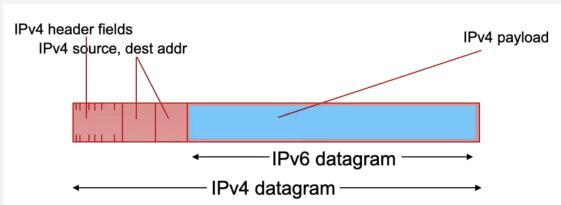
- flow label: identify datagrams in same flow
- 128-bit IPv6 addresses
- priority: identify priority among datagrams in flow

What's missing in IPv6?

- no checksum (to speed processing)
- no fragmentation
- no options

Transitioning from IPv4 to IPv6.

- not all routers can be upgraded simultaneously
- **tunneling:** IPv6 datagram carried as *payload* in IPv4 datagram along IPv4 routers (packet within packet)
- tunnelling is used quite extensively in other contexts



- When an IPv6 sends a datagram through an IPv4 network, it wraps the IPv6 datagram in an IPv4 datagram.
- Once the tunneled datagram comes back to an IPv6 router, it gets untunneled if it sends to another IPv6 router.

F. The control plane

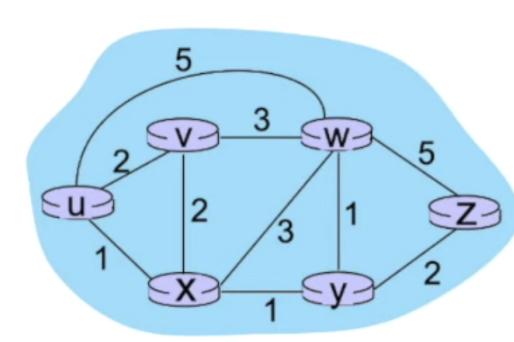
The control plane's responsibility is to forward datagrams to their correct destinations. There are two main methods:

- 1) **Per-router control plane:** the routers calculate the paths that the datagrams that arrive to them must go
- 2) **Software-Defined Networking (SDN) control plane:** there exists a remote server that calculates the paths that the datagrams in a network must go.

Routing algorithms.

Routing algorithms can be understood by graph abstraction of a network with its edges being denoted by the network operator:

- It could always be 1
- It could be inversely related to bandwidth
- It could be related to congestion



There are two classifications of routing algorithms:

- 1) *global*: all routers have complete information of link costs
- 2) *decentralised*: routers only have information of neighbours - iterative process required

Dijkstra's link-state routing algorithm.

- **centralised** and **iterative**
- after k iterations, we know the shortest path to k nodes

```

C(a, b): link cost a -> b
D(a): estimate of least cost path
p(a): predecessor node of a in path
N': set of nodes with path known

# Initialisation
N' = {u} # u is the source
for all nodes a:
  if a next to u:
    D(a) = c(u, a)
  else:
    D(a) = infinity

while |N'| < n:
  find a not in N' such that D(a) is min
  add a to N'
  update D(b) for b -> a and b != N':
    D(b) = min(D(b), D(a) + c(a, b))
  
```

Complexity of Dijkstra's algorithm.

$O(n \log n)$ using a Fibonacci heap for n nodes.

Each router must broadcast its link state information to other n routers. There exists broadcast algorithms which take $O(n)$ to broadcast to entire network of n routers. Therefore, for each n router, $O(n)$, therefore $O(n^2)$.

$$O(n^2) + O(n \log n) = O(n^2)$$

Troubles with Dijkstra's and traffic-dependent cost: oscillations.

- when link costs depend on traffic volume, *route oscillations* possible
- once a route is chosen, it will update the traffic taken in the route
- but when it reaches the next router, the traffic cost update may affect the path once more

The constantly changing router traffic may cause routes to go back and forth.

Bellman-Ford (B-F) Equation.

The Bellman-Ford algorithm is a dynamic programming algorithm that finds the single-source shortest path; unlike Dijkstra's, it is able to deal with negative edges.

The dynamic programming algorithm is dictated by equation

$$D(x \rightarrow y) = \min_{v \text{ adj. } x} \{c_{x,v} + D(v \rightarrow y)\}$$

B-F Algorithm.

Finding paths from x .

$C[a]$: cost array from x to a

$E[a, b]$: edge cost from a to b

N : total amount of nodes

Initialise neighbour node link

for $N - 1$ iterations:

```
for [a, b] in E:
    if C[a] + [a, b] < C[b]:
        C[b] = C[a] + [a, b]
```

Distance vector algorithm.

Distance vector. The *distance vector* is simply the cost information for a single node to every other node in the network.

- from time-to-time, each node sends its own distance vector (DV) estimate to neighbors
- when x receives new DV estimate from any neighbour, it updates its own DV using B-F algorithm.

