Computer Networks

Ferdinand Brunne

30. Juli 2024

Inhaltsverzeichnis

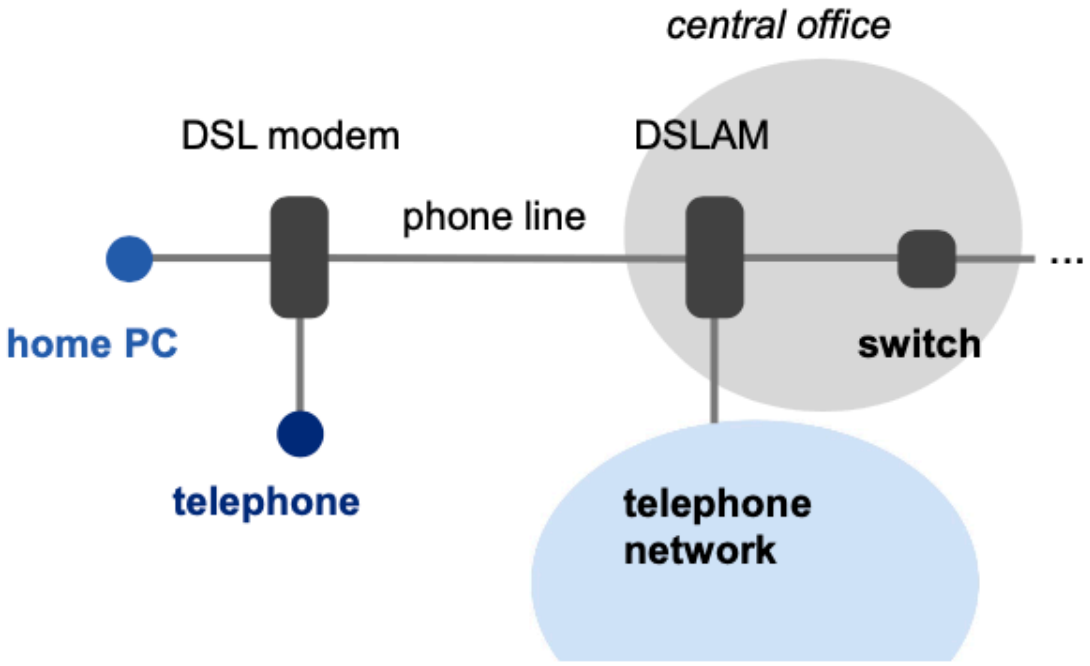
1 Overview & principles of Networks

1.1 What is the network made of?

Graph made of

- End-Systems: (e.g. smartphone, PC, server, ...) → verticies
- switches/routers: in-network → verticies
- Links (connecting switches, routers and end-systems) → edges

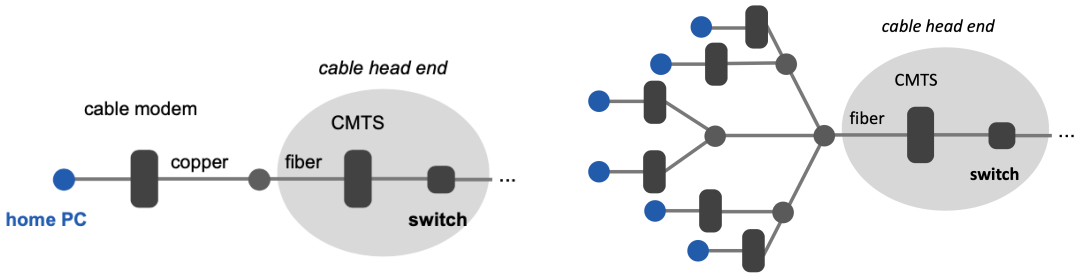
1.1.1 DSL (=Digital Subscriber Line)



Connection composed of:

- Downstream data channel (high throughput) + Upstream data channel (low throughput)
- 2-way phone channel

1.1.2 Cable



Many households share the same channel

1.2 How is the network shared?

There are two ways to distribute the available bandwidth to multiple end-systems using the same physical cable: Reservation (circuit-switching) and best-effort (packet-switching)

1.2.1 Reservation

Ask for resources **in advance**: Get access or not. If access: resources can be used

1. User:
   (a) Needs to know **what to ask for and when** (prediction of what to send)
   (b) Needs to **request** reservation
   (c) If access, access **guaranteed**

(d) **Pays per reservation**

2. Network:

   (a) Has **admission control** (needs to figure out if it has capacity)
   (b) Needs to keep **state** at all switches to **save who has reservation**
   (c) **Schedule** data from different sources

1.2.2 Best-Effort

Send data when necessary: Network **buffers or discards** packets if they don't fit.

1. User:

   (a) **No prediction or reservation** of data necessary
   (b) But: **No short-term guarantees**, must adapt to network
   (c) **Pays only for resources they use**

2. Network:

   (a) Can **accept any data** it has capacity for
   (b) Can **drop the data at any point** if no capacity
   (c) **No state for reservation** necessary
   (d) Can **treat all data equally**

1.2.3 Utilization: Reservation vs Best-effort

For **Peak rate** P and **average rate** A, **Utilization** defined as $\frac{A}{P}$.

- **Reservation** usually has low utilization (reserves bandwidth but doesn't use everything all the time) $\rightarrow$ only use if need **guarantees** for data to come through, or if one knows that one will have **high utilization** (low P/A ratio)
- **On demand** can achieve higher utilization, but is inconsistent.

1.3 How does communication happen? (Principles of Networking)

Communication happens through **protocols** that **define the way the exchange of informations happens**.

1.3.1 First principle: Modularity (Network Layer Model)

Needed to **provide structure** for the protocols. Each **layer is implemented on the network stack**. Internet communication can be decomposed in **5 layers**: Each layer takes a message from

L	Name	service provided	role	protocol
5	Application	network access	exchanges **messages** between processes (on different systems)	HTTP, SMTP, FTP, SIP
4	Transport	end-to-end delivery	transports **segments** between end-systems	TCP (reliable), UDP (unreliable), SCTP
3	Network	global best-effort delivery	moves **packets** around the network	IP
2	Link	local best-effort delivery	moves **frames** across a link	Ethernet, Wifi, DSL, LTE, ...
1	Physical	physical transfer of bits	moves **bits** across a physical medium	Twisted pair/fiber/coaxial-cable

the layer above and encapsulates it with its own header to identify in which layer the data is.

1.3.2 Second principle: end-to-end principle

1. **General question**: How to make data delivery reliable?
2. **Realization**: Checking reliability only inside the network (and not at the end devices) is not sufficient: Network itself can always fail.
3. **End-points check for correctness**;
4. **Retry on failure**. Now the network can be best-effort and unreliable. Communication only fails if endpoints fail or the whole network fails

Implementing reliability inside the network:

1. Can **reduce number of necessary retries** and thus **improve performance**, but: **increased complexity**, often not necessary for many applications, doesn't improve end-point complexity
2. Therefore: **Only implement on lower level if host cannot or if it improves the performance** (and doesn't introduce unnecessary overhead for unrelated applications). e.g. protection from malicious hosts, DDoS in network

1.3.3 Third principle: Fate-sharing principle

= When storing state in a distributed system, **co-locate it with entities that rely on that state**.

1. = State + program that relies on state, share the same fate together
2. If program fails, state is removed and doesn't affect the rest of the system
3. Necessary so that not everything breaks if something crashes

1.4 How to characterize the network?

1.4.1 Delay (Latency)

= Time it takes for a packet to reach its destination. There are several factors increasing the delay at each node in the network:

1. **Transmission delay** = Time required to **push all the bits** of a package onto the network link = $\frac{\text{packet size}}{\text{link bandwidth}}$
2. **Propagation delay** = Time required for a bit to **travel** to the end of the link = $\frac{\text{link length}}{\text{travel speed}}$
3. **Queuing delay** At each node data is stored in queues and then propagated onto the link. Therefore the queuing delay depends on the **pattern of the data send through the node** (e.g. if multiple devices send at the same time the queuing delay is higher).
   = The time a packet has to **wait in a buffer** before being processed or sent. Depends on:
   (a) Average packet arrival rate $a = \frac{\text{packets}}{\text{second}}$
   (b) Average packet arrival rate $a = \frac{\text{packets}}{\text{second}}$
   (c) - fixed packet length $L = \frac{\text{bits}}{\text{packet}}$
   (d) - average bits arrival rate $La = \frac{\text{bits}}{\text{second}}$
   (e) - traffic intensity $\frac{La}{R}$
      i. If > 1 (i.e. more bits arrive the leave a node): queue increases and therefore also the queuing delay
      ii. If ≤ 1 queue shrinks, queueing delay dependent on burst size of traffic (i.e. if much data arrives at once it is bursty)
4. **Processing delay**: Very small, usually doesn't matter

1.4.2 Loss

If a queue is full packets need to be dropped (packed loss). Mainly characterized by queue size.

- Can cause delays because of retries. Can reduce chance of needing a retry by sending data multiple times

1.4.3 Throughput

= rate at which host receives data = $\frac{\text{data size}}{\text{transfer time}}$. Depends on the transmission time of the node with the slowest transmission time = **bottleneck link**

2 Application Layer

2.1 DNS

= Domain Name Service. The DNS System is a distributed (on multiple computers) database that resolves a name to an IP address (e.g. ethz.ch to 129.132.19.216).

- n to n relation: One name can have multiple IPs (to distribute the access: load balancing), multiple names can be mapped to one IP (to reuse the machine)

2.1.1 DNS Hierarchies

1. **Naming structure**
   (a) Addresses are hierarchical: read name from right to left (e.g. inf.ethz.ch → ch. → ethz. → inf.)
   (b) Top Level Domains (TLDs) e.g. .com, .de, .ch are at the top. Each TLD is connected to root
   (c) Each domain (2LDs) e.g. ethz.ch, google.com are subtrees of one TLD
2. **Hierarchical administration**
   (a) Each node in the name tree can be administered by different entities (e.g. root is managed by the Internet Assigned Numbers Authority, ch managed by Swiss Education & Research Network, ...)
   (b) This means that name collision is trivially avoided: There is exactly one path in the name tree + there is only one root! (exception: private namespaces)
3. **Hierarchical infrastructure**
   (a) the 13 root servers are managed professionally (e.g. a.root-servers.net by VeriSign, Inc). Use of **BGP anycast**: not only 13 physical servers: multiple servers for one root server
   (b) The TLDs are managed by private or non-profit organizations
   (c) The rest is managed by Internet Service Providers (ISPs) or locally

2.1.2 DNS Table Entry

1. DNS server stores resource records including (*name*, *value*, *type*, TTL)
2. TTL stands for Time To Live and corresponds to the time a DNS Service waits to fetch the record again (i.e. how often an entry is updated). name and value have different meanings depending on the type:

type	name	value
A	hostname	IP-address
NS	domain	DNS server name
MX	domain	mail server name
CNAME	alias	canonical name
PTR	IP-address	hostname

2.1.3 DNS Reliability and Availability

1. DNS queries and replies use UDP (Port 53), reliability is implemented by retries
2. To ensure availability each domain must have at least a primary and a secondary DNS Server
   - DNS available as long as one is up
   - DNS requests can be load-balanced
   - If timeout on one server can use others (retrying the same server only after exponential backoff possible)

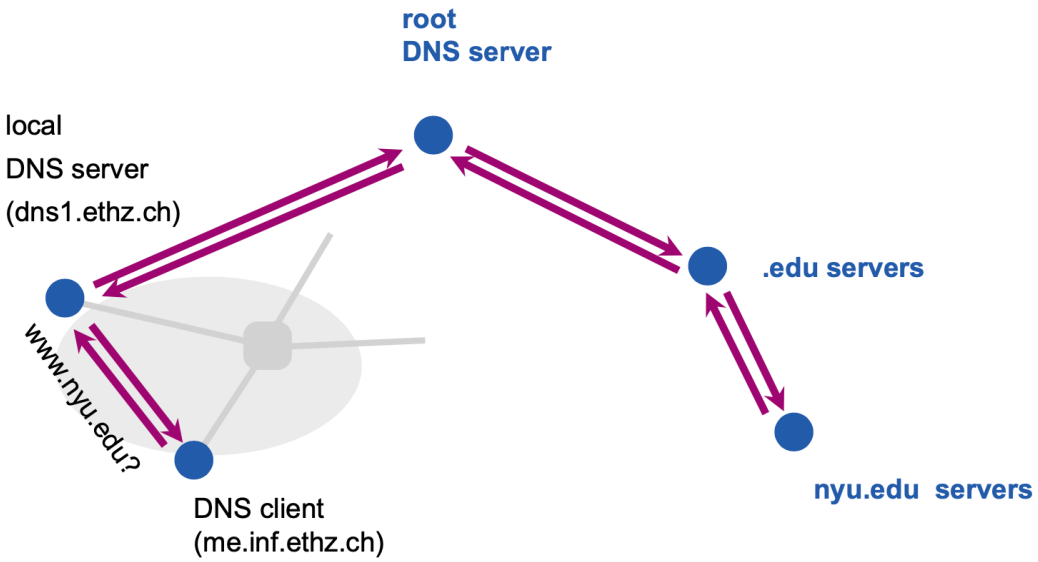
2.1.4 DNS Address Resolving

1. Each DNS Server knows the addresses of the 13 root servers
2. Each root server knows the addresses of the TLD servers
3. Each TLD server knows the addresses of 2LD servers...

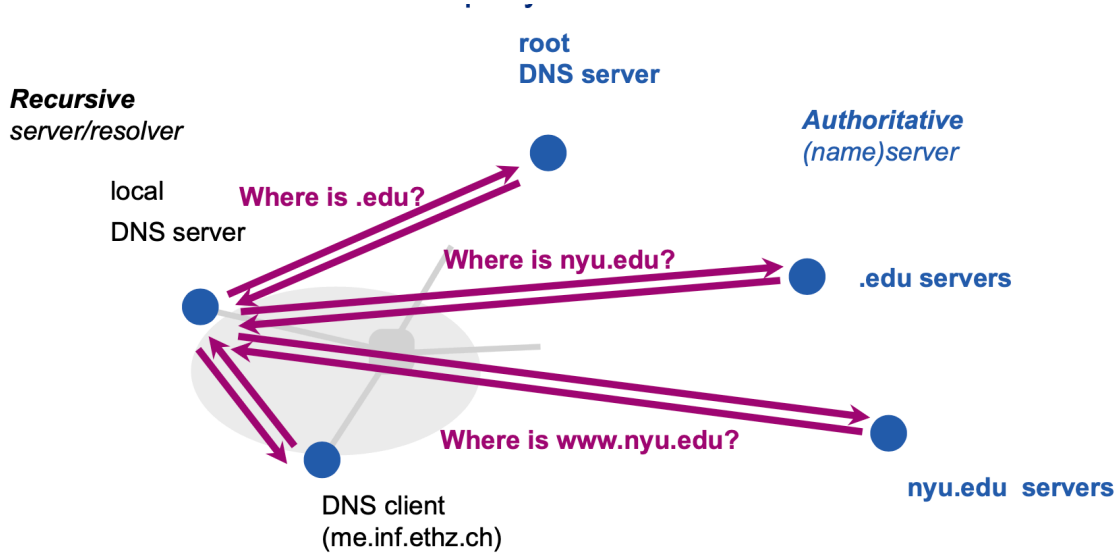
An address is usually resolved as follows:

1. Send request to local DNS server (Resolver software = stub resolver)
2. DNS server (usually near the endhost) returns IP (=recursive resolver)

Recursive DNS resolution



Inefficient as each node has to store state about request → root server is bottleneck
Iterative DNS resolution



First ask root and return, then TLD and return then 2LD... Can be seen in console with `dig +trace nyu.edu`

DNS Caching

Each DNS Server relies on caching to be able to access frequently visited addresses quicker. TTL determines the time an entry can be cached.

2.1.5 Glue Records

A NS record (i.e. a record assigning pairing domain and DNS server name) can have circular dependencies:

1. The name of a DNS server can include the domain (e.g. the name of the DNS server for google.com is ns1.google.com)
2. Additional section contains **Glue records**: Assign DNS server name a domain (e.g. ns1.google.com to 216.239.32.10)

2.2 The Web

2.2.1 URL

= Uniform Resource Locator; refers to an internet resource in the form `protocol://hostname[:port]/directory_pat` where:

1. `protocol` is the protocol used to make a connection (e.g. HTTP(S), FTP, SMTP, ...)
2. `hostname` is the IP or DNS name of the server
3. `port` the port at which the server is accessed. Defaulted to protocol standard (e.g. 80 for HTTP or 443 for HTTPS)
4. `directory_path/resource` identifies the resource on the server

2.2.2 HTTP

= synchronous request-response protocol

- stateless, text-based, line-orientated

HTTP request format

1. First line:

(a) method: GET (want resource from server), POST (send data to server) or HEAD (want only header)

(b) url: server-relative address (e.g. /index.html)

(c) version (1.0, 1.1, 2.0, 3.0)

2. header (variable length, human readable): authorization info, user agent, referrer (i.e. the cause of the request), ..

3. body: content of request

HTTP response format

1. First line:

   (a) version: GET (want resource from server), POST (send data to server) or HEAD (want only header)

   (b) status: 1xx (informational), 2xx (success), 3xx (rediction), 4xx (client error), 5xx (server error)

   (c) reason phrase: 200 (OK), 301 (moved permanently), 303 (moved temporarily), 304 (not modified), 404 (not found), 505 (not found)

2. header (variable length, human readable): ocation (for redicts), content-length, content-type, expires (for caching), last modified

3. body: content of response

HTTP Cookies

1. HTTP is a stateless protocol: each request is treated independently

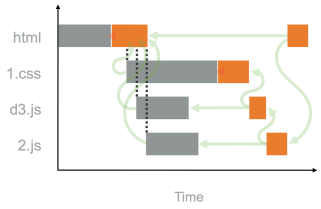
   (a) Simplifies failure handling and makes scaling easier. But: some applications need state (e.g. authentication, tracking) → Cookie:

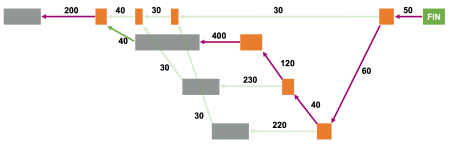
   (b) Client stores small state on behalf of server and sends the state in all future requests

2.2.3 Dependencies and Load Time

When loading an HTML Website, code is loaded from different sources with an HTTP request. Before processing subsequent request the previous one has finish first to prevent dependency problems:

```
<html>
    <body onload="done();">
        <link src="1.css">
        <script src="d3.js"></script>
        <script src="2.js"></script>
        <div id="main"></div>
    </body>
</html>
```





The total load time is found by summing the load times along the critical path of the dependency graph:

1. Do topological sort on nodes (=tasks) (if u needs to be executed before v then u is before v in the ordering)

2. Get longest path by going through the sorted nodes and setting the finish time to the max of the nodes it depends on.

Critical Path Analysis: Speeding up a Task on the critical path will:

1. Speed-up the end-to-end process and/or:

2. expose a different critical path

Moreover the definition of load time depends on the context (i.e. time until last call/last visual change, etc.)

2.2.4 Improving Load Performance

1. **Simplify, restructure, redesign Web pages**
   (a) More efficient compression and image codecs (WebP)
   (b) In-Line JS/CSS
   (c) Tagging resources as asynch (therefore removing dependencies)
2. **Use faster computing devices**: Watch for Benchmark used: Sunspider popular for Javascript/Web applications
3. **Increase Network Bandwidth**: Only significant if bandwidth less then a few Mbps.
4. **Increase Network Latency (RTT)**
   (a) To fetch n objects in standard TCP ca. $2n * RTT$ is needed: n times TCP establishment + HTTP-request-response. Where RTT is the time to receive the answer after sending a request.
   (b) When fetching M objects concurrently (by establishing multiple TCP connections): $(2n/M)RTT$. But: Server-side burden, connections have to fightämong themselves for bandwidth.
   (c) Using persistent connections (i.e. not always TCP setup): $(n+1)RTT$
   (d) Asynchronous pipelining of requests and replies, one connection (Batch multiple requests into one TCP request to reduce number of packets): 2 for small objects that fit into one TCP request.
5. **Caching** I.e. storing often used data/request-response pairs (DNS-requests).
   (a) Locations: client (Browser cache), close to client (forward proxy, Content Distribution Network), close to destination (reverse proxy)
   (b) forward proxy: Multiple clients access forward proxy (which is near them) **before accessing the rest of the network.** A forward proxy could be the ISP or an enterprise and speeds up responses to clients/prevents network overload.
   (c) reverse proxy: Each server has a reverse proxy which is accessed **before accessing the main server structure**. Therefore it's from the content provider and prevents server overload from repetitive requests.
   (d) Validation: Server hints when an object expires, client conditionally re-request resources with ïf-modified-since"header → server only responds with object if modified.

3 Transport Layer

3.1 What should the transport layer provide?

1. Connects Application- and Network layer:
   (a) **Applications:** should be restricted to app-specific functionality, want convenient way of using the network
   (b) **Network**: should to be minimal and restricted to finding path from A to B (routing) + best effort packet delivery
   (c) The Transport Layer should therefore provide an **interface for (reliable) transport over unreliable medium** to relieve burden from Application and Network
   (d) The transport layer is part of the host networking stack, i.e. the networking code is located on the host and shared by the applications.
2. Key requirements:

(a) **Data delivery to the correct application:** Multiple applications on a host may use a network interface: Transport needs to demultiplex incoming data (with ports)

(b) **Byte-streams abstractions for applications**: Network deals with packets but application needs different kind of input: Transport layer needs to translate between them

(c) **Reliable transport**

(d) **Prevent overloading**: the receiver (flow control) and the network (congestion control).

3. General idea: Send packet → receive acknowledgement. If not acknowledgement for packet is received in time: re-send.

3.2 Providing reliable transport

3.2.1 What is reliable transport?

There are four main problems for reliable transport over a network: Data packets can **get lost, corrupted, reordered or duplicated**. Therefore providing reliable transport means providing:

1. **correctness**: data is delivered, in order and unmodified
2. **timeliness**: minimize time to transfer data
3. **efficiency**: use available bandwidth optimally
4. **fairness**: concurrent communication should be in some sense fair

3.2.2 Correctness

= "A packet is always sent again if the previous was lost or corrupted"

3.2.3 Timeliness and efficiency

1. **Send/Receive window**: Packets that can be send/received at one time. Once a an acknowledgement is received for a packet, it is removed from the send-window.

   (a) Send-window-size = 1: send one packet, wait for an acknowledgment, send next, ... (untimely, inefficient)

   (b) Send-window-size > receive-window-size: receiver overwhelmed, drops packets

   (c) Flow control: keeping send-window-size ≤ receive-window-size

   (d) Optimal send-window-size (assumuming ∞ rec-window-size): For throughput T and delay d we can send dT packets bytes per time delay. As it takes d seconds to know if the packets were received we send another dT packets in that time. Thus the optimal send-window-size is $2dT$.

2. **Timeout time**:

   (a) Small timers: better timeliness if packet was lost, but less efficiency, if packet was received by ack has just not arrived yet.

   (b) Large timers: worse timeliness because longer wait times.

3. **Acknowledgement strategies**:

   (a) **Individual ACKs:** For each received packet send separate acknowledgement (e.g. received packet1, received packet2, received packet3)

      i. Missing packets are implicit (can infer missing packet by identifying gaps)

      ii. Advantages: know fate of each packet, simple window algorithm

      iii. Disadvantages: loss of ACK packet causes unnecessary retransmission (if the the packet is received and the ack response is lost, then the packet needs to be resend)

(b) **Cumulative ACKs**: Send ACK for multiple continuous arrived packets (e.g. received packet-1 to packet-5) after each arrived packet. If packet is lost the counter doesn't go up (received up to: 1, 2, 3, 4, 5, 5, 5, ...)

i. Advantage: Less prone to ACK loss.

ii. Problem: After detecting packet loss: which packets need to be resend?

(c) **Full information ACKs**: Send multiple cumulative ACKs in one ACK (e.g. received packet-1 to packet-5 plus packet-7 to packet-9: could infer that packet-6 is lost).

i. Advantage: Prevent unnecessary retransmissions

ii. Problem: Overhead (hence lower efficiency) because of long ACK packets.

4. **Reordering, duplication, delay and corruption of packets**

(a) Reordering. (reordering of packets can be confused for loss and cause unnecessary retransmssion):

i. individual: ACKs are sent out-of-order as well

ii. cumulative: duplicate ACKs are send

iii. full: gapped ACKs

(b) Duplication of packets (ACKs are send multiple times).

i. individual and full: no problem

ii. cumulative: duplicate ACKSs could be confused with packet loss

(c) Delay. Can create useless timeouts for all strategies

(d) Corruption. On detection of corruption (checksum doesn't match): treat packet as lost

5. **Sender's reaction to feedback (ACKs) = Retransmission algorithms**

(a) Go-Back-N: Receiver as simple as possible: upon inferred loss, retransmit everything not acknowledged yet

i. receiver: acknowledge last in-order packet received (cumulative)

ii. sender: use a single timer to detect loss, reset at each ACK upon timeout, resend all packets in window, starting with the lost one: no reaction to duplicate ACKs

(b) Selective-Repeat: avoid unnecessary retransmissions

i. receiver: acknowledge each packet, in-order or not (individual), buffer out-of-order packets

ii. sender: use per-packet timer to detect loss, upon loss only resend lost packet, no reaction to ACK gaps

3.2.4 Fairness

= properly"deal with concurrent communication (not properly defined, minimal goal is to avoid starvation)

1. equal-per-flow: Each flow through a link gets the same bandwidth (same #packets per second)

(a) Effectiveness: Not always effective (bandwidth across the network is not necessarily maximised)

(b) Fairness: If a connection goes through multiple links while getting the same bandwidth as others, its total bandwidth usage is bigger + impossible to divide total bandwidth

2. Max-Min allocation: the lowest demand is maximized, if satisfied: the second lowest demand is maximized, if satisfied: ...

(a) Start with all flows at rate 0

(b) Increase the flows simultaneously until there is a new bottleneck in the network

(c) Hold the fixed rate of the flows that are bottlenecked

(d) Go to step 2 for the remaining flows

3. Max-Min allocation with sending-window:
   (a) Progressively increase the sending window size (max = receive-window-size)
   (b) Whenever a loss is detected, decrease the window size (signal of congestion)
   (c) Repeat.

3.3 UDP

Basic transport-layer functionality:

1. Takes: SRC port, DEST port, checksum, length, data
2. Gives: Data delivery to correct application with demultiplexing with ports, optionally check correctness with checksums, no delivery guarantee.
3. Why use UDP:
   (a) No overhead. No need to ensure reliable transport (ordering, etc), small header per packet
   (b) No delay. for connection establishment
   (c) Control. Application can control when what data ist send
   (d) Scalability. No connection state, buffers or timers
   (e) Good for. DNS, Gaming, VoIP

3.4 TCP

3.4.1 What TCP provides

1. Reliable, in-order delivery. Through:
   (a) ACKs. Carry byte sequence numbers, cumulative
   (b) Timer. Resend on timeout, double the timer value
   (c) Fast retransmit. On three duplicate ACKs
   (d) "connections" and associated state at end hosts
2. Flow control: Sliding window of size no larger than receiver wants
3. Congestion control: Dynamic adaptation to network-path capacity

Therefore its header contains the following:

1. Source Port, Destination Port
2. Sequence Number, Acknowledgement Number
3. HdrLen, 0, Flags, Advtised Window,
4. Checksum, Urgent Pointer
5. Options (variable)

3.4.2 TCP Segments, Sequence Numbers and Acknowledgment

1. Definition Segment. Sender collects Bytes in a segment and sends the segment to the receiver if it's full or after a predefined timeout time.

2. Size of Segment. Segment + TCP header (20 bytes) is transported through an IP packet with MTU (maximum transmission unit) and an IP header. Thus

$$MSS = MTU - TCPHeader - IPHeader$$

3. Sequence Number. Number of first byte in Segment. If the segment contains B bytes and has sequence number X, then it has bytes $X, X+1, \ldots, X+B-1$

4. Acknowledgement Number. If the receiver has received packets contiguously up to $X+B-1$ the acknowledgment number is $X+B$ i.e. the next expected byte. (duplicate ACKs allowed)

3.4.3 Advertised Window

1. Defintion. In the acknowledgement the receiver sets the advertised window W to the number of bytes it can process. Therefore the sender can only send W bytes beyond the next expected byte. Only after receiving an acknowledgement with another W, it can send more: Flow control

2. Transfer rate. Let B be the bandwidth of the connection. If $W/RTT < B$: transfer speed W/RTT else transfer speed B.

3. How to obtain the Advertised Window.

   (a) Send buffer. The sender maintains a send buffer containing all sent but not acknowledged packets:

   $$\{\text{ackno (last acknowledged byte + 1)}, \ldots, \text{number of last sent byte}\}$$

   (b) Receive buffer. the receiver maintains a receive buffer containing all received packets not processed by the application. The receive buffer has a maximum size, thus:

   $$|\{\text{Last byte read + 1}, \ldots, \text{Last received packet}\}| \leq \text{BufSize}$$

   (c) Determining size. Therefore

   $$W = \text{BufSize} - (\text{last received byte} - \text{last byte read})$$

   When more data is read by the application, W increases, if data arrives it gets smaller. To ensure that the receiver has always enough space in the buffer the sender makes sure, that:

   $$\text{last-byte-sent} - \text{ackno} \leq W$$

   (d) Problem $W = 0$. If $W = 0$ and the receiver doesn't send any data on its own, the sender will never know if $W > 0$, as it doesn't send data and therefore doesn't receive acknowledgements. To solve this the sender continues to send small packets without any data which the receiver acknowledges with an updated W.