Therefore, each node:

- wait for change in local link cost or msg from neighbour
- recompute DV estimates usign DV received from neighbour
- if my DV to any destination has changed, send **my DV** to my neighbours

Therefore, the distance vector algorithm is iterative, asynchronous, distributed and self-stopping.

Distance vector: link cost changes.

Link cost changes:

- node detects local link cost change
- updates routing info, recalculates local DV
- if DV changes, notify neighbours

Convergence.

Convergence is the time during which all routers come to an agreement about the best paths through the internetwork

- Networks advertise good news quickly
- but propagate bad news slowly

Count-to-infinity problem. Consider three routers, A, B and C. If A routes to C through B, then A tells B that C is unreachable when advertising routes back to B.

- We do this to avoid B believing that there is now an alternate route to C, when A is going through B
- It stops two routers relying on eachother for a destination that no longer exists.

Comparing link state (LS) and distance vector (DV) algorithms.

Message complexity

- LS: n routers, $O(n^2)$ messages sent
- DV: exchange between neighbors, convergence time varies

Speed of convergence

- LS: $O(n^2)$ algorithm, $O(n^2)$ messages which may have oscillations
- DV: convergence time varies
 - routing loops
 - count-to-infinity problem

Robustness; what happens if a router malfunctions

- LS
 - Router can advertise incorrect link cost
 - Each router computes its own table, so safer
- DV
 - DV router can advertise incorrect **path** cost

- each router's DV is used by network, and so this error propagates *throughout the network*.

G. Internet Control Message Plane

ICMP: internet control message protocol.

- used by hosts and routers to communicate network-level information
 - error reporting: unreachable host, network, port, protocol
 - echo request/reply (ping)
- network-layer "above" IP:
 - ICMP messages carried in IP datagrams, protocol number: 1
- ICMP messages: type, code plus header and first 8 bytes of IP datagram causing error

Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

How Traceroute utilises ICMP.

- source sends sets of UDP segments to destination
- 1st set has TTL = 1, 2nd set has TTL = 2, etc.
 - Remember TTL gets decremented at every router
 - So at the first router, the first set will be timed out, and an ICMP message will be sent.
- datagram in nth set arrives to nth router:
 - router discards datagram and sends source ICMP messages (type 11, code 0)
 - IP address of router where TTL expired is source IP address of datagram containing this ICMP message
- when ICMP message arrives at source: record RTTs
- how do the datagrams know when to stop?

- destination generally returns ICMP "port unreachable" message (type 3, code 3)
- but isn't necessarily true

Internet routing. Routing in the internet works across **two levels**:

- 1) Intra-domain (within ISP/Network)
 - An 'autonomous system' (AS) is a region of a network under one ISP
 - In an AS, intra-AS/intra-domain routing is established
- 2) Inter-domain (between ISP/Networks)
 - There is also routing between ISPs and networks.

V. Link Layer

The link layer is the lowest layer in the Internet protocol stack and is responsible for framing data for transmission over the physical medium, addressing within a LAN and error detection.

Framing. Framing is the process of packaging data for reliable delivery over a physical network. It provides structure, addressing and error-checking.

Services provided by the link-layer.

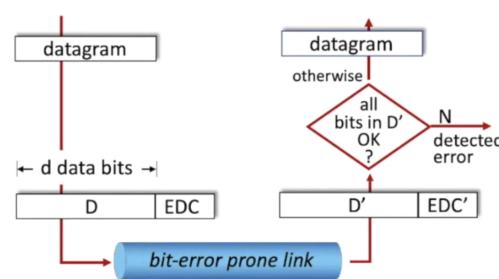
- 1) Framing
- 2) Reliable Delivery
- 3) Error-checking
- 4) Flow control
- 5) Full duplex

Terminology for the link layer.

- hosts, router: **nodes**
- communication channels that *directly connect physically adjacent nodes*: **links**
 - wired, wireless
 - LANs
- layer-2 packet: **frame**, encapsulates a datagram

A. Error detection and correction techniques

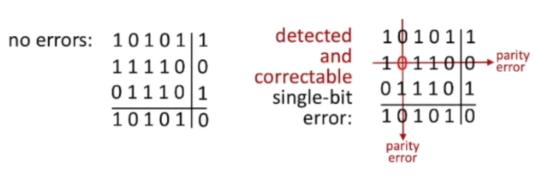
Error detection. Error detection can be abstracted by the following diagram



But how is error detection actually done?

Parity checking

- **single bit parity/1D parity:** a single bit is appended onto the datagram, in which the bit is set so there is a *even number of 1's*.
- On the receiver side, the receiver checks whether there is an even number of 1's
- **2D parity:** each row and column of bits has their own parity bits. The bits are arranged into a grid.
- In this case, we are able to detect two-bit errors, by checking the evenness of both rows and columns.
- In the case of a single bit error, we can *detect and correct* single bit error



We can also use *checksum* in the link layer. But it is hardly used (as it is not very powerful). **Cyclic Redundancy Check (CRC)**

- more power error-detection coding
- D: data bits (binary numbers)
- G: bit pattern (generator), of $r+1$ bits (given, specified in CRC standard)

$$\langle D, R \rangle = D \cdot 2^r \text{ XOR } R$$

- sender: compute r CRC bits, R , such that $\langle D, R \rangle$ exactly divisible by G ($\text{mod } 2$)
 - receiver knows G , divides $\langle D, R \rangle$ by G . If non-zero remainder: error detected
 - can detect all burst errors less than $r + 1$ bits
 - widely used in practice
- But how do we compute R ?

Well:

$$D \cdot 2^r \text{ XOR } R = nG$$

$$D \cdot 2^r = nG \text{ XOR } R$$

$$R = \text{remainder} \left[\frac{D \cdot 2^r}{G} \right]$$

An example of finding CRC parameters.

Consider that

- $D = 1101$
- $G = 1011$ (a four-bit polynomial of degree 3)
 - $x^3 + x + 1$
- Therefore, $r = 3$. D is in binary representation. So, $D \cdot 2^r$ is just a left shift by 3 for D . This

becomes

$$1101000$$

We now divide by G . We do this by continually XORing and bringing down the next bit (similar to long division).

- 1) $1101 \oplus 1011 = 0110$. Remove trailing zero, bring down next bit.
- 2) $1100 \oplus 1011 = 0111$. Remove trailing zero, bring down next bit.
- 3) $1110 \oplus 1011 = 0101$. Remove trailing zero, bring down next bit.
- 4) $1010 \oplus 1011 = 0001$. Thereby, the CRC bits are 001.

So $\langle D, R \rangle = 1101001$. Does this divide through to zero?

- 1) $1101 \oplus 1011 = 0110$. Remove trailing zero, bring down next bit.
- 2) $1100 \oplus 1011 = 0111$. Remove trailing zero, bring down next bit.
- 3) $1110 \oplus 1011 = 0101$. Remove trailing zero, bring down next bit.
- 4) $1101 \oplus 1011 = 0000$. Thereby, remainder is 0.

B. Multiple access links and media access control (MAC) protocols

Two types of "links":

- point-to-point
 - point-to-point link between Ethernet switch, host
- broadcast (shared wire or medium)/media access control (MAC)
 - old-school Ethernet
 - upstream HFC in cable-based access network
 - 802.11 wireless LAN, 4G, satellite

Thus, for a single shared broadcast channel, if there is two or more simultaneous transmissions by nodes, there exists a **collision**.

What is a multiple access protocol?

- distributed algorithm that determines how nodes share channel, i.e determine when node can transmit
- communication about channel sharing must use channel itself
 - no external channel for coordination...

Three broad classes of multiple access protocols:

- "taking turns": nodes take turns, but nodes with more to send can take longer turns
- random access: channel not divided, allow collisions - but recovery is implemented
- channel partitioning: divide channel into smaller pieces (time slots, frequency, node) - allocate piece to node for exclusive use

Channel partitioning protocols.

Time Division Multiple Access (TDMA).

- access to channel in "rounds"
- each station gets fixed length slot in each round
- unused slots go idle

Frequency Division Multiple Access (FDMA).

- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle

Random access protocols.

Slotted ALOHA.

- allows collisions to happen
- use randomisation to choose when to transmit
- all frames are the same size; times are divided into equal sizes; nodes are synchronised.
- senders randomly send frames
- if a collision occurs, node retransmits frame in each subsequent slot with probability p until success

Efficiency: long run fraction of *successful slots*

- prob. that given node has success in a slot: $p(1 - p)^{N-1}$
- prob that any node has a success $Np(1 - p)^{N-1}$
- max efficiency, find p^* that maximises $Np^*(1 - p^*)^{N-1}$

Carrier Sense Multiple Access (CSMA).

- simple CSMA: listen before transmit.
 - if channel sensed idle: transmit entire frame
 - otherwise: defer transmission
- CSMDA/CD: CSMA with collision detection
 - collisions detected within short time
 - colliding transmissions aborted, reducing channel wastage
 - collision detection easy in wired, difficult with wireless
 - *collisions may occur due to propagation delays*

Ethernet CSMA/CD algorithm.