3.4.4 HdrLen, 0, Urgent Pointer

1. HdrLen. Number of 4-byte words in header; If 5 words: options field empty

2. 0. 6 reserved bits set to 0

3. Used with a flag to indicate urgent data

3.4.5 TCP Connection Establishment

When establishing a connection TCP follows three steps (three way handshake):

1. Sender sends TCP packet SYN: SYN flag set to 1 and randomly chosen sequence number (initial sequence number: client-isn)
2. Receiver returns SYN-ACK: acknowledgement number set to client-isn + 1, SYN flag set to 1 and randomly chosen sequence number: server-isn
3. Sender acknowledges ACK: acknowledgement number set to server-isn + 1

On loss of SYN packet (discarded or not delivered):

1. Sender does not receiver SYN-ACK packet, eventually sender retransmits. When is difficult to determine as receiver can be anywhere. Default is 3s.
2. User can force retransmit (e.g. reloading the page)

There are security issues with the handshake:

1. For multiple clients the server needs to store the pair (server-isn, client). Therefore by sending many syn packets the server can be exhausted.
2. Defense by SYN cookies (idk how the hash thing works)

3.4.6 TCP Connection Teardown

1. Normal: Sender sends packet with FIN flag set, receiver sends ack packet → receiver sends FIN packet, sender acks
2. Compact: Sender sends packet with FIN flag set, receiver sends packet with FIN flag and ACK flag, sender acks
3. Abrupt: Sender sends packet with RST flag set (if e.g. the application crashed), if it receives any more data, it sends packet again. Thus if RST is lost, the data that the server sends afterwards is lost as well (it cannot be handled by the sender anymore)

3.4.7 TCP Retransmission

TCP uses a combination of selective repeat and go-back-n:

1. Timer. TCP uses one timer that is started when the first packet is sent or after an acknowledgement is received. (e.g. if packets 1,2,3,4 are sent and an acknowledgement for 1,2 is received the timer is reset to 0). If the timer „goes off" the (one, due) packet is resent.
2. Timeout. Ideally the timeout time should be the round-trip-time (RTT) (i.e. the time needed to send a packet and receive its acknowledgement). Is the timeout larger than RTT time is wasted waiting, while if the timeout is smaller the package is necessarily resent, because there was no time for the ack to arrive. It is however impossible to know RTT and thus it has to be estimated:
   (a) Exponential Averaging:

   $$SampleRTT = AckRvdTime - AckSendTime$$
   $$EstimatedRTT = \alpha EstimatedRTT + (1 - \alpha) SampleRTT$$

   Problem: If packet is resent there is no way to differentiate between, „real" ACK of the initial packet and the new ACK.
   (b) Karn/Partridge Algorithm: Ignores times if a packet has been resent. Then uses exponential averaging with $\alpha = 0.85$. Then (exponential backoff):
      i. Set timeout value: $RTO = 2EstimatedRTT$

ii. If RTO expires: $RTO \leftarrow 2RTO$ (up to $60s$)

iii. On successful original transmission (the only time a new measurement comes in): $RTO \leftarrow 2EstimatedRTT$

Problem: $EstimatedRTT$ varies

(c) Jacobson/Karels Algorithm:

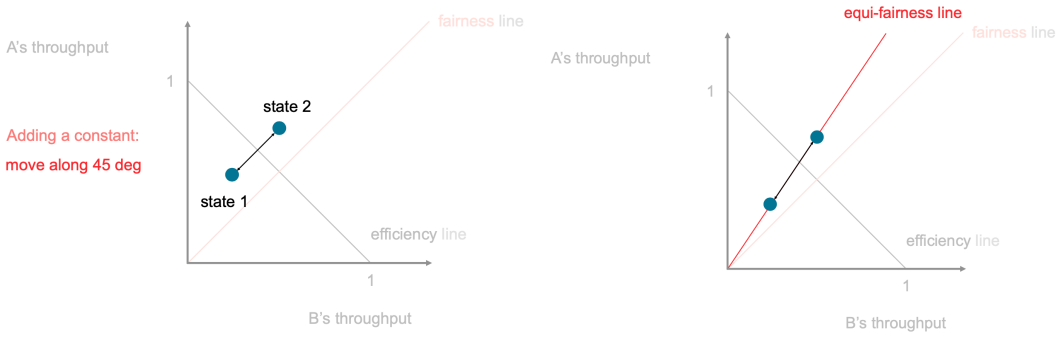
$$Deviation = |SampleRTT - EstimatedRTT|$$
$$EstimatedDeviation = ExpoentialAverage(Deviation)$$
$$RTO = EstimatedRTT + 4EstimatedDeviation$$

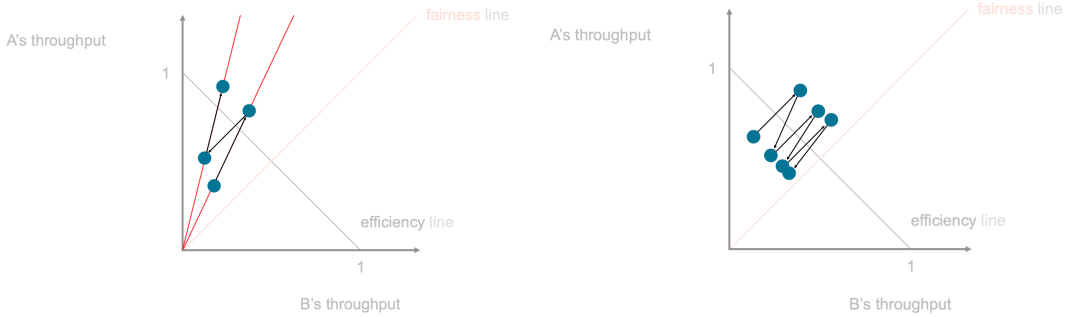
3. Loss: TCP uses cumulative ACKs. E.g. for 100B packets and sequence numbers $100, 200, \ldots$: Acknowledges up to a packet number. If ackno is repeated multiple times, it means that packets are still arriving, but the repeated one hasn't: TCP resends packet after 3 repeats.

3.4.8 TCP Congestion Control

Congestion control prevents a set of senders from overloading the network. TCP uses a congestion window. Then the sending window (i.e. the maximum number of bytes send without acknownowledgment) is defined as min(congestion-window, rec-window).

1. Detecting congestion
   (a) Following solutions aren't possible: network tells source (signal could be lost), measure packet delay (signal is noisy, delay varies)
   (b) Therefore: Measure packet-loss: duplicated ACKs (mild congestion signal: some packets still arrive), timeout (severe congestion control: packets get lost)
2. Bandwidth estimation: How to adapt sending rate?
   (a) Intuition: increase congestion window until packet loss is detected, then decrease window until not and so forth
   (b) In practice the increase/decrease is either additive ($val = a + val$) or multiplicative ($val = b * val$). Therefore there are four options for adapting the window (AIAD, AIMD, MIAD, MIMD)
3. Fairness. TCP chooses the bandwith estimation method by a notion of fairness: *two identical flows should end up with the same bandwidth.* The following can be observed:

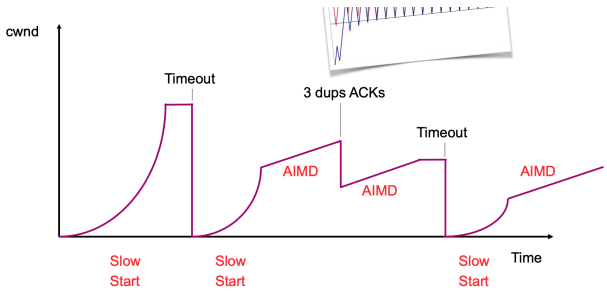




In AIAD and MIMD (top) the system never leaves a position with an unequal share of bandwidth, while MIAD (bottom left) even favors the flow with the greater rate at the beginning (goes to sides). Only AIMD (bottom right) converges to fairness and efficiency (goes to the middle). Therefore TCP implements AIMD.

4. Bringing it all together.
   (a) Slow start. Start with congestion window ($cwnd = 1$) for each received ack: $cwnd = cwnd + 1$. Thus there is an exponential increase of $cwnd$ (e.g. 2 packets get two acks thus $cwnd$ is doubled)
   (b) Congestion avoidance. Once the congestion window is bigger than some threshold ($cwnd > ssthresh$) TCO does AIMD, which is implemented in the following way:
      i. On ACK: increasing with $cwnd = 1/cwnd + cwnd$. This is additive increase.
      ii. On timeout: Decreasing by setting the threshold to half of the current congestion window ($ssthresh = cwnd/2$) and then resetting $cwnd = 1$. This equals multiplicative decrease, as the value that is reached by slow start is halved.
      iii. Fast retransmit. Other than the timer reaching 0 a timeout occurs if a sender receives 3 duplicate ACKs. This is called „fast retransmit"
      iv. Fast recovery. After a fast retransmit instead of using slow start, $cwnd$ is immediately set to $sshresh$. The sender then proceeds with AIMD.

The following graphic can be obtained:



3.4.9 Other TCP Congestion Control Algorithms

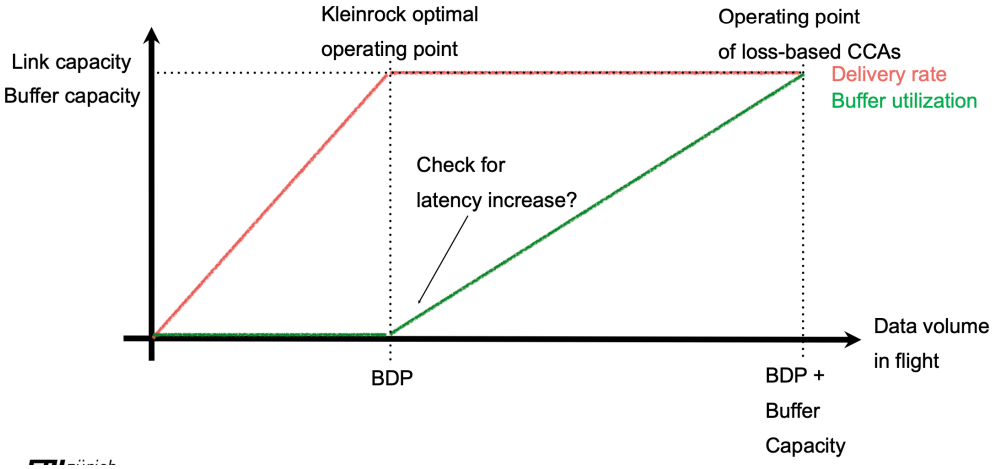
1. TCP Reno. Same as before with:
   (a) Additive increase with $cwnd = 1/RTT + cwnd$, multiplicative decrease with $cwnd = 0.5cwnd$. This means that $cwnd$ is increased by 1 every RTT.
   (b) Detects congestion with Loss (loss-based CCA)
   (c) Problem: Different RTTs: Senders with slower RTTs will increase bandwidth more slowly and therefore obtain less bandwidth.
2. TCP Cubic. Same as before, but different Congestion Avoidance

(a) Keep state s (time since timeout) and w_{max} (window size at timeout). Then define window size as a continous function of s: $cwnd = w_{max} + 0.4(s - (0.75w_{max})^{1/3})^3$, i.e. the longer the timeout (bigger s) the larger the window.

(b) Thus TCP Cubic independant of RTT

(c) Problem: Bufferbloat: TCP cubic increases *cwnd* at the same rate until a loss occurs. At this point the buffers in the link are already full. This results in overloaded buffers and high latencies.

(d) This can be summarised with the following graph:



Loss-based CCAs operate in a state of maximum buffer utilization (i.e the data in flight is the sum of available bandwidth-delay-product and the buffer capacity) which leads to higher latencies. The optimum (called the Kleinrock optimal operating point) is in a state where the data is flight is just the available bandwidth-delay-product.

3. TCP Vegas

(a) Congestion avoidance (taking buffer capacity into account).

i. Estimate delay T as the minimum measured RTT

ii. Measure current RTT (T_{curr}), then:

iii. Choose $\alpha, \beta, \alpha < \beta$. If $cwnd/T_{curr} < cwnd/(T - \beta)$ (i.e. the current RTT is bigger than optimal RTT (delay)) linearly decrease *cwnd*. Else if $cwnd/T_{curr} > cwnd/(T - \alpha)$, increase *cwnd* linearly

(b) Problem: As TCP Vegas sends less if there is no space in the buffer, it also sends less than Loss-Based CCAs and is therefore uncompetitive.

4. TCP BBR

(a) Congestion control. Keep state $BtlBw$ and $MinRTT$. Repeat periods, where each periods consists of 8 phases. Every phase has a duration of $MinRTT$. In one period data is sent at a rate of $BtlBw$ until in one randomly chosen phase $BtlBw = 1.25BtlBw$ and in the next $BtwBw = 0.75BtlBw$. After the period set $BtwBw$ to the maximum observed delivery rate.

(b) Problem: Better than loss-based CCAs, because the random probing of TCP BBR causes loss which results in a smaller congestion windows in loss-based algorithms.

3.5 QUIC

3.5.1 Why QUIC?

1. Application context has changed: Webbrowsing

2. TCP handshakes are too expensive

3. TCO doesn't handle mobility (e.g. changing networks while using the same server)

4. Difficult to evolve TCP (middleboxes, firewalls block different looking headers)

3.5.2 Application context

1. Head-of-line blocking. Assuming multiple datastreams through one TCP connection: If one stream halts (i.e. the next due packet is not send) all other packets belonging to other application have to wait, too (=Head-of-line blocking)

2. In QUIC there is not head-of-line blocking. Therefore multiple streams can send independently

3.5.3 Handshakes

1. Traditional TCP. For every connection to a server TCP needs to make TCP and TLS handshake. Even if the connection has already been established once → unnecessary repeated handshake.

2. QUIC. Combined TCP and TLS handshake into QUIC handshake and stores transport cookie. All consequent connection can immediately send data to the server and authenticate themselves with the cookie: Not need for handshake.

3. Problem. Replay Attacks: Copying a sent packet and sending it again, possibly having code be executed multiple times (e.g. bank transfers). This is not possible in TCP, as data can only be sent between client and server. The client cannot be imitated because of the handshake.

3.5.4 Mobility & Evolve TCP

1. As seen in the section before a TCP connection to a server is specific to a source IP (client)

2. However in UDP the source doesn't matter. It can deliver packets with just the destination IP.

3. Therefore QUIC is build on UDP and puts the missing header information into the data section of UDP. Instead of using IP as identification QUIC uses connection IDs which are randomly initialised during the inital handshake

4. Moreover a portion of the data section of the UDP packet is not encrypted so that QUIC header information can be acessed.

3.6 Sockets

3.6.1 What are Sockets for?

1. If an application wants to send packets through the network (i.e. use the transport layer of the operating system) it uses a socket.

2. Therefore a sockets translates between (multiple) applications and transport layer.

3.6.2 Ports

1. Problem. Multiple applications, one stream of incoming data packets. How to map data packet to application

2. Solution. Ports: Each application gets unique port, operating system stores this relation. Incoming packets have port in header.

3. Sending and receiving data.

   (a) As known from the previous section, both UDP and TCP have dest and src port in the header.

(b) Applications running on servers have known, agreed upon ports in range 0-1023 (e.g. https on port 443, http:80, ssh:22)

(c) The application running on the client side are randomly given ports in range 1024-65535.

(d) When communicating, the sender chooses the appropriate dest port and sends its own port to which it now listens. The receiver has an open port to which it always listens. When receiving a message it responds with the in the message received port.

3.6.3 TCP v. UDP

1. It is generally differentiated between UDP (Datagram Sockets) and TCP (Stream Sockets)
2. TCP. The OS stores (local port, local ip, remote port, remote ip) $\leftrightarrow$ socket
3. UDP. The OS stores (local port, local ip) $\leftrightarrow$ socket

4 Network Layer

4.1 Introduction

1. Network Layer provides service for transport layer and builds on link layer.
2. Main question: How do packets reach their destination, two areas: Routing and Forwarding
   (a) Routing: process of deciding in which direction to send traffic (control plane of network): Network wide (global) and expensive
   (b) process of sending a packet on its way (data plane of network): Node process (local) and fast

4.2 Network Service Models

4.2.1 Store and Forward Packet Switching

Question: What kind of service does the Network layer provide to the Transport layer? There are two Models:

1. Datagram Model (connectionless)
2. Virtual Circuits (connection-oriented)

Both use a concept called Store-and-Forward Packet Switching, which works as follows:

1. Network is built from routers, which receive a complete packet, store it temporarily (if necessary) and then forward it
2. One router has two buffers: One input buffer and one output buffer. After a packet arrives, it is stored in the input buffer, then through some logic (*fabric*) forwarded to the output buffer from which it is relayed to the output.
3. Each buffer is usually a FIFO queue. Once full, it discards incoming packets.

4.2.2 Datagram Model

1. Idea.
   (a) Each packet contains a destination address.
   (b) Each router forwards the packet. However two packets with the same source and destination don't necessarily take the same path.
   (c) The forwarding happens with a table that is keyed by the destination address and gives the next hop (i.e. path to go along). A table entry may change over time.
2. IP. Is in the network layer. It uses the Datagram Model and thus carries dest and src address in each packet.

4.2.3 Virtual Circuits

1. The Virtual Circuit consists of three phases:
   (a) Connection establishment (building the circuit): Choosing a path, store path in routers
   (b) Data transfer (using the circuit): Forward packets along path
   (c) Connection teardown (deleting the circuit): Remove circuit information from routers
2. The forwarding happens with a table that is keyed by a label unique to the established circuit and different per link. The table gives the next link and new label for the packet.
3. MLPS (Multi-Protocol Label Switching). Often used by ISPs:
   (a) Set up circuits inside ISP backbone ahead of time
   (b) If IP packet enters backbone, add MPLS label to it, use the circuit in backbone and remove label when the packet leaves the backbone.

4.2.4 Datagram v Virtual Circuit

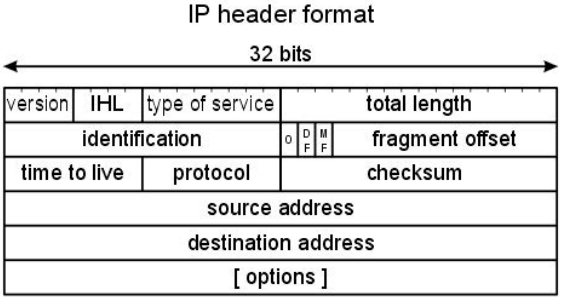
Issue	Datagrams	Virtual Circuits
Setup phase	Not needed	Required
Router state	Per destination	Per connection
Addresses	Packet carries full address	Packet carries short label
Routing	Per packet	Per circuit
Failures	Easier to mask	Difficult to mask
Quality of service	Difficult to add	Easier to add

4.3 IPv4

4.3.1 Internetworking & IP Headers

1. Problem: Different Networks may differ significantly:
   (a) Service Model (VC/Datagrams), Addressing, QoS (Quality of Service. e.g. are some packets prioritised), packet size, security
2. IP was built to be the common link: Asks little of link-layer, but gives little to the transport layer: Broad support for different technologies.

Therefore the IP-Header supports many functionalities:



1. Version, Header (IHL) and Total length, Protocol, and Header Checksum: General requirements
2. Source address, Destination address: Provides a layer of addressing above link addresses (IP is a Datagram Model)
3. Identification, Fragment offset, Fragment control bits: Handle packet size differences

4.3.2 IP Addresses and Prefixes

1. Structure. An IPv4-address consists of 32 bits, which are separated by dots into 4 groups of 8 bits, i.e. A.B.C.D
2. Modern IP Prefixes.
   (a) Definition. Given a number $0 \leq L \leq 32$. Two IP-Addresses belong to the same prefix of length L if their first L bits are equal. Thus the number of prefixes of length L is (L bits are equal and $32 - L$ can be chosen freely):
   $$\#\text{Prefixes of length } L = 2^{32-L}$$
   (b) Notation. The prefix-address is lowest number of all addresses in the prefix (i.e. the first L bits followed by 0s) with the length of the prefix separated with a /. E.g. 128.13.0.0/16 is 128.13.0.0 to 128.13.255.255. Obviously a bitmask can be used to obtain the prefix.
   (c) Meaning. Therefore a prefix is more specific if L is large and less specific if L is small.
   (d) Prefixes and Hosts. An IP prefix of length L is used to refer to a network. The remaining $32 - L$ bits can be used to refer to hosts within the network.
      i. However the smallest address in prefix (ending with 0s) is used to **refer to the network itself**
      ii. and the largest (ending with 1s) is used to **send data to all hosts (broadcast)**.

   Thus it is possible to distinguish between $2^{32-L} - 2$ hosts.
3. Historical IP Prefixes. Previously, instead of arbitrarily choosing L, prefix classes were defined that mapped to a number of prefix-bits (e.g. class A meant $L = 8$ and 2^{24} addresses).
4. Public/Private IP Addresses.
   (a) Public. IP address used for global internet, must be unique and specifically allocated
   (b) Private. IP addresses used in a private network (e.g. home, company). To connect to the worldwide internet public IP+NAT is needed (e.g. 192.168.0.0/16 by RFC 1918)

4.3.3 IP Forwarding (Longest Matching Prefix)

1. General. IPv4 uses IP-Prefixes as keys for the forwarding-table
2. Overlapping of prefixes (longest matching prefix).
   (a) The prefixes used as keys might overlap (i.e. one might contain the other).
   (b) Therefore the most specific IP-prefix (i.e. the one with the largest L) which includes the destination address is chosen as the key
3. Compressing Forwarding Tables. If one prefix doesn't change the behaviour of the routing table (e.g. it is contained in another and forwards to the same hop) it can be removed.
4. Host/Router distinction: A host sends out all traffic not in the own network (=remote traffic (out of prefix). The router knows where to forward it.
5. Host forwarding table. Forwarding table at host that decides where to send a packet first. (e.g. 0.0.0.0/0 catches all IP-addresses and is default)

6. Flexibility. Can provide default with less specific prefixes and special case behaviour with specific prefixes.
7. Performance. Finding the longest matching prefix is more complex than normal table lookup. Got fast due to ternary content-addressable memory (TCAM). However TCAM consumes a lot of power.
8. Other things forwarding can do. Decrement TTL (preventing loops), check checksum (add reliability), fragment large packets (to fit on link), send congestion signals, generate error messages, ...

4.3.4 IP Helper Protocols: ARP and DHCP

1. **Getting the IP-address: DHCP**
   (a) Problem: When a device is first connected to the internet, what **IP does it have**, what network does it belong to (i.e. what is the **network ip prefix**?), **where it the router** to send data to by default (what is its ip)? Where is the **DNS server**?
   (b) Dynamic Host Configuration Protocol (DHCP):
      i. Application Layer with UDP and Ports 67, 68
      ii. New node sends message to all devices with a broadcast by setting the destination ip to 255.255.255.255 (and mac address to all 1s as well) Routers are configured not to propagate the message outside the network.
      iii. DHCP-servers that receive the message (DISCOVER) reply with and OFFER packet (network prefix, router ip, a free IP address, etc.)
      iv. The node responds with a RESPONSE packet, announcing its IP, which then the DHCP server acknowledges with an ACK-packet
      v. To renew the IP the node has to repeat the REPONSE-ACK process.
   (c) Problems: IP-address conflict (multiple devices in network have same ip)
2. **Getting destination MAC-address: ARP**
   (a) Problem: A node needs a Link layer address (MAC-address) to send a frame over the local link. However it only has the destination IP-address of a target-device.
   (b) Address Resolution Protocol (ARP):
      i. ARP on top of link layer.
      ii. ARP request (with IP of the target included) is sent with broadcast to all nodes in network
      iii. Target sees its IP in packet and sends a REPLY packet with its MAC-address.
      iv. Node stores (mac, ip) pair in ARP cache.
   (c) Other discovery protocols: zeroconf, Bonjour; some don't use broadcast.

4.3.5 Fragmentation

Main question: How much data can be sent at once (different networks have different maximum transmission units MTU)? Want to send as much data as possible.

1. Idea. If a packet arrives at router and it's too big to forward, **split in multiple smaller packets**. The **receiving host reassembles the packets** to reduce load.
2. IP-header fields (Identification, total length, fragment offset, MF bit) used to handle fragmentation
3. IP Fragmentation Procedure
   (a) Split packets so that they're as big as possible
   (b) Copy IP-header to each piece

(c) Adjust length of each piece and set offset to first piece.

(d) Set MF (More Fragments) bit in all pieces but the last.

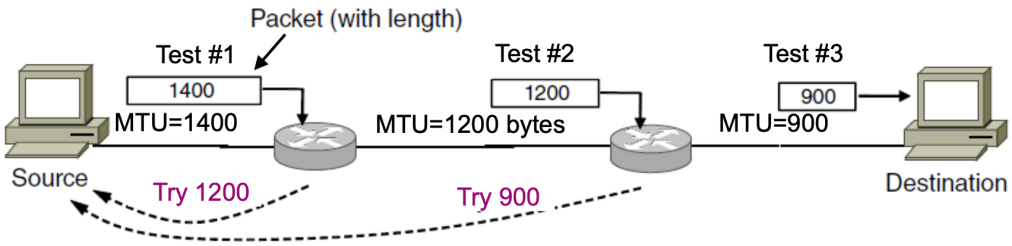
4. The receiving host can reassemble the pieces: Same values in the identification mean the pieces belong together and the MF-bit tells the host when there are not pieces left.

5. Problems: More work for routers, if one piece is lost, the entire packet needs to be resent, security vulnerabilities (DoS, DNS Cache poisoning, etc)

4.3.6 MTU Discovery, ICMP and Traceroute

Same Problems as before: We don't know how much data can be sent through a network. Before: split the packets if they don't fit. This time: find maximum amount of data that can be sent.

1. Idea. Host test path by sending large packet with DF (Dont Fragment) bit set. If router detects that the packet is too large, it tells the host the maximum size, which then sends a smaller packet.



2. Path can change over time: MTU discovery happens all the time.

3. **ICMP**. The Protocol used is the Internet Control Message Protocol (ICMP).

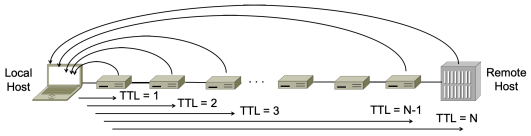
   (a) Layer: ICMP is in the Transport Layer and uses IP (has Protocol ID 1)

   (b) Functionality: If router detects error while forwarding it sends an ICMP-error back to the host (i.e. the IP-source address) and then discards the received packet.

   (c) Format: Type, Code, Checksum, Payload (contains beginning of IP-header of received packet); (Ping not a forwarding error, can be obtained by request)

Name	Type / Code	Usage
Dest. Unreachable (Net or Host)	3 / 0 or 1	Lack of connectivity
Dest. Unreachable (Fragment)	3 / 4	Path MTU Discovery
Time Exceeded (Transit)	11 / 0	Traceroute
Echo Request or Reply	8 or 0 / 0	Ping

4. **Traceroute**. Traceroute uses the Time Exceeded code to find the path the packet is sent through. It sends packets with an increasing TTL time starting at 1. If a router detects a timeout it responds with a ICMP packet containing the IP-address of the router



4.4 NAT

4.4.1 General Idea & Middleboxes

1. NAT box connects internal and external network. Internal hosts use connected to the external network with fewer external addresses.
2. A NAT box is a type of Middlebox: Middleboxes sit inside the network (where there usually routers), but add functionality on top of IP (e.g. with Transport and Application Layers)
   (a) Advantages: Control over many hosts + fast deployment path
   (b) Disadvantages: Causes internet ossification (Middleboxes often filter traffic): impossible to deploy new transport protocols + Breaks internet layering: affects connectivity
3. Usecase: Home network are in NAT: they use private IP-addresses for communication in the network. The NAT-box (the home router) connects all home devices to the internet with a single IP-address.

4.4.2 How NAT works

1. **Address and Port translation**. NAT device keeps table with (internal-ip, internal-port - external-ip, external-port). As multiple internal hosts might have the same internal-port, all external-ports are different from the internal-port and distinct (otherwise two hosts might have the same external-port and ip)
2. **From inside to outside**: Internal host sends packet; NAT box receives packet, reads source ip and port and looks for them in the table; It finds an entry with the corresponding external ip and port (or creates one) and then sends a new packet with the external port and ip as source
3. **From outside to inside**: External hosts sends packet; NAT box receives packet and looks and reads destination ip and port and looks for them in the table. If it doesn't find an entry the traffic is blocked, otherwise it can be forwarded to the local ip and port
4. **Port Forwarding** To deploy a webserver incoming traffic on a specific port (e.g. 80/443) needs to be forwarded to private network address.
5. **Downsides**. Broken Connectivity (can only receive packets after address translation is set up) + Issues if no explicit connection is set up (e.g. UDP) (often fixed by regularly sending useless packets to keep a table entry (keep-alive-packets)) + breaks some apps
6. **Upsides**. Relieves IP address pressure + Easy to deploy + useful functionality (firewalls, hides internal network structure, own address space)

4.5 IPv6

4.5.1 IPv6 Header

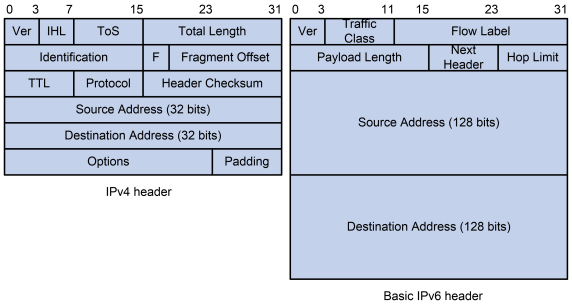
1. New fields:
   (a) Flow Label: Groups and Identifies packets belonging to one flow
2. Modified/renamed fields:
   (a) TTL → Hop limit
   (b) 32-bit addresses → 128-bit addresses
   (c) Total Length → Payload Length
   (d) Differentiated Services/TOS → Traffic class
   (e) Protocol → Next header
3. Removed fields:

(a) IHL: No longer needed (header size fixed)

(b) Fragmentation bits, offset, identification: No fragmentation supported

(c) Header checksum: too time-consuming, enough to check them at link and transport layer

(d) Options: Can instead indicate separate extension headers with next-header



4.5.2 IPv6 Addresses & Prefixes

1. **Notation**. 8 groups of 4 hex digits (16 bits), omit leading 0s in each group with ::. Omit Longest groups only consisting of 0s with ::.

2. **Organization**. First bits define routing prefix (identifies responsible isp), next bits define subnet id (subnet or customer of isp), last 64-bits define interface identifier (can be based on mac-address).

3. **IPv6 Unicast Addresses**. An IPv6 host can have multiple IPv6 addresses

   (a) Link-Local: fe80::/10 (or fe80::/64). Every host has a Link-Local address based on its MAC-address

   (b) GUA (global unique address): (currently 2000:/3. Globally reachable, the normal IPv6

   (c) ULA (unique local address): (fc00::/7) For local deployments (like RFC 1918 192.168.0.0/16). With NAT translated GUA.

4. **Special IPv6 Addresses**. Loopback address: ::1/128 (like 127.0.0.1), Unspecified address: ::0/128, special addresses for transition mechanisms, etc

4.5.3 ARP and DHCP in IPv6: NDP

DHCP is replaced with:

1. Router Solicitation: Host sends router solicitation message to find routers.

2. Router Advertisement: When router receives a router solicitation message (or just periodically) it sends a message with various link and internet parameters (e.g. global ip, default gateway).

ARP is replaced with

1. Neighbour Solicitation. Message send to determine link address of neighbour.

2. Neighbour Advertisement. Answer to neighbour solicitation.

Moreover NDP provides:

1. On startup host automatically sets link-local IPv6 address

2. Duplicate Address Detection possible

4.5.4 NAT with IPv6

Still possible to do NAT with IPv6, not necessary because there are enough addresses. However still useful to block incoming traffic

4.5.5 IPv6 Transition

How to deploy IPv6: Incompatible with IPv4

1. **Dual Stack**: support IPv4 and IPv6 in parallel.
   (a) Problem: Contact IPv4 or IPv6 address?
   (b) Solution: Do both. Issue A (IPv4) and AAAA (IPv6) DNS queries simultaneously. If AAAA record received, start IPv6 connection. If A record received, wait some time for AAAA record, then start connection with what is available. If connection fails with any record, try the other one.
2. **Tunneling**: Send IPv6 over IPv4. Problem: Setting up tunnel endpoints and routing the packet

4.6 Routing Algorithms

4.6.1 Goals & Setting of Routing

1. Correctness: Find the correct path
2. Efficiency: Find best path (i.e. one which minimizes some metric)
3. Path Fairness: No nodes are starved (eventually all nodes receive data)
4. Fast Convergence: fast recovery after change
5. Scalability: Works well as network grows large

Setting: Decentralized, Distributed

1. All nodes are equal (i.e. there is no node that controls the others (controller))
2. Nodes only have information from message their neighbours send them
3. Nodes operate concurrently

4.6.2 Dijkstra

1. **Setting.** Each link has a cost (distance). We define the best path between two nodes as the path with the smallest cost (if equal choose randomly).
2. **Algorithm.**
   (a) Save list of distances d to each node. Initialize distance from source to 0, all others to ∞. List of unvistited nodes
   (b) While list of unvisited nodes v not null:
   - From v choose node N with smallest distance in d.
   - Add link to N to the shortest path tree
   - Update distances in d by checking if any of them are shorter when using the new link (Thus only checking neighbours of N).
3. **Discussion.**
   (a) Dijkstra finds shortest path to all nodes. The further they are away the later they are found. $\rightarrow$ fulfils efficiency goal.
   (b) But: Runtime depends on finding the node with the smallest distance (superlinear in network-size $\rightarrow$ large) + Gives complete sink/source tree (more than is actually needed for forwarding) + requires complete topology/knowledge of network