- 1) Ethernet receives datagram from network layer, creates frame
- 2) If Ethernet senses channel:
 - if idle: start frame transmission
 - if busy: wait until channel idle, then transmit
- 3) If entire frame transmitted without collision – done!
- 4) If another transmission detected while sending: abort, and send a *jam* signal
- 5) After aborting, enter binary (exponential) back-off:
 - after m -th collision, chooses K at random from $\{0, 1, \dots, 2^m - 1\}$. NIC waits $K \cdot 512$ bit times, and then returns to step 2.
 - more collisions, means it's more likely there are more nodes. So the backoff interval becomes longer.

Taking turns protocols.

Polling.

- **centralised controller** uses polling messages to "invite" client nodes to transmit in turn
- controller forwards received frames to specified destinations

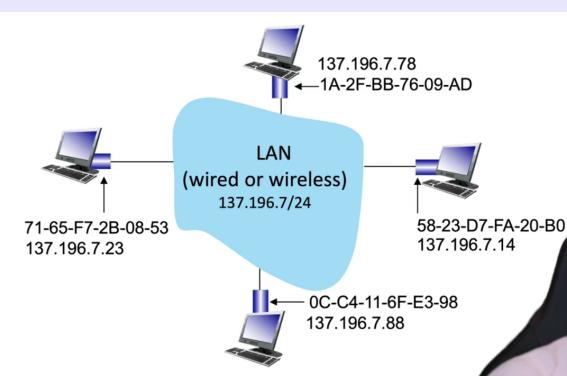
Token passing.

- control *token* message explicitly passed from one node to next, sequentially
- transmit only while holding token

C. MAC addresses and Address Resolution Protocol (ARP)

MAC addresses.

- MAC (or LAN or physical or Ethernet) address
- used *locally* to get frame from one interface to another physically connected interface (same subnet)
- 48-bit MAC address (for most LANs) embedded in network interface card (NIC)
- 1A-2F-BB-76-09-AD or 1A:2F:BB:76:09:AD
- exact matching **only**, unlike the prefix matching behaviour of IP addresses.



ARP: address resolution protocol.

How to determine interface's MAC address, knowing it's IP address?

- ARP table: each IP node on LAN has table
 - IP/MAC address mappings for some LAN nodes
- <IP address; MAC address; TTL>

But how are these ARP tables populated?

- 1) A wants to send datagram to B, but B's MAC address is not in A's ARP table.
- 2) A broadcasts ARP query, containing B's IP address
 - If a node closer to A then B contains the MAC address, it sends back the address
 - Otherwise, this query eventually reaches B, in which the destination returns the MAC address

Hosts at different subnets do not care about each other's MAC addresses, and rather relays this job onto further hops. Thus, the MAC address of the packet on a multi-hop packet refers to the next hop.

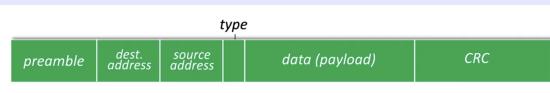
D. Ethernet and Switches

Ethernet.

- first widely used LAN technology
- simpler, cheap
- kept up with speed race: 10 Mbps - 400 Gbps
- single cheap, multiple speeds

Buses - where there was one long link, and thus all nodes share the same collision domain were very popular. However, **switches are now prevalent**.

Switches prevail today, which is a layer-2 (link layer) device. This means they only serve to forward in a LAN.



- 48-bit source, destination MAC addresses.
- preamble is used to synchronise receiver and sender clock rates
- types indicates higher layer protocol. used to multiplex
- CRC: cyclic redundancy check at receiver

Adapters, Network Interface Cards (NIC) and Network Adapters. Adapters are the hardware (with associated software) that implement the link-layer

Switches.

- switches are strictly link-layer devices
- stores and forwards ethernet frames
- examine incoming frame's MAC address, selectively forward frame to one-or-more outgoing links, uses CSMA/CD to access segment
- **transparent**: hosts unaware of presence of switches
- Switches can allow multiple frames to forward through to different destinations, unlike a bus configuration

How does a switch know which interface to forward to?

- switches learn which hosts can be reached through which interfaces.
- switches too keep an interface table, matching addresses to interfaces
- if the MAC address does not exist on the interface table, then it sends the frame to ALL interfaces to learn.

Interconnecting switches

- switches are fairly plug-and-play; as switches are self learning.
- multiple switches can exist, connecting to different switches, as interface tables will be filled.
- intermediary switches will store mappings to other switches, which are connected to hosts

Broadcast LANs vs Switched LANs.

Broadcast LANs are LANs where each frame sent from a host X is received by every single other node. Wireless networks are a common example of broadcast LANs. Thus, all frames are received and processed by each host (but not necessarily decapsulated).

In switched LANs, only the intended destination addresses receive the frames.