4.6.3 Hierarchical Routing

1. **Concept.** Multiple routing units: IP-prefixes (hosts) and region (e.g. ISP network): Frist route among regions, then inside the specific region.
2. **Observations.**
   (a) Outside a region there is one path to all the hosts within the region → saves space & computation time
   (b) However: Each node can choose its own way to a region (and its hosts). There is not central unit for a region determining the routing.
3. **Irl**. Regions defined by IP-prefixes: routers keep information about close destinations in more detail (i.e. more specific ip-prefixes) and information for far destinations with less detail (i.e. less specific ip-prefixes):
4. **Subnets and Aggregation.**
   (a) Subnets: Internally split one network into multiple smaller subnetworks by internally splitting up a less specific ip-prefix into multiple more specific ones (i.e. the internal subnets split the available prefix range of the entire network)
   (b) Aggregation: Summarize/join multiple separate ip-prefixes to a less specific one and advertise it to the rest of the internet.
      i. Optimize the local routing table (e.g. compressing forwarding tables)
      ii. Decide which prefix to advertise (must make sure no one else can advertise a less specific prefix that "hides" it)
5. **Different types of routing**
   (a) Intra-Domain routing. Intra-Domain protocols (within an autonomous system): interior gateway protocol (IGP)
   (b) Inter-Domain routing: Routing between multiple ASes: exterior gateway protocol (EGP)

4.6.4 Distance-Vector Routing

Setting:

1. Nodes know only the cost to their neighbors; not the topology
2. Nodes communicate only with their neighbors using messages
3. All nodes run the same algorithm (distributed Bellman-Ford) concurrently
4. Nodes and links may fail, messages may be lost

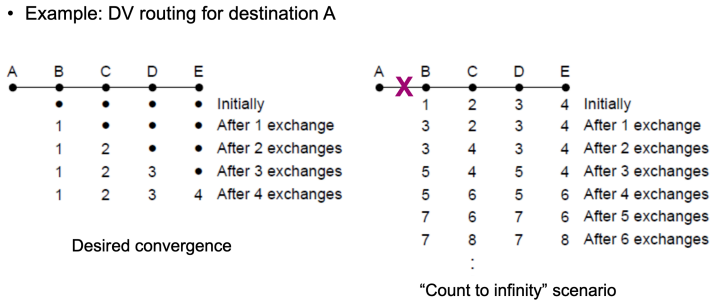
Algorithm:

1. Each node has a vector of distances d (and and next hop) to all destinations.
2. Initialize d with 0 cost to itself and ∞ cost to all others
3. Periodically send vector to neighbors
4. After receiving vectors of all neighbors:
   - For neighbor i and cost of link to it c_i: add c_i to the received vector of neighbour i
   - For each entry in the own vector: Set to the minimum of the corresponding entries in the updated received vectors

Dynamics:

1. Adding routes: Updates travel one hop per exchange: after n rounds every node knows the shortest paths up to n hops. → $\#nodes - 1$ rounds

2. Removing routes: When node gone there are no more exchanges → other nodes forget
3. Partitioned Networks: Some nodes unreachable → count to infinity, doesn't converge



Solutions:

(a) Heuristics: *Split horizon* (don't send routs to neighbor you learned them from???), *poison reverse* (routes sent back to neighbor (that were learnt from it) have metric ∞)

(b) Don't use DV but rather Link State

4. **Routing Information Protocol (RIP)**

(a) DV with hop count as metric (cost): Infinity is 16 hops + uses split horizon & Poison reverse

(b) Vectors are sent every 30 seconds + runs on to of UPD (application layer???)+ timeout of 180 seconds

4.6.5 Flooding

1. Send message to all nodes in network:
2. Sends incoming message to all neighbors, if message (identified by source, sequence number) not already received before (duplicate suppression)
3. Receiver acknowledges (sender possibly resends: stop-and-wait)

4.6.6 Link-State Routing

Setting (same as DV):

1. Nodes know only the cost to their neighbors; not the topology
2. Nodes communicate only with their neighbors using messages
3. All nodes run the same algorithm concurrently
4. Nodes and links may fail, messages may be lost

Algorithm:

1. Nodes flood network with information about their topology (i.e. (the costs to) its neighbors) in a Link-State-Packet (LSP). Thus each node knows the full topology of the network.
2. Each node computes its own forwarding table (e.g. with dijkstra and the generate source/sink tree)

Dynamics:

1. Link failure: Both nodes notice → updated LSPs (that doesn't include link anymore)
2. Node failure: All neighbors notice & remove link.

3. Adding node: others receive LSP from it and add it to their topology
4. Adding link: Nodes of link update their LSP
5. Problems: Seqno reaches max,or is corrupted + Node crashes and loses seqno + network partitions then heals + some weird corner cases
   (a) Solution: Include age on LSPs and forget old (non refreshed) information

4.6.7 DV vs LS

Goal	Distance-Vector	Link-State
Correctness	Distributed Bellman-Ford	Replicated Dijkstra
Efficient paths	Approx. with shortest paths	Approx. with shortest paths
Fair paths	Approx. with shortest paths	Approx. with shortest paths
Fast convergence	Slow – many exchanges	Fast – flood and compute
Scalability	Excellent: storage/compute	Moderate: storage/compute

4.7 Equal Cost Multi-Path Routing (ECMP Routing)

1. **Concept.** = Link-State routing but using Dijkstra's Algorithm without choosing one path if there are **multiple paths with the same cost** but **keeping all of them** as forwarding options. Thus instead of a source/sink tree we have a directed acyclic graph.
2. **Forwarding.**
   (a) Option 1: if there are multiple possible paths, randomly pick next hop. Thus good load balancing, but jittery and can cause packet reordering
   (b) Option 2: Try to send packets from one flow along the same path (i.e. map flow identifier to next hop). Thus less load balancing, but no jitter within flow. Identify flow by using 5-tuple (src-IP, dest-IP, src-Port, dest-Port, Protocol) or use IPv6 FlowID

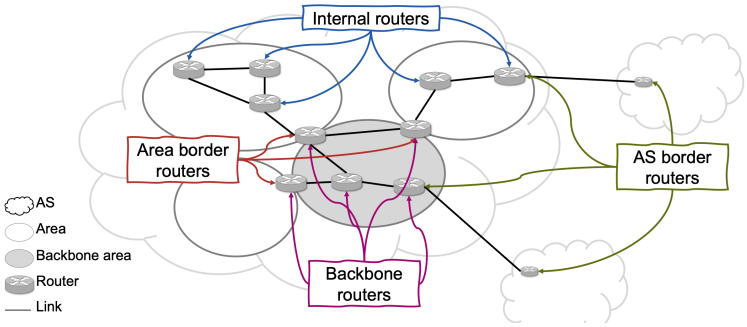
4.7.1 Intermediate System to Intermediate System (IS-IS) & Open Shortest Path First (OSPF)

1. **General information**: Used in large enterprise and isp networks (intra-network protocols (in autonomous system)), Link state protocol with added features (e.g. areas)
2. **Concept**
   (a) Split autonomous system into multiple smaller (autonomous) areas
   (b) Each area is identified by a 32-big number (looks like IP).
   (c) The routing happens for each area independently (i.e. each area uses only its own topology and runs Link State for the area) and the area's topology is not visible from its outside.
   (d) Each router keeps a link-state database for the areas they are connected to and communicates only summarized information of it to its outside
   (e) Areas are exclusively connected through a backbone area which is identified by number 0.0.0.0 an is contiguous.
3. **Routers** We differentiate between multiple types of routers:
   (a) Internal Routers: Belong to a single area (not in the backbone): Run LS with routers belonging to their area

(b) Backbone Routers: Routers with connection to backbone area.

(c) Area Border Routers: Belong to the backbone area and at least one other area: Run LS for all connected areas, then condense topological information and relay to backbone

(d) AS Boundary Routers: Routers connected to other autonomous systems, can be internal or backbone routers, they advertise external routing information throughout the entire autonomous system.



4. **OSPF Routing Algoritm**

(a) Routers keep link-state database storing the area's topology

(b) Routers send out link-state advertisements (LSAs) (authenticated with shared cryptographics keys): IP-Packets containing originating router ID and its interfaces with its respectives costs (?)

(c) Area border routers and AS border routers distribute additional LSAs containing information about their connected area or AS.

(d) Properties Supports ECMP Routing + there are extensions for traffic engineering (OSPF-TE) to exchange additional information beyond link costs and more elaborate traffic engineering

5. **IS-IS**: OSPF based on it; uses LS routing, Dijkstra and areas.

	OSPF	IS-IS
Purpose	Designed for IP traffic	Neutral towards layer-3 protocols
Encapsulation	Runs on top of IP	Runs directly over layer 2
Area boundaries	On **routers** Routers belong to different areas	On **links** Routers belong to a single area
Backbone area	Contiguous area 0.0.0.0 connects all other areas	No backbone area
IPv6 support	Added in OSPF v3	Supported implicitly

4.8 Constructing a Routing Table & Bloom Filters

4.8.1 Constructing a Routing Table

Optional for exam; TODO

4.8.2 Bloom Filters

1. **Problem**: Check if element is a duplicate, but cannot store all previous elements
2. **General Concept**:

   (a) Set up bit-vector V of m bits initialized to 0. Use k hash functions H_i that output numbers $\{0, \ldots, m-1\}$.
   (b) When inserting element e, calculate their k hash values $H_i(e)$ and set $V(H_i(e)) = 1$.
   (c) When testing if element e' has been seed: Compute k hash function and check if all values are set to 1. If yes: seen before
3. **Choosing the right values**: m: number of bits, k: number of hash-functions, n number of elements inserted
   (a) $\mathbb{P}[False - Positive] = (1 - [1 - \frac{1}{m}]^{kn})^k \approx (1 - e^{-\frac{kn}{m}})^k$
   (b) k minimizes $\mathbb{P}$ for $k = \frac{m}{n} \ln(2)$
   (c) Given optimal k, m is optimal if $m = -\frac{n \ln p}{(\ln 2)^2}$
4. **Properties**: $O(1)$ membership testing, constant number of bits per element, $O(n)$ memory overhead compared to no filter, normally no False Negatives, some False Positives
5. **Problem**: Bloom filter fills up → at some point packets that haven't been seen will be very likely to be classified as "seen" (False Positives). Solution: Reset bloom filter periodically. Therefore: Some duplicates not seen (introduces False Negative) and some falsely detected
6. **Duplicate Detection in Practice**: Use two bloom filters, reset them periodically with a phase shift. Add only if in certain time range and to the bloom filter according to the period they are in.

4.9 BGP

4.9.1 What inter-domain routing is driven by

1. What inter-domain routing needs
   (a) Networks (ASes) don't want other to see internal topologies
   (b) Networks needs to control where to send and receive traffic
   (c) Scalability of routing protocol (millions of routers)
2. Problems of Link-State and Distance Vector Routing
   (a) Link-State: Floods topology information, each node to compute the entire path, minimizes some notion of total distance
   (b) Distance-Vector: minimizes some common distance, converges slowly

4.9.2 Customer-Provider vs Peer-to-Peer

1. Customer-Provider: Customers pay providers to get Internet connectivity
   (a) The amount paid is based on peak usage, usually according to the 95th percentile rule
2. Peer-to-Peer: Peers don't pay each other for connectivity, they do it out of common interest
3. Rules for Forwarding:
   (a) Providers forward traffic for their customers
   (b) Peers do not traffic transit between each other
   (c) Customers do not traffic transit between their providers

4.9.3 Path Selection and Path Advertisement

1. **Path Advertisement**: Which path to advertise? (Where can the traffic come from?)

		send to		
		customer	peer	provider
from	customer	✓	✓	✓
	peer	✓	-	-
	provider	✓	-	-

2. Which path to use? (Where do we send traffic to?)

4.9.4 BGP Protocol

Two types:

1. eBGP: external BGP sessions connect border routers in different ASes; used to learn routes to external destinations
2. iBGP: Internal BGP sessions connect the routers in the same AS; used to spread externally-learned routes internally
3. After internal dissemination of routes: Announce them externally to other ASes.

BGP message types:

1. open: establish TCP-based BGP sessions
2. notification: report unusual conditions
3. update: inform neighbor of a new/a change of/the removal of the best route
4. keepalive: inform neighbor that the connection is alive

BGP update:

1. IP-prefix
2. Attributes: Describe route properties that are used in route selection/export decisions: are local (only for iBGP) or global (iBGP + eBGP)
   (a) NEXT-HOP: AS-entry/exit router (the first router to choose after leaving own AS or the last router to visit after entering AS)
   (b) AS-PATH: Path of ASes a packet traverses (to avoid loops, control outbound and inbound traffic)
   (c) LOCAL-PREF: local attribute, determines how preferred a route is within the AS (to get out of it) (outbound traffic control)
   (d) MED: global attribute, Multi-Exit Discriminator, lower MED value indicates closeness and is preferred over a higher value (if traffic comes from one AS):(inbound traffic control)
   (e) LOCAL-PREF vs MED: Suppose there are two entry routers of an AS, where one has a lower MED than the other ($MED - R1 < MED - R2$). If the LOCAL-PREF of the sending AS is the same for $R1$ and $R2$, $R1$ is chosen. However if $LOCAL-PREF-R1 < LOCAL - PREF - R2$ the AS chooses $R2$!

BGP prefers routes that (high to low relevance):

1. with higher LOCAL-PREF
2. with shorter AS-PATH length
3. with lower MED
4. with lower IGP metric to the next-hop (=hot potato routing = get the packet out of the AS as quickly as possible; leads to asymmetric trafficking)
5. with smaller egress IP address (tie-break)

How to set LOCAL-PREF? Prefer

1. customers over (get money)
2. peers over
3. providers (have to pay money)

4.9.5 Problems

1. Reachability: policy routing does not guarantee reachability even if the graph is connected
2. Security: Security considerations are simply absent from BGP specifications (ASes can advertise any prefixes, ASes can arbitrarily modify route content, ASes can forward traffic along different paths)
3. Convergence: With arbitrary policies, BGP may fail to converge; but: If all AS policies follow the cust/peer/provider rules, BGP is guaranteed to converge (Oscillations require "preferences cycles" which make no economic sense)
4. Performance: BGP path selection is mostly based on economics, not based on accurate performance criteria
5. Anomalies: BGP configuration is hard to get right (BGP is often manually configured, BGP abstraction is fundamentally flawed: disjoint, router-based configuration to effect AS-wide policy)
6. Relevance: The world of BGP policies is rapidly changing (ISPs are now eyeballs talking to content networks, Transit becomes less important and less profitable, No systematic practices)

5 Link Layer

5.1 What does the Link Layer provide?

1. Framing: Delimiting start/end of frames
2. Error detection/correction: Handling errors
3. Retransmissions: Handling loss & errors
4. Multiple Access: Classic Ethernet, 802.11
5. Switching: Modern Ethernet

5.2 Retransmissions with ARQ (Automatic Repeat reQuest)

ARQ often used when errors are common or must be corrected: WiFi, TCP.

1. Receiver automatically acknowledges correct frames with ack
2. Sender automatically resends after timeout, until ack is received
3. Stop-and-Wait protocol implements ARQ

5.3 Multiplexing and Multiple Access

Multiplexing is the network word for the sharing of a resource (e.g. sharing a link between multiple users)

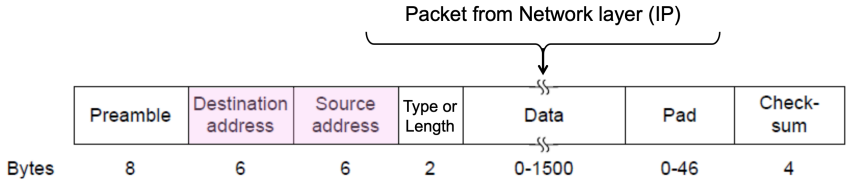
1. Time-division multiplexing: Users take turns on a fixed schedule
2. Frequency-division multiplexing: Put different users on different frequency bands
3. Comparison: In TDM a user sends at a high rate a fraction of the time; in FDM, a user sends at a low rate all the time
4. Use-cases: continuous traffic, fixed number of users (TV and radio (FDM), 2G cellular)

How to multiplex network traffic?

1. Problem: Network traffic is bursty (Load varies greatly overtime, variable number of active users) → inefficient to always allocate users when they are ON (use much data at the same time and then none)
2. Using Multiple Access increase efficiency: Randomized MA: Nodes randomize their resource access attempts and Contention-free MA: Nodes order their resource access attempts

5.4 Randomized Multiple Access: Classic Ethernet

1. Question: When should nodes send their data to use available bandwidth well?
2. Idea (ALOHA): Node send traffic when it has some. If there was a collision (no ack received): wait a random time and resend.
   (a) Simple, decentralized protocol that works well under low load!
   (b) But: Not efficient under high load: at most 18%, improvement by splitting time into slots: 36%.
3. Idea (Classic Ethernet): Improve ALOHA, but listen to traffic before sending (Doh!): Only wires (= Carrier Sense Multiple Access, CSMA)
   (a) **Problem**: Still possible to listen and hear nothing when another node is sending because of delay → collisions still possible
   (b) **Collision Detection**: Can reduce the cost of collisions by detecting them and aborting (Jam) the rest of the frame time (= CSMA with Collision Detection)
   (c) **CSMA/CD Complications**: To inform nodes of collision: Collision can be detected in time window of $2D$ seconds → make frame size big enough so that it lasts for $2D$ seconds (so that no other frame can be sent before the collision is detected), in Ethernet 64 bytes
   (d) **CSMA Persistence**: What to do if other node sends? Wait and send if there's space? Problem if multiple nodes want to send: they collide. Thus for N queued senders, each sends with probability $1/N$
   (e) **Binary Exponential Backoff**: Estimating the probability: 1st collision: 0..1 frame times, 2nd collision: 0..3 frame times, 3rd collision: 0..7 frame times: BEB BEB doubles interval for each successive collision
   (f) **Classic Ethernet: IEEE 802.3**: Uses multiple access with 1-persistent CSMA/CD with BEB: 85% efficiency.
4. Ethernet Frame Format: Address to identify receiver/sender, CRC-32 for error detection; no ACK or retransmission, start with physical layer preamble:



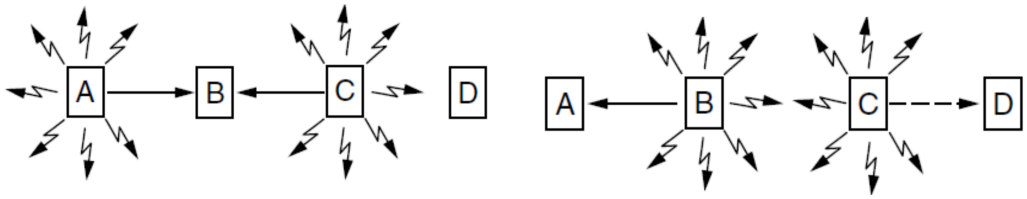
5.5 Wireless Multiple Access

5.5.1 What not CSMA/CD for Wireless?

1. Nodes may have different areas of coverage → Carrier Sense not possible (not all nodes receive the signal)
2. Nodes can't hear while sending → no Collision Detection possible

5.5.2 Hidden vs Exposed Terminals

1. Hidden Terminals (left): Can't hear each other (to coordinate); collide at B → want to send so that there are no collisions.
2. Exposed Terminals (right): Can hear each other; don't collide at receivers A and D → want to send all the time to increase performance



5.5.3 Multiple Access with Collision Avoidance (MACA)

1. Protocol:
   (a) A sender node transmits a RTS (Request-To-Send, with frame length)
   (b) The receiver replies with a CTS (Clear-To-Send, with frame length)
   (c) Sender transmits the frame while nodes hearing the CTS stay silent
   (d) Thus collisions on the RTS/CTS are still possible, but less likely
2. Hidden Terminals (A sends to B with Hidden Terminal C)
   (a) A sends RTS to B
   (b) B sends CTS to A (but C hears it because it is in range!)
   (c) A sends frame while C defers
3. Exposed Terminals (B sends to A; C sends to D)
   (a) B and C send RTS to A and D
   (b) A and D send CTS to Band C
   (c) B and C send frames

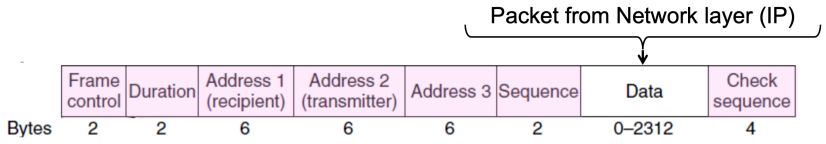
5.6 CSMA/CA for Multiple Access: 802.11

5.6.1 Technology (Physical Layer) of 802.11

1. Idea. Clients get connectivity from AP (Access Point)
2. Uses 20/40 MHz channels on ISM bands (a portion of the radio spectrum); 802.11b/g/n on 2.4GHz; 802.11a/n on 5GHz
3. OFDM (Orthogonal Frequency-Division Multiplexing) modulation (except 802.11b):
   (a) Different amplitudes/phases for varying SNRs (Signal-to-Noise Ratios) (?)
   (b) Rates from 6 to 54 Mbps plus error correction
   (c) 802.11n uses multiple antennas

5.6.2 Link Layer of 802.11: CSMA/CA

1. 802.11 Frame Format: Three addresses (source, receive, address of access point), CRC-32 for error detection, Frames are ACKed and retransmitted with ARQ, encryption, power save



2. Idea. Use CSMA/CA and optional RTS/CTS
   (a) Collision Avoidance (CA): Sender avoids collisions by inserting small random gaps (i.e. random backoff)

5.7 Contention Free Multiple Access

5.7.1 Why not Randomized Multiple Access?

1. CSMA is good under low load: Grants immediate access + Little overhead(few collisions)
2. But not so good under high load: High overhead(expect collisions) + Access time varies (lucky/unlucky)
3. Thus we use Turn-Taking Multiple Access Protocols: define an order in which nodes get a chance to send/pass

5.7.2 Token Rings

1. Idea. Arrange nodes in a ring; token rotates "permission to send" to each node in turn (= sprechstein)
2. Advantages. Fixed overhead with no collisions (= more efficient under load) + Regular chance to send with no unlucky nodes (Predictable service, easily extended to guaranteed quality of service)
3. Disadvantage: Complexity: What happens if token gets lost? Who manages the token (Crash?) + overhead at low load
4. In practice: random multiple access usually good enough.

5.8 LAN-Switches

5.8.1 Concept

1. Hosts are wired to Ethernet switches with twisted pair: Switch serves to connect the hosts
2. Inside a hub (physical-layer box): All ports are wired together; more convenient and reliable than a single shared wire
3. Inside a switch (link-layer box): Uses frame addresses to connect input port to the right output port; multiple frames may be switched in parallel
   (a) Port may be used for both input and output (full-duplex)
   (b) No multiple access protocol (can just send into a switch)
   (c) Need buffers for multiple inputs to send to one output
   (d) Sustained overload will fill buffer and lead to frame loss
4. Advantages of Switches. Switches and hubs have replaced the shared cable of classic Ethernet (Convenient & more reliable: one broken cable doesn't break the connection) + Switches offer scalable performance

5.8.2 Switch Forwarding: Single Switch

1. Problem. Switch needs to find the right output port for the destination address in the Ethernet frame without looking at IP.
2. Backward Learning. Switch forwards frames with a port/address table as follows:
   (a) To fill the table, it looks at the source address of input frames (e.g. if a switch receives a src address at a certain port, it knows that it can reach the address through that port when sending later)
   (b) To forward, it sends to the port, or else broadcasts to all ports
3. If there are no loops this approach works with multiple switches ass well.

5.8.3 Switch Forwarding: Multiples Switches with Spanning Trees

1. Problem: If loop in switch structure, frames could be forwarded in circles...
2. Idea: Switches collectively find a spanning tree for the topology
   (a) A subset of links that is a tree (no loops) and reaches all switches
   (b) Switches forward as normal but only on spanning tree
   (c) Broadcast will go up to the root of the tree and than down the branches
3. Algoritm:
   (a) Each switch initially believes it is the root of the tree
   (b) Each switch sends periodic updates to neighbours with: its address + address of the root + and distance to root
   (c) When neighbour receives update: if update has smaller distance to root, update own values. If distance the same: update on smaller root-address

5.9 Framing

5.9.1 Motivation (why length doesn't work)

1. Idea. Assume a stream of bytes. Each frame begins with a length field that indicates how many bytes the frame includes. Thus we would know that after length bytes a new frame begins.
2. Problem. If the length field of one frame is corrupted **all** consequent frames will not be correctly identified as we assume that the frame after the corrupted one stars earlier/later and read another wrong length field.

5.9.2 Byte stuffing

1. Idea.
   (a) Have a special flag byte value that means start/end of frame: FLAG
   (b) However the value of FLAG could appear in the data section of the frame. Thus we replace FLAG with special code ESC.
   (c) However ESC could appear naturally in the data as well.
2. Solution:
   (a) After performing the first two steps (adding FLAG and replacing FLAG with ESC in data)
   (b) Replace each occurrence of FLAG with ESC, FLAG
   (c) Replace each occurrence of ESC with ESC, ESC
   (d) Now any remaining FLAG is the start/end of a frame

5.9.3 Bit stuffing

1. Call a flag six consecutive 1s
2. On transmit, after five 1s in the data, insert a 0
3. On receive, a 0 after five 1s is deleted

5.9.4 PPP over SONET

1. PPP: Point-to-Point Protocol: Used to frame IP packets sent over SONET optical links
2. PPP uses byte stuffing with FLAG 0x7E and ESC 0x7D.
3. To stuff (unstuff) a byte, add (remove) ESC (0x7D), and XOR byte with 0x20
4. Removes FLAG from the contents of the frame

5.10 Error Detection and Correction

5.10.1 Error Codes & Hamming Distance

1. Idea. Suppose there are D data bits. The sender computes $R = f(D)$ check bits and appends them to the data before sending. This combination is called codeword. The receiver receives D, R and computes $f(D)$ as well. If $f(D) \neq R$ we know there is an error.
2. Hamming Distance. Minimum number of bit flips needed to change between any pair of legal codewords.
3. Error detection with Hamming Distance. For a code of Hamming distance $d + 1$, up to d errors will always be detected
4. Error correction with Hamming Distance. For a code of Hamming distance $2d + 1$, up to d errors can always be corrected by mapping to the closest codeword

5.10.2 Error Detection with Checksums

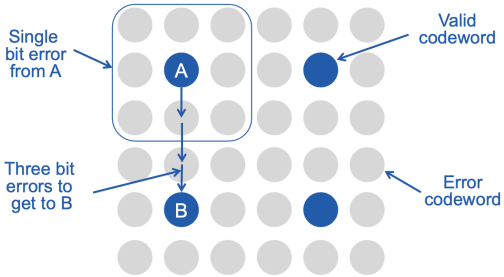
1. Parity Bit. Take D data bits, append 1 check bit that is the sum of the D bits. But if two bitflips sum stays the same → not really good at detecting errors
2. Checksums. Sum up data in N-bit words
   (a) By RFC 791: "The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words"
   (b) Can check checksum by doing by adding the checksum to the sum of 16-bit words and check if 0 (bc. checksum is 1's complement)

5.10.3 Error Detection with Cyclic Redundancy Check (CRC)

1. Send Procedure
   (a) Extend the n data bits with k zeros
   (b) Divide by the generator value C
   (c) Keep remainder, ignore quotient
   (d) Add remainder to the k zeros
2. Receive Procedure: Divide bits by C and check for zero remainder
3. Properties of CRC-32
   (a) Hamming-Distnace 4: detects up to triple bit errors
   (b) Also detects odd number of errors
   (c) And bursts of up to k bits in error
   (d) Not vulnerable to systematic errors (i.e., moving data around) like checksums

5.10.4 Error Correction

1. Problem. Correction would be easier if we had reliable check bit, however data could be correct and check bits corrupted. Can correct codewords with d error if hamming distance $\geq 2d+1$ by mapping to one unique nearest codewords: Need at least hamming distance of 3.



2. Using Hamming-Code for error correction: Method for constructing a code with a distance of 3
   (a) Use $n = 2^k - k - 1$
   (b) Put i-th check bit in position $p_i = 2^{i-1}$, fill gaps with data.
   (c) Check bit in position p_i is parity of the values that have the i-th bit set. (e.g. third check bit in position $p_3 = 4$ is parity of all values that have the third bit set, xx...x1xx). Begin calculating parity from smallest position.
   (d) When receiving: recalculate parity of each check bit position and add check bit to it. Then arrange them in a bitstring. (e.g. p'_3, p'_2, p'_1). The bitstring is called syndrome, if it's 0, there is no error, otherwise flip bit at position of the syndrome's value (e.g. for syndrome 110 flip bit 6) to obtain the actual value.

5.10.5 Error Detection vs Correction

1. Error correction: Needed when errors are expected: Small number of errors are correctable; Or when there's no time for retransmission
2. Error detection: More efficient when errors are not expected; And when errors are large when they do occur.

6 Physical Layer

6.1 Link Properties

1. Rate (or Bandwidth) R in bits/second
2. Delay (or Latency) L in seconds: delay to send a message over a link
   (a) Transmission Delay: Time to put M bit signal on the wire: $M/Rseconds$
   (b) Propagation Delay D: time for bits to propagate across the wire $D = length/speedofsignal \approx length/(c2/3)$
3. Bandwidth-Delay-Product BDP (or BD): Amount of Data in flight at once: $R \cdot D$

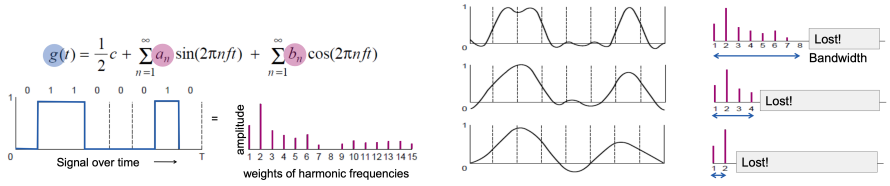
6.2 Types of Media

1. **Wires – Twisted Pair**
   (a) The twists of two cables can reduce the radiated signal or effects of external interference
   (b) Used in LANs and telephone lines
2. **Wires – Coaxial Cable:**
   (a) Copper Core, insulating material, braided outer conductor, protective plastic covering
   (b) Improved shielding improves performance
3. **Fiber:**
   (a) Long, thin, pure strands of glass (core: glass, cladding: glass, jacket: plastic)
   (b) High bandwidth over long distance; light trapped in optical fiber and is totally reflected
   (c) two variants: multi-mode (cheaper, short distances) and single mode (100km)
4. **Wireless:**
   (a) Sender radiates signal over a region in many directions, there can be multiple receivers
   (b) Signals of the same frequency interfere at receiver, thus coordination is necessary
   (c) Frequencies: Microwave (e.g. 3G); ISM: Industry Science Medicine frequencies (e.g. WI-FI)

6.3 Sending a signal

6.3.1 Frequency Representation

A signal in time domain is represented by amplitudes in frequency domain:



Lower bandwidth (i.e. less available frequencies) means that 0s and 1s cannot be cleanly separated anymore (=degraded signal)

6.3.2 Signals over different Media

1. **Signals over Wire**
   (a) The signal is delayed (propagates at $2/3c$)
   (b) The signal is attenuated (loses strength)
   (c) Frequencies above a cutoff are highly attenuated (there is a range of frequencies on which signals are sent)
   (d) Signal has noise ($\rightarrow$ causes errors)
2. **Signals over Fiber**
   (a) Light propagates with very low loss in three very wide frequency bands
   (b) Use a carrier to send information
3. **Signals over Wireless**
   (a) Travel at speed of light, spread out and attenuate faster than $1/dist^2$
   (b) Multiple signals on the same frequency interfere at a receiver
   (c) Spatial reuse (of same freq.): Simultaneously using the same frequency channel in different spatial locations without causing significant interference.
   (d) Wireless multipath: Signals bounce off objects and take multiple paths $\rightarrow$ signal can block/strengthen itself

6.4 Modulation

= How to represents bits in signals

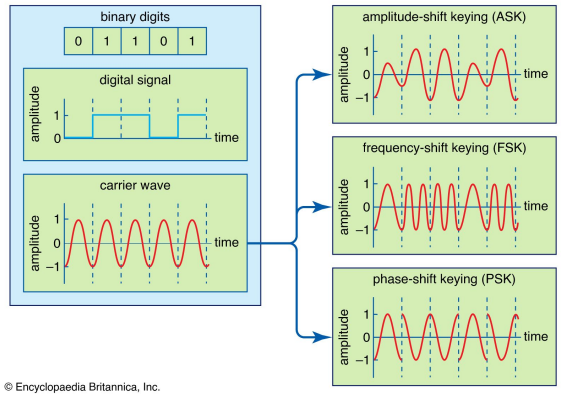
6.4.1 Baseband Modulation

= Signal is sent directly

1. Non-Return to Zero (NRZ): High voltage (+V) represents a 1, and low voltage (-V) represents a 0
2. Clock-Recovery: The receiver needs frequent transmission of 1s to decode the signal (can't know how many 0s there are if there are to many consecutively, or the other way around)
   (a) 4B/5B: Make maps every possible pair of 4 bits to pairs of 5 bits that don't contain at most 3 consecutive 0s.
   (b) + Invert signal on many consecutive 1s
   (c) Other Methods: Manchester coding and scrambling

6.4.2 Passband Modulation

Idea. Have Carrier-Wave at given frequency: modulate the carrier by changing amplitute (higher when 1, lower when 0), frequency (one frequency for 1, another for 0) and phase (change phase of signal)



6.5 Limits of Propagation

1. **Channel Properties:**
   (a) Bandwidth B: Limits the rate of transitions
   (b) Signal Strength S, Noise Strength N: Limit how many signals we can distinguish
2. **Nyquist Limit**: For V signal levels the maximum bit rate (rate at that bits are actually sent $\neq$ bandwidth) is
$$R = 2B\log_2(V)\ bits/s$$
3. **Shannon Capacity:** Idea: The amount of levels we can distinguish between deepends on S/N = Signal-to-Noise ratio SNR, in deciBels: $SNR_{dB} = 10\log_{10}(S/R)$. Shannon proposes that the maximum information carrying rate C of the channel is
$$C = B\log_2(1 + S/N)\ bits/s$$
4. **Wired vs Wireless:**
   (a) Wire: Can build link to have fixed SNR and B. Therefore they can make fixed data rate.
   (b) Wireless: Can build to have fixed B, but SNR varies greatly: SNR determines data rate.