VI. A day in the life of a web request

We now consider the journey of a web request for a laptop which connects to an ethernet port, and then attempts to load google.com

1) First, we need an IP address

- The laptop uses a DHCP request encapsulated in UDP, encapsulated in IP
- The frame is broadcasted
- The DHCP server sends a DHCP ack, and is forwarded back to the client with an IP address, IP address of first hop, name and IP address of DNS server
- The client accepts these terms, and then the DHCP server acknowledges the client accepting these terms.

The client now has an IP address, it's first hop and the IP address of the DNS server.

2) Now the client sends a DNS request - but we don't know the DNS server's MAC address yet

- An ARP query is sent for the DNS server's MAC address
- An ARP reply is sent back by the DNS server (or another closer hop)

3) Then the DNS response is sent with the IP address of the web page.

4) Now a HTTP request is sent to the web page's IP address

- A TCP connection must be established first to the web page's host
- After this, a HTTP request is sent.
- A HTTP response is sent back, and the transaction is completed.

For these questions, consider each layer that it passes through and anything important that occurs at each layer.

VII. Wireless and Mobile Networks

Elements of a wireless network. Consider the following elements of a wireless network

Wireless hosts.

- laptop, smartphone, IoT
- run applications
- may be stationary (non-mobile) or mobile

Base station/access points.

- typically connected to a wired network
- relay - responsible for sending packets between wired network and wireless host(s) in its "area"

Wireless link.

- typically used to connect mobile(s) to base stations to base station, also used as a backbone link
- multiple access protocol coordinates link access
- various transmission rates and distances, frequency bands

Ad hoc wireless networks.

- no base stations
- nodes can only transmit to other nodes to link coverage
- nodes can organise themselves into a network: route among themselves

	single hop	multiple hops
infrastructure (e.g., APs)	host connects to base station (WiFi, cellular) which connects to larger Internet	host may have to relay through several wireless nodes to connect to larger Internet: <i>mesh net</i>
no infrastructure	no base station, no connection to larger Internet (Bluetooth, ad hoc nets)	no base station, no connection to larger Internet. May have to relay to reach other a given wireless node MANET, VANET

Wireless link characteristics.

- **decreased signal strength:** radio signal attenuates as it propagates through matter
- **interference from other sources:** wireless network frequencies are shared by many devices
- **multipath propagation:** radio signal reflects off objects ground, arriving at destination at slightly different times

Thereby, wireless links are quite noisy - and to be considered useful, it must have a high enough *signal-to-noise ratio* (SNR). SNR is considered with reference to bit error rate (BER)

- given physical layer: increase power → increase SNR → decrease BER
- given SNR: choose physical layer that meets BER requirement, giving highest throughput

Hidden terminal problem.

- B and A hear each other
- B and C hear each other
- A and C can not hear each other → A and C are unaware of their interference at B (collisions)

Signal attenuation.

- Signals weaken as it travels through a medium or space
- Therefore, two hosts may not be able to reach other
- This can be the reason for causing the hidden terminal problem

A. How Wi-Fi works

LAN architecture in Wi-Fi networks.

Basic Service Set (BSS) consist of:

- wireless hosts (only hosts if ad hoc mode)
- access point (AP); base station

The Wi-Fi spectrum is divided into channels at different frequencies

- Access point (AP) admin chooses frequency for access point
- Interference possible: channel can be same as that chosen by neighboring access point

Connecting a new host. When a new host arrives, they must associate with an AP.

- scans channels, listening for beacon frames containing AP's name (SSID) and MAC address
- selects AP to associate with
- they may perform **authentication**
- then typically runs DHCP to get IP address in AP's subnet

Scanning.

Passive scanning involves:

- beacon frames are sent from APs
- association requests are sent from base stations, and then devices respond

Active scanning involves:

- Probe request frame broadcasts are sent from device
- Probe response frames are sent from APs
- Association frame requests are sent from the device to AP
- Association response frames are sent from the AP to device

B. IEEE 802.11: Wireless LAN

CSMA/CA (Collision avoidance).

CSMA/CA is akin to CSMA/CD, but with collision avoidance instead of collision detection. **SIFS** stands for shortest interframe space, and is the shortest waiting time. **DIFS** stands for distributed interframe space, and is the time waited before sending a new data frame.

On the sender side

- 1) If sense channel idle for DIFs, then transmit entire frame (no collision detection)
- 2) If channel is busy, then
 - a) start random backoff time
 - b) timer counts down while channel idle
 - c) transmit when timer expires
 - d) if no ACK, increase random backoff interval, repeat 2

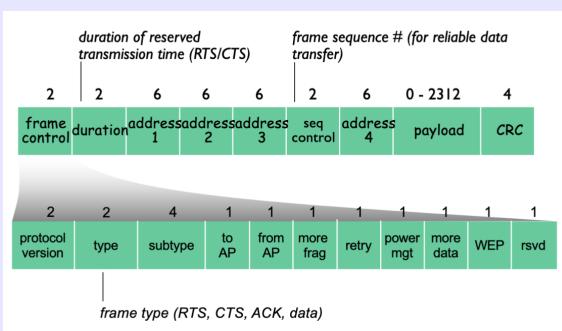
On the receiver side

- 1) If frame received OK
 - a) return ACK after SIFS

The idea is that small request packets are sent to the base station, to consider whether they are ready to receive.

- sender first transmits small request-to-send (RTS) packet to base station using CSMA
 - RTS may collide; in this case, no clear-to-send (CTS) is sent.
- Base station broadcasts CTS in response to RTS
- CTS heard by all nodes
 - sender transmits data frame
 - other nodes stop transmission

IEEE 802.11 MAC Protocol.



- Address 1: MAC address of wireless host or AP to receive the frame
- Address 2: MAC address of wireless host or AP transmitting this frame
- Address 3: MAC address of router interface to which AP is attached
- Address 4: used only in ad hoc mode
- Duration: duration of reserved transmission time
- Seq control: frame sequence number (for RDT)
- type: frame type (RTS, CTS, ACK, data)

VIII. Network Security

A. Cryptography

Cryptography principles.

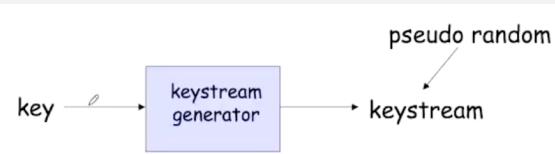
- **Confidentiality:** only sender, intended receiver should "understand" message contents
- **Authentication:** sender, receiver want to confirm identity of each other
- **Message integrity:** sender, receiver want to ensure message not altered without detection

Ciphers and their types.

A cipher is an algorithm or method used to **encrypt** and **decrypt** information - transforming plaintext into ciphertext and vice versa.

- **Stream ciphers:** encrypt one bit at a time
- **Block ciphers:** break plain text into equal-size blocks, and then encrypt each block as a unit

Stream Ciphers.

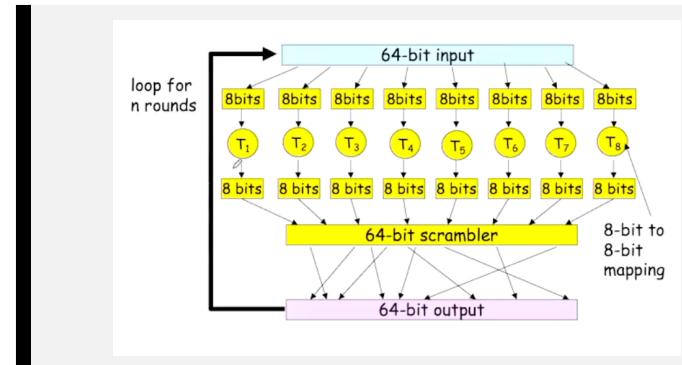


- Combine each bit of keystream with bit of plain-text to get bit of cipher text
- $m(i)$ is the i th bit of message
- $ks(i)$ is the i th bit of keystream
- $c(i)$ is the i th bit of ciphertext
- $c(i) = ks(i) \oplus m(i)$
- $m(i) = ks(i) \oplus c(i)$

The key is the input to a cipher, which creates the text which will cipher the input.

Block Cipher.

- Ciphertext processed as k bit blocks
- 1-to-1 mapping is used to map k -bit block of plaintext to k -bit block of ciphertext
- To prevent brute force attacks, choose large k .
 - We will have 2^k entries if we used a block cipher table (where each k bit block is mapped to a distinct k bit cipher).
- Rather than having one large table, multiple small tables could be used and re-scrambled to create more randomness.



Breaking encryption schemes.

- cipher-text only attack: the attacker only has cipher text to analyse
 - they can either brute force (search through all the keys)
 - statistical analysis
- known-plaintext attack: the attacker has plain-text corresponding to ciphertext
- chosen-plaintext attack: the attacker has the cipher text for a given plaintext

Symmetric key cryptography and some standards.

Both the encryptor and the decryptor share the same (symmetric) key. To agree on a value, the two participants use some network protocol.

Data Encryption Standard (DES).

- US encryption standard
- 56-bit symmetric key, 64-bit plaintext input
- block cipher with cipher block chaining
- now brute-forceable in less than a day
- there is no good 'statistical' attack
- 3DES (encryption with 3 different keys) makes DES more secure

Advanced Encryption Standard (AES).

- symmetric-key, processes data in 128 bit blocks
- 128, 192 or 256 bit keys
- brute force decryption takes 149 trillion years for AES

Cipher Block Chaining (CBC).