6.6 DSL

1. Reuses twisted pair telephone line to the home
2. Uses passband modulation (called OFDM §7.4.3)
3. Separate bands for upstream and downstream (larger)
4. Modulation varies both amplitude and phase (QAM)

7 Algorithms for Networks

7.1 Optimal Paths

7.1.1 Shortest Paths

1. Goal. Given Graph with weighted edges, minimize sum of weights of path between two vertices:
2. Algorithms:
   (a) BFS (unweighted graphs in $\mathcal{O}(|V| + |E|)$)

(b) Dijkstra (positively weighted graphs in $\mathcal{O}((|V| + |E|) \log(|V|))$)

(c) Bellman-Ford (weighted graphs (no negative cycles) in $\mathcal{O}(|V| \cdot |E|)$)

3. Weights in networking: Latency, Bandwidth, Loss rate

4. Problems:

   (a) Non-additive weights: We assume the weight of a path is the sum of the weights of the edges, but Bandwidth and Loss rate are not (bandwidth is min of all edges and loss rate is multiplicative). Algorithms can be modified

   (b) Multiple criteria: Shortest-widest path (first optimize for bandwidth and then choose smallest path) Widest-shortest path (optimize for length, choose largest bandwidth) = lexicographic order

   (c) Shortest-widest path does not fulfill isotonicity (Given two paths P1 and P2 from A to B and a path P3 from B to C. If P1 is better than P2 then P1 + P3 is better than (or equal to) P2 + P3): Need to keep track of multiple paths, not possible to use standard algorithms.

7.1.2 Optimal Traffic

1. Maximize what we are able to send at once and make system reliable (cannot be easily cut off) → min-cut/max-flow

2. Algorithms

   (a) Ford–Fulkerson: augmenting paths in the residual network (doesn't necessarily terminate)

   (b) Edmonds–Karp: Optimized search (shortest) for augmenting paths (terminates), $\mathcal{O}(|E|^2 \cdot |V|)$ or $\mathcal{O}(|V|^2 \cdot |E|)$

3. Problems

   (a) Multiple Sources/Desinations: Can be modeled with additional source/destination that connects to multiple nodes.

   (b) Multiple-commodity-flow (see later)

7.1.3 Finding simultaneous circuits

= maximum-cardinality matching → use maxflow to solve.

7.2 Linear Programming

7.2.1 General Form

1. Goal. Given a vector $c = (c_1, \ldots, c_n)^T$, a $m \times n$ matrix A and a vector $b = (b_1, \ldots, b_m)$:

   (a) Maximize $\Theta = c^T x = \sum_{i=1}^{n} c_i x_i$

   (b) under the constraint: $Ax \leq b$ ($\forall i \in \{1, \ldots, m\} : \sum_{j=1}^{n} a_{ij} x_j \leq b_i$)

2. Transforming LP to canconical form:

   (a) Minimization of $\Theta \rightarrow$ Maximization of $-\Theta$

   (b) Greater than: $Ax \geq b \rightarrow -Ax \leq -b$

   (c) Equality: $Ax = b \rightarrow Ax \leq b$ and $-Ax \leq -b$

   (d) Variables with undefined sign: Introduce variables x_i^+ and x_i^- and replace each occurrence of x_i by $x_i^+ - x_i^-$

3. Algorithms. Simplex (worst-case exponential runtime, but usually fast), Karmarkar's algorithm in $\mathcal{O}(n^{3.5})$. In general: rule of thumb $\mathcal{O}(n^3)$.

7.2.2 Max-Flow with LP

1. Objective: maximize flow f
2. Variables:
   (a) Flow from source to destination $S \to T$: f
   (b) Flow per edge $u \to v$: f_{uv}
   (c) Net outflow of v: $F(v) = \sum_{w \in E^+(v)} f_{vw} - \sum_{w \in E^-(v)} f_{wv}$
3. Constraints: Can be transformed to matrix
   (a) Edge capacities $\forall u \to v \in E : 0 \leq f_{uv} \leq c_{uv}$
   (b) Flow from S: $f = F(S)$
   (c) Flow into T: $f = -F(T)$
   (d) Flow conservation: $\forall v \in V \backslash \{S, T\} : F(v) = 0$

7.2.3 Multi-Commodity Flow with LP

1. Problem situation:
   (a) Given a graph $G(V, E)$ with capacities $c(u, v)$ for each edge and k commodities (S_i, T_i, d_i) consisting of source S_i, sink T_i and demand d_i.
   (b) The usual max-flow constraints hold for each commodity
2. Possible objectives:
   (a) Maximize sum of flows of all commodities (i.e. maximize $\sum_\alpha f_\alpha$)
   (b) Maximize minimum of all commodities
   (c) Define demand for each commodity and maximize average "fulfillment ratio"
3. Variables for max sum:
   (a) Flow for commodity α, $S_\alpha \to T_\alpha$: f_α
   (b) For for commodity α on edge $u \to v$: $f_{\alpha,uv}$
   (c) For commodity α, net outflow of v: $F_\alpha(v) = \sum_{w \in E^+(v)} f_{\alpha,vw} - \sum_{w \in E^-(v)} f_{\alpha,wv}$
4. Constraints for max sum:
   (a) Edge capacities: $\forall u \to v \in E : 0 \leq f_{\alpha,uv} \wedge \sum_\alpha f_{\alpha,uv} \leq c_{uv}$
   (b) Flow from S_α: $f_\alpha = F_\alpha(S_\alpha)$
   (c) Flow into T_α: $f_\alpha = -F_\alpha(T_\alpha)$
   (d) Flow conservation: $\forall v \in V \backslash \{S_\alpha, T_\alpha\} : F_\alpha(v) = 0$

7.2.4 Shortest Paths with LP - Variant 1

1. Problem situation: Graph $G(V, E)$ with weights w_{uv} and $S, T \in V$
2. Objective: Minimize length of path $S \to T$: $\sum_{u \to v \in E} x_{uv} w_{uv}$
3. Variables:
   (a) x_{uv}: Is edge $u \to v$ on shortest path?
   (b) $X(v) = \sum_{w \in E^+(v)} x_{vw} - \sum_{w \in E^-(v)} x_{wv}$
4. Constraints:
   (a) $S \to T$ is connected: $\forall v \in V \backslash \{S, T\} : X(v) = 0$

(b) Path starts at S: $X(S) = 1$

(c) $x \in \{0, 1\}$: Still works for $x \in [0, 1]$!

5. Understanding the Problem: Minimizing the sum means to set all $x_{uv} = 0$ if $u \to v$ is not on the shortest path. However not all x_{uv} can be 0 as $X(S) = 1$. Suppose we have found the shortest path and the corresponding x_{uv}. If we took an $x_{uv} = 0$ (i.e. $u \to v$ is not on the shortest path) and increased its value the total sum would increase.

7.2.5 Shortest Paths with LP - Variant 2

1. Objective: Maximize d_T
2. Variables: Distance from S: d_u ($d_S = 0$)
3. Constraint: $\forall u \to v \in E : d_v \leq d_u + w_{uv}$

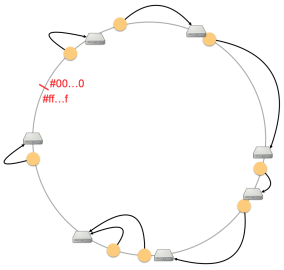
7.2.6 Integer Linear Programming

1. Definition. Linear Programming with variables restricted to integers.
2. Complexity. General problems NP hard (however some can still be solved efficiently)

7.3 Probabilistic Techniques

7.3.1 Load Balancing

1. Problem description. Application deployed on multiple servers. Request to application can be balanced among them. How to distribute multiple request to keep response time low and have uniform load at servers?
2. **Choose servers uniformly random.** Numbers of imbalances = m balls at n bins problem:
   (a) **Probability two requests at same server**: $1/n$
   (b) **Expected number of same servers after m requests**: $\frac{1}{n}\binom{m}{2} = \sum_{\{i,j\}:i,j \in Req} P[X_{ij} = 1]$
   (c) **#Requests after expectation** > 1? $1 = \frac{m(m-1)}{2n} \implies m \approx \sqrt{2n}$
   (d) **Expected maximum load per server**? For n servers: $\mathcal{O}(\frac{\log n}{\log \log n})$ maximum load per bin (i.e. if $n = 2^{2^k}$ then maximum load in $\frac{2^k}{k}$ = BAD)
   (e) **Decreasing maximum load:** Instead of picking one basket at once, pick m. Then expected maximum load in $\mathcal{O}(\frac{\log \log n}{\log m})$ (i.e. if $n = 2^{2^k}$ then maximum load in $\frac{k}{\log m}$)
3. **Randomize over Sessions**
   (a) Idea. Application might be interactive: Requests from one session should not be sent to different servers $\to$ Randomly choose server only at beginning of session.
   (b) Hash functions. We say $server\_id = (H(user\_id) \mod n) + 1$ for some hash-function H. The hash function is assumed to be uniform, therefore a server is chosen uniformly random per session. Usually $user\_id$ is tuple (src-IP, dest-IP, src-port, dest-port, protocol): ECMP.
   (c) Server failure.
      i. Problem: If one server failes most $user_id$s would be assigned to new server (when applying $server\_id = (H(user\_id) \mod (n-1)) + 1$, then for every n sessions $n - 1$ are reassigned)
      ii. Solution: "consistent hashing". Instead of transforming the $user\_id$ hash to a $server\_id$ (by using mod), hash $server\_id$ as well. Then assign $user\_id$ to the $server\_id$ whose hash value is the next following the hash of $user\_id$.



iii. Multiple Server Hashes. Give servers multiple ids to balance load after another server crashes.

7.3.2 Membership Testing: Bloom Filters (again)

1. **Problem**: Check if element is a duplicate, but cannot store all previous elements
2. **General Concept**:
   (a) Set up bit-vector V of m bits initialized to 0. Use k hash functions H_i that output numbers $\{0, \ldots, m-1\}$.
   (b) When inserting element e, calculate their k hash values $H_i(e)$ and set $V(H_i(e)) = 1$.
   (c) When testing if element e' has been seed: Compute k hash function and check if all values are set to 1. If yes: seen before
3. **Choosing the right values**: m: number of bits, k: number of hash-functions, n number of elements inserted
   (a) $\mathbb{P}[\text{False-Positive}] = (1 - [1 - \frac{1}{m}]^{kn})^k \approx (1 - e^{-\frac{kn}{m}})^k$
   (b) k minimizes $\mathbb{P}$ for $k = \frac{m}{n} \ln(2) \approx 0.7\frac{m}{n}$, then $\mathbb{P}[\text{False-Positive}] \approx 2^{-\frac{m}{n} \ln 2}$
   (c) Given optimal k, m is optimal if $m = -\frac{n \ln p}{(\ln 2)^2}$
4. **Properties**: $O(1)$ membership testing, constant number of bits per element, $O(n)$ memory overhead compared to no filter, normally no False Negatives, some False Positives
5. **Problem**: Bloom filter fills up $\rightarrow$ at some point packets that haven't been seen will be very likely to be classified as "seen" (False Positives). Solution: Reset bloom filter periodically. Therefore: Some duplicates not seen (introduces False Negative) and some falsely detected
6. **Duplicate Detection in Practice**: Use two bloom filters, reset them periodically with a phase shift. Add only if in certain time range and to the bloom filter according to the period they are in.

Cache Filtering: Log requested data in bloom filter and cache data if there is a hit in bloom filter (i.e. cache on second request)

7.3.3 Traffic Monitoring

1. Problem. Want to analyze flows (i.e. multiple packets with same flow identifier/user_id) or based on partrs header fields of packets (Source, Dest, etc). But: A lot of traffic at once, big overhead to monitor data.
2. Idea. Use probabilistic modeling to reduce overhead (i.e. estimate the data based on samples). Challenges:
   (a) Output an "accurate" estimate with high probability
   (b) Measure with limited memory/processing
3. Choosing item uniformly random from group size n: Choose $v \in [0, 1)$ randomly. Select i if $i - 1 \leq n \cdot v < i$

4. Choosing item uniformly random from group of unknown size (Can see every item once, can store one of them): Replace item in store with i-th one with probability $1/i \rightarrow$ all item have the same probability of being in the store at the end.

General-purpose measurements: Sampled NetFlow

1. Sample every k-th packet
2. For each flow: keep entry $\{C_{pkt}, C_{byte}\}$ and update it if the sampled packet is of the respective flow (i.e. increase C_{pkt} and C_{byte})
3. Estimate actual number by multiplying the C_{pkt} and C_{byte} by k.
4. (Dis)Advantages:
   (a) (+) easy to implement, reduced processing time (compared to looking at all packets)
   (b) (−) Memory overhead (need to store tuple for every flow), inaccurate estimate for short lived flows (best for long lived flows, sending many small packets)

Large-Flow detection: General Problem

1. Problem: Detect large flows (Flows that consume more than a given threshold of link capacity, e.g. 1%)
2. Efficiency: Possible to efficiently identify large flows because #of large flows << #of flows in total during a given measurement interval (we don't have to store per flow state)
3. Accuracy Metrics:
   (a) False Positive: wrongly include a small flow in the result
   (b) False Negative: wrongly exclude a large flow in the result (failed to identify)
   (c) Measurement error: error in traffic volume of the identified large flows
4. Methods: Sampling-Based (NetFlow, Sample-And-Hold), Sketch-Based (Multistage Filters), Eviction-Based (EARDet)

Large-Flow detection: Sample-And-Hold

1. Sample: byte-sampling technique (in contrast to packet sampling in NetFlow) taking packet size into account (samples each byte with probability p and thus packet of size s with probability $p_s = 1-(1-p)^s \approx p \cdot s$ for small p). Reduces memory overhead, because larger packets are more likely to be sampled and thus less packets in total need to be stored.
2. Hold: Update flow entry for all subsequent packets once it has been created (i.e. once a packet is in hold all packets of the same flow will (with probability 1) be counted): requires flow-table lookup
3. Algorithm:
   (a) For every packet: check if flow entry exists: If yes: do hold, if no: sample with probability p_s.
   (b) Update flow entries $\{C_{pkt}, C_{byte}\}$ if packet hold or sampling successful.
   (c) At the end: Flow entries in cache are the sampled large flows

NetFlow vs Sample-And-Hold

	Sample-and-Hold	NetFlow
Sampling technique	Byte sampling	Packet sampling
Estimate of number of packets/bytes in a flow	No over-counting	May over-count or underestimate
Error rate (Given memory overhead M)	$O(M^{-1})$	$O(M^{-1/2})$
Inspect headers of all packets?	Yes	No

Large-Flow detection: Multistage Filters

1. Singlestage Filter: $\approx$ Bloom Filter
   (a) Keep array of n counters
   (b) Router hashes the flow ID of the incoming packet with function H: Output in $\{0, \ldots, n\}$
   (c) Increase i-th counter of array if output of H is i.
   (d) A flow is considered a large flow if the counter value in the associated counter $\geq$ threshold.
2. Problem with Singlestage: A small flow is identified as a large flow if it's id has the same hash
3. Multistage Filter: Multiple singlestage filters with different hash functions.
   (a) A flow is considered large if the associated counter value in all stages are $\geq$ threshold
   (b) Properties: fixed memory resources, no FN few FP (decreases exponentially with number of stages)

Large-Flow detection: Frequent Item Finding (Majority Algorithm)

1. Overview: Linear time algorithm without FP or FN finding frequent items in $1/\theta$ space. First round: identify all frequent items, second round: eliminate all FPs.
2. Goal: Find all items that appear more than k times in a stream of m items with no false negatives and $n = m/k - 1$ counters.
3. Algorithm:
   (a) Initialize n empty counters with associated empty labels.
   (b) For each new_item:
   - If $\exists i$ with $counter_i > 0$ and $label_i == new\_item$: $counter_i{+}{+}$
   - Else if $\exists i$ with $counter_i == 0$ and $label_i == new\_item$: $counter_i = 1$
   - Else: $\forall i : counter_i - -$
4. Labels at the end are candidates for items that occur more than k times. Check by traversing the elements a second time and counting how often the candidates appear.