Motivation: A problem that arises is that repeated *input blocks* will result in the same cipher text, which may provide vulnerabilities to malicious users. An initial attempt at solving this problem is as follows:

- XOR the i -th input block $m(i)$ with some random number $r(i)$.

- Then for some symmetric key K_s , we have:

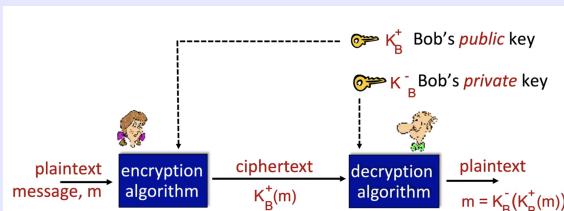
$$C(i) = K_s(m(i) \oplus r(i))$$

where $C(i)$ is the cipher text for the i -th block
• $r(i)$ and $C(i)$ can then be sent to the receiver

Cipher block chaining simplifies the above method by sending one random value $C(0)$ at the start. For subsequent blocks, $C(i - 1)$ is used to XOR.

Public Key Cryptography.

In public key cryptography, the sender and receiver do not share a private key. Rather, there exists a public encryption key known to all, and a **private decryption key known only to receiver**.



There are two general requirements for public key encryption algorithms

- need $K_B^+(x)$ and $K_B^-(x)$ such that

$$K_B^-(K_B^+(x)) = x$$

- given public key K_B^+ , it should be impossible to compute private key K_B^-

An aside about modular arithmetic.

- $[(a \text{ mod } n) + (b \text{ mod } n)] \text{ mod } n = (a + b) \text{ mod } n$
- $[(a \text{ mod } n) - (b \text{ mod } n)] \text{ mod } n = (a - b) \text{ mod } n$
- $[(a \text{ mod } n) * (b \text{ mod } n)] \text{ mod } n = (a * b) \text{ mod } n$
- $(a \text{ mod } n)^d \text{ mod } n = a^d \text{ mod } n$

RSA Algorithm.

To RSA, messages are a bit pattern. Bit pattern can be uniquely represented by an integer number. Thus, encrypting a message is the same as encrypting a number. Consider a message m .

- Choose two large prime numbers p, q (e.g 1024 bits each)
- Compute $n = pq, z = (p - 1)(q - 1)$ (z is also expressed as $\phi(n)$)
- Choose $e : e < n$ (e for encryption) that has no common factors with z
- Choose d (d for decryption) such that $ed - 1$ is exactly divisible by z
- Public key is (n, e) and private is (n, d)

Encryption and decryption.

- given (n, e) and (n, d)
- to encrypt message m , compute $c = m^e \text{ mod } n$
- to decrypt received bit pattern c , compute $m = c^d \text{ mod } n$

$$m = (m^e \text{ mod } n)^d \text{ mod } n$$

An example of RSA keys and utilising them.

- Choose two prime numbers, $p = 2$ and $q = 7$
- Now, $n = 2 \cdot 7 = 14$.
- We now have that $\phi(n) = (2 - 1) \cdot (7 - 1) = 6$
- We now have to find the encryption key e .
 - $e < n$ and e is coprime with $\phi(n)$.
 - 5 is a candidate for e . Let $e = 5$.
- We now have to find the decryption key d .
 - We need to find d such that $5d - 1 \text{ mod } 6 = 0$
 - $d = 11$ fits this - as $55 - 1 \text{ mod } 6 = 54 \text{ mod } 6 = 0$.

So we have $K^+ = (14, 5)$ and $K^- = (14, 11)$.

Now consider encrypting and decrypting the letter B , and let it equal 2. To **encrypt**:

$$\bullet \quad c = 2^5 \text{ mod } 14 = 4$$

And then to **decrypt**:

$$\bullet \quad m = 4^{11} \text{ mod } 14 = 4194304 \text{ mod } 14 = 2$$

which is our original message!

The symmetry and properties of the RSA algorithm.

Consider the following important property

$$K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$$

Why is RSA secure?

- computing the factors of n without knowing p and q is computationally very difficult because they are given to be prime
- they are also extremely large numbers
- but, RSA is computationally expensive**, and so RSA isn't a fix-it-all solution
- rather, RSA is used to exchange the symmetric session key K_s , and then following this, using symmetric cryptography.

B. Authentication and Message Integrity

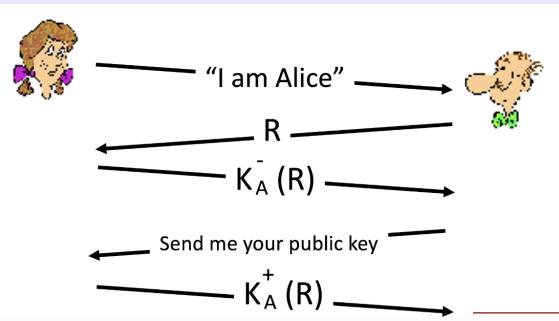
What is authentication?.