Large-Flow detection: Frequent Item Finding (EARDet)

1. Modification of MG: No second pass
2. No FP or FN
3. Deterministic performance independent of input traffic
4. Small storage but many counters
5. Disadvantage: per-packet counter update is more expensive

#Flows estimation: probabilistic counting

1. Ideas (trying to count how many different flows):
   (a) Record all distinct flow IDs. Problem: $\mathcal{O}(N)$ memory overhead for N distinct flows (often infeasible)
   (b) Or increase a counter for all if a incoming packet is new (determine if packet is new with bloom filter in $\mathcal{O}(1)$). Problem still $\mathcal{O}(N)$ memory overhead of bloom filter.
2. Probabilistic counting:
   (a) Hash FlowID. Value in $[0, 1)$.
   (b) Keep FlowID with the smallest hash value.
   (c) Then: The expectation of the smallest value for n flows is $v = 1/(n + 1)$. (If Hash uniformly distributed)
   (d) Therefore we can estimate n with the smallest observed v: $\tilde{n} = 1/v - 1$
3. Problems:
   (a) Minimum has large variance: estimate not robust
   (b) Possibility to influence the estimation from the outside by controlling only one input
4. Bar-Yossef et al.
   (a) Don't keep track of the one smallest, but of the k smallest hash values
   (b) Expectation of k-th smallest is $v_k = k/(n + 1)$
   (c) Then estimate with k-th smallest value: $\tilde{n} = k/v_k - 1$: Less variance.
   (d) Can additionally use sampling

8 Network Applications

8.1 CDNs (Content Delivery/Distribution Networks)

8.1.1 General Concept

1. Caching: After fetching a resource for a client, also store it in a cache to save time and decrease network and server load (like storing DNS records at DNS resolvers)
2. Replication:
   (a) Place content that will likely be requested close to clients
   (b) Can distribute load over multiple servers
   (c) Optimizes the latency of request.

8.1.2 Caching

1. Idea: Reactively cache content requested by clients
2. What data **cannot** be cached?
   (a) Dynamic data: some data is inherently dynamic
   (b) Scripts: results can be based on parameters
   (c) Cookies: content is specific to a user/session
   (d) TLS/QUIC: encrypted content can only be accessed by endpoints (not in the network)
   (e) Advertising: content provider wants to measure number of accesses
3. Validating cached objects.

(a) Server hints when an object expires as well as last modified date of it

(b) Client conditionally requests a resource using the "if-modified-since" header in HTTP request.

(c) Server compares modified-since time with own last modified. If the resource was modified (i.e. the last modification of the server was after the last modification of the client) the server returns "OK" with the new resource. Otherwise it returns "Not Modified" without the resource.

4. Caching locations.

   (a) Client. Browser cache

   (b) Close to client. Forward Proxy, CDNs

   (c) Close to server/destination. Reverse Proxy.

5. forward proxy: Multiple clients access forward proxy (which is near them) **before accessing the rest of the network.** A forward proxy could be the ISP or an enterprise and speeds up responses to clients/prevents network overload.

6. reverse proxy: Each server has a reverse proxy which is accessed **before accessing the main server structure**. Therefore it's from the content provider and prevents server overload from repetitive requests.

8.1.3 Global Replication

1. Idea. Bring content closer to clients, for:

   (a) Fault tolerance: Service remains available (at least partially) even if some datacenter or network fails

   (b) Load balancing: Distribute requests over multiple servers

   (c) Optimized Latency: Requests are directed to "nearby" server

   (d) Less Network inefficiency: No need to transmit data across the globe

2. How?

   (a) Spread the content servers globally

   (b) Network the sites and the origin

   (c) Direct clients to appropriate servers

Spread the content servers globally

1. Optimize network distance: place content at Internet exchange points (IXPs)

Connect the sites and the origin

1. Options:

   (a) Private Network: Full control over network paths & High predictability of available bandwidth; But: expensive and potentially low redundancy

   (b) Internet: Simple & cheap & available anywhere & can use multiple providers for redundancy; But: paths unpredictable and suboptimal & bandwidth not predictable

2. Overlay routing over Internet

   (a) Circuitous networks: For nodes A, B, C there may exists direct paths $A \to X$, $X \to B$, $A \to B$.

   (b) Overlay routing. The triangle inequality doesn't hold for these paths. To get from A to B it may be better to send $A \to X$, $X \to B$ than directly over $A \to B$ (i.e. we can send with lower latency and loss). This can be done by sending a packet to X which includes a packet to B = overlay routing.

(c) When to use overlay routing? Monitoring latency and loss for different routes and then choosing the best one.

Direct clients to appropriate servers

1. DNS-Based: Return different IP addresses based on receiver's geo location or server load. To prevent caching of the DNS record (bad if location changes or server is overloaded) use small TTL. Uses URL-rewriting
   + Very high control, Dynamic changes are possible
   − Complicated, Potential issues when clients do not use their local DNS resolver (the resolver determines the location)
2. BGP-anycast-based: Always use same IP address, but advertise them (with BGP) from different locations. For a request BGP then finds the closest location.
   + Optimization is done by BGP (simple to use)
   − Less precise control, Longer reconfiguration times (have to advertise IP)

8.2 Internet Video

8.2.1 Common Approach

1. Encode video in multiple bitrates
2. Replicate using a CDN
3. Video player picks bitrate adaptively (Estimate connection's available bandwidth & Pick a bitrate $\leq$ available bandwidth)

8.2.2 Encoding & Replication

1. Encode video in multiple resolutions; lower resolutions needs lees bitrates to be transported over the network.
2. Chunks. Part video in chunks. When a client streams a video it can request chunks of different resolutions. Chunks are communicated in manifest.
3. Every encoding is replicated on multiple servers with a CDN.

8.2.3 Adaption

1. Idea. Continuously monitor buffer and available network capacity. If capacity ¡ current bitrate: decrease quality and therefore bitrate of next chunk.
2. Problem: Network capacity very dynamic and difficult to estimate.
3. Buffer-based adaption:
   (a) If buffer nearly full: select **higher** bitrates (has more time to arrive before buffer is empty)
   (b) If buffer close to empty: select **lower** bitrates
   (c) At startup: Pick a rate based on immediate past throughput

9 Routing Security & SCION

9.1 Basic Terms

1. Secrecy: Keep data hidden from unintended receivers
2. Confidentiality: Keep someone else's data secret
3. Privacy: Keep data about a person secret

4. Anonymity: Keep identity of a protocol participant secret
5. Data Integrity: Ensure data is "correct" (i.e., correct syntax & unchanged): No unauthorized or improper changes
6. Entity Authentication: Verify the identity of another protocol participant
7. Data Authentication: Ensure that data originates from claimed sender

9.2 Encryption Primitives

9.2.1 Symmetric Encryption Primitives

1. E.g. Block cipher (pseudo-random permutation PRP), Stream cipher (pseudo-random generators PRG), Message authentication code (MAC)
2. Idea:
   (a) Encryption key k = Decryption key k
   (b) Encryption: E_k(plaintext) = ciphertext
   (c) Decryption D_k(ciphertext) = plaintext

9.2.2 Asymmetric Encryption Primitives

1. E.g. Diffie-Hellman key agreement, Public-key encryption, Digital signature
2. Idea:
   (a) Encryption key K is publicly known: public key
   (b) Decryption key K^{-1} is secret: private key
   (c) Encryption E_K(plaintext) = ciphertext
   (d) Decryption $E_{K^{-1}}$(ciphertext) = plaintext

9.2.3 Asymmetric Signature Primitives

1. E.g. Diffie-Hellman key agreement, Public-key encryption, Digital signature
2. Idea:
   (a) Encryption key K is publicly known: public key
   (b) Decryption key K^{-1} is secret: private key
   (c) Signature Verification S_K(msg, sig) = true/false
   (d) Signature Generation $E_{K^{-1}}$(msg) = sig

9.3 Attacks on intra-domain routing

1. Vulnerabilities
   (a) Compromise one router
   (b) compromise one routing adjacency/link (attacker acts as a Man-in-the-Middle)
   (c) Attacker obtains complete network view & ability to inject messages network wide because intra-domain routing with OSPF relies on flooding the network data.
2. Types of Attacks
   (a) Interception: eavesdrop on/drop/modify/inject/delay traffic by steering traffic along paths controlled by the attacker

(b) DoS: induce churn to overload the routers (by announcing/withdrawing at high frequency) & flood routers' link state database (by injecting prefixes) & induce congestion/delay (by steering traffic through bad routes) & prevent reachability (by steering traffic through blackholes, loops)

3. Intercepting traffic:

   (a) Attacker can inject fake nodes that have lower cost than the original path and correspond to an actual physical node. When nodes then perform routing they choose the fake node and forward packets along a new path.

   (b) Thus: It is always possible to find fake OSPF messages forcing the routers to compute any forwarding tree: Can program the way the network behaves from a single location

4. DoS: induce congestion and prevent reachability: By injecting fake nodes the attacker can steer traffic along a low throughput path or let packets be forwarded in loops.

5. Solution: Cryptography

   (a) Two problems: False advertisements can be made & legitimate advertisements can be altered

   (b) → Use cryptographic authentication: Send authenticated announcements (with header) and encrypt topology information.

9.4 Attacks on Inter-domain routing: Problems of BGP

Can influence security of Tor, Bitcoin, certificates, VPN, ...

9.4.1 BGP does not validate the origin of advertisements

1. IP Address Ownership

   (a) IP Address block assignment: Regional Internet Registries and ISP assign blocks of IP addresses

   (b) IP Prefixes: IP Prefixes are announced by the AS who owns it (or by its upstream providers)

   (c) Problem: Any other AS can originate the same prefix (=prefix hijacking). BGP doesn't verify the authorization! Moreover existing registers of prefix ownership are inaccurate.

2. Prefix Hijacking: Announce same IP Prefix: Some traffic is forwarded to injected IP Prefix. Possible to:

   (a) Discard traffic (blackhole)

   (b) Inspect traffic (snooping)

3. Sub-Prefix Hijacking: Announce more specific IP-Prefix: All traffic is forwarded there first!

4. Step-by-step: How to Hijack an IP-Prefix

   (a) Need AS with BGP sessions on router that are configured to announce the desired prefix

   (b) How to get the router? Network operator makes mistake or be network operator or break into router and reconfigure it.

   (c) How to make other ASes believe? Have them not apply protective filtering

5. Example: Mail spamming

   (a) Spammer: establishes TCP connection to mail server, sends spam mail and disconnects. Problem originating IP address is easy to trace

   (b) Solution: Hijack someone's address space or unused address block and use IP address to send spam :)

6. Debugging Prefix Hijacking. Problems:
   (a) Victim AS doesn't see the problem
   (b) Not necessarily loss of connectivity (snooping only causes degradation of performance)
   (c) If connectivity loss: only for small parts of the internet

   Only way is to analyze updates from many points or to use traceroute from many points

9.4.2 BGP does not validate the content of advertisements

1. Possible harmful changes to advertisement:
   (a) Remove ASes form the AS-PATH (Attract sources that normally try to avoid a certain AS or make path seem shorter); detection only possible if direct path between the ASes next to the removed one doesn't actually exist.
   (b) Add AS-identifier to AS-PATH. (If something actually arrives at the AS whose identifier was added loop detection might be triggered: DoS attack or block unwanted traffic; Or pretend that adjacent AS has more connections than it actually has); Detection possible by the inserted AS (if it sees the route) or adjacent ones.
   (c) Add AS-identifier to end of AS-PATH (E.g. to evade detection by adding a valid AS at the end)
2. Export routes it shouldn't (e.g. according to customer-provider guidelines); Defense by filtering routes by prefix and AS path
3. Making inconsistent routes
   (a) Peers need consistent routes: same prefix at all peering points, and with same AS-PATH length
   (b) Can violate to trick neighbor into keeping packets longer (i.e. in contrast to the "hot potato" routing rule) or just a configuration error
   (c) Defense: Analyze BGP updates or traffic for signs of inconsistency
4. Current defense practices:
   (a) Apply BCPs (best common practices): Securing BGP peering sessions between routers + filtering routes by prefix and AS path + packet filters to block unexpected control traffic
   (b) Problem: Doesn't solve the fundamental problems (e.g. Can't tell who owns IP + Can't tell if AS path invalid or not + Can't be sure data packets follow the chosen route)
   (c) Solutions: RPKI (Resource public key infrastructure): provides per prefix certificates; enables signatures of first AS hop (=Route Origin Authorization ROA)

9.4.3 Preventing and Detecting Errors

1. BGPsec: secure BGP
   (a) BGPsec provides: Route attestations (i.e. The AS path of a BGP update message can be verified: order and removal or addition of ASes): Each AS on the path signs the message, when receiving the signs can be verified.
   (b) BGPsec uses RPKI: Each IP-prefix is assigned a certificate by Regional Internet Registry (RIR). This certificate can be used to identify the first AS hop with ROA.
   (c) Deployment Challenges:
      i. Need registry of all ip-prefix owners
      ii. Need to know pubic key of any given AS
      iii. Need to add digital signatures to BGP

iv. Operations need to be done quickly (to avoid delay)
v. Difficult to deploy incrementally

2. Incrementally deployable solutions:
   (a) Should be Backward compatible (no changes to router hard-/software & no changes at other ASes) and good for early adopters (Security benefits for ASes that deploy the solution)
   (b) Option 1. Detecting suspicious routes
      i. Monitor BGP update messages and use past updates as registry to find suspicious activity (e.g. monitor which AS usually announces which address block or which AS paths are common)
      ii. Out-of-band detection mechanisms to generate alerts & reports
   (c) Options 2. Avoiding suspicious routs
      i. Don't immediately use suspicious routs: prefer already seen ones and delay adoption of new routs
      ii. Delay enables opportunity to understand the route & some attacks aren't permanent

9.4.4 Attacks on the data plane

1. Routers forward data packets supposedly alone the path chosen by control plane. BUT: not guaranteed
2. How? Control routers along the path or hijack prefix (guess password of router, buy compromised router, be router owner)
3. Dropping packets in data plane:
   (a) Hard to detect if still sending routing announcement
   (b) Harder if only some specific packets are dropped
   (c) Even harder if traffic is slowed down (Not really different from normal congestion)
4. Send packets in a different direction
5. Direct packets to a different destination: Impersonate legitimate destination or snoop traffic and forward along normal route
6. Detection with traceroute if more time than usually is needed or end-to-end checks with certificates

9.5 SCION