The goal of authentication is to verify to the recipient that the sender is actually the advertised sender. Consider the below novel attempts at solving the authentication problem.

- Just sending 'I am Alice' to Bob can be spoofed by any attacker.
- Alice could also send her IP address - but attackers can spoof her IP address.
- Alice could send her IP address and a 'secret password' - but attackers could sniff this packet, and re-send it as a **replay attack**.
- Alice could send her IP address and a *encrypted* password (using a public key) - but the **replay attack** still works here.

A **nonce** is a number used only once-in-a-lifetime.

- Instead, Bob could send a nonce R to Alice, and Alice must return R encrypted with a shared secret key.
 - The shared secret key would be obtained using public key cryptography
 - Once Alice sends the encrypted R , Bob can obtain her public key, decrypt it, and confirm that it is R
 - But a malicious attacker could *still spoof the initial approach* - and afterwards use their own public and private key sets to validate.



Digital Signatures.

A digital signature is a cryptograpgic technique analogous to hand-written signatures

- sender (Bob) digitally signs document
- *verifiable, nonforgeable*: recipient (Alice) can prove to someone that Bob, and noone else (including Alice), must have signed document
- simple digital signature for message m
 - Bob signs m by encrypting with his private key K_b , creating "signed" message $K_b^-(m)$
- Thereby applying Bob's public key should resolve in the original message, verifying Bob owns the message.

Message digests.

Message digests aim to make message signatures more efficient by producing a smaller footprint of the intended message m .

- **Goal:** *fixed-length, easy-to-compute digital footprint.*

- apply hash function H to m , get fixed sized message digest $H(m)$
 - The function is many-to-1 (many messages may get the same hash)
 - produces a fixed-size msg digest
 - given a message digest x , it is computationally infeasible to find m

So how does it all come together?.

On the sender's side

- 1) The message m is hashed with hash function H to become $H(m)$
- 2) The sender's private key K_S^- is applied, to produce $K_S^-(H(m))$
- 3) The sender sends m as well as $K_S^-(H(m))$

On the receiver's side

- 1) The message m and the hashed function $K_S^-(H(m))$ is received.
- 2) The message m is hashed on the receiver side to produce $H(m)$.
- 3) The receiver requests the sender's public key K_S^+ .
- 4) The encrypted hash message is decrypted, $K_S^+(K_S^-(H(m))) = H(m)$
- 5) If the hashed message and the decrypted hash message are equal, then success.

Hash function algorithms.

But how is the hash function H determined?

- MD5 hash function is widely used
 - computes 128-bit message digest in 4-step process
- SHA-1 is also used
 - this is the US standard
 - 160-bit message digest
- SHA-2 and SHA-3 are more recent standards.

Now we have solved that we can verify signatures to a given public key. This doesn't necessarily guarantee that whoever we have received the message and the signature from, is actually the person we intend to receive from. Thereby, we need a way to bind *identities* to *public keys*.

Certification Authorities (CA).

Certification authorities bind public keys to a particular entity, E .

- Entity (person, website, router) registers its public key with a certifying entity (CE), providing "proof of identity" to CA
 - CA creates certificate binding identity E to E 's public key
 - certificate containing E 's public key digitally signed by CA: CA says "this is E's public

key"

Now, when a receiver receives a message - they will no longer receive the public key directly from the sender; but rather receive it from a certifying entity (CE).

C. Securing E-mail

E-mail confidentiality.

Sender side

- 1) Generate a random symmetric key K_s
- 2) Encrypt message with K_s
- 3) Also encrypt K_s with receiver public key
- 4) Send both $K_s(m)$ and $K_B^+(K_s)$ to receiver

Receiver side

- 1) Uses private key to decrypt for K_s
- 2) use K_s to decrypt $K_s(m)$, and read message m .

E-mail integrity and authentication.

Sender side

- 1) Sender digitally signs their message with their private key, providing integrity and authentication
- 2) Sends both the message (in the clear) and digital signature

Receiver side

- 1) Receiver receives the message as well as the signed message
- 2) Requests sender's public key from a certifying entity - and then decrypts the signed message
- 3) If both messages (message digests) match, then integrity and authentication are upheld.

Now combining both, the sender would utilise:

- symmetric key (confidentiality)
- receiver's public key (confidentiality)
- their own private key (integrity and authentication)

PGP: a standard for secure email.

- De-facto standard for email encryption
- Comes with:
 - public, private key pair
 - MD5 or SHA for message digest
 - CAST, triple-DES or DEA for symmetric key
 - RSA for public key encryption