Abstract Computer Networks

Roman Böhringer, June 2019

Table of contents

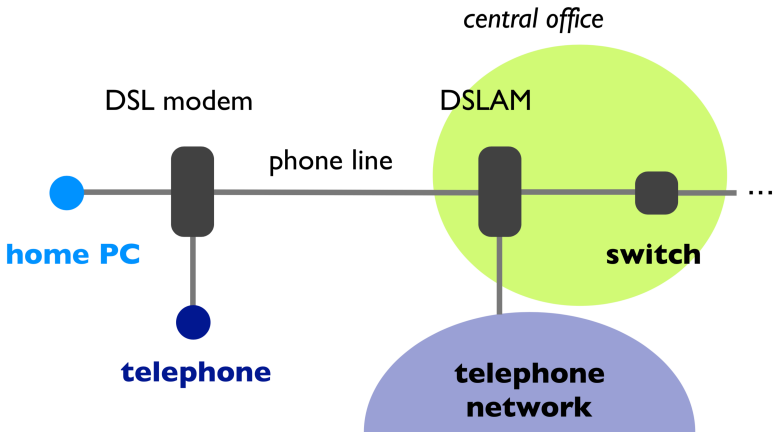
1 Overview & Principles

1.1 Basic principles

A network is made of three basic components:

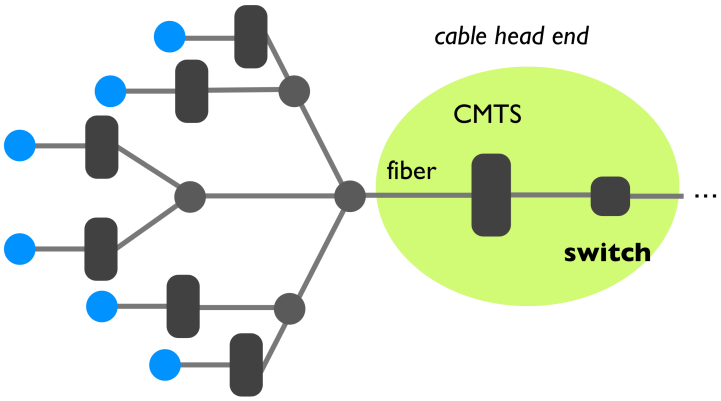
- End-systems that send and receive data
- Switches and routers that forward data to the destination
- Links that interconnect end-systems and switches / routers

Digital Subscriber Line (DSL) brings high bandwidth to households over phone lines. The last mile usually looks like this:



It is composed of three channels, a downstream data channel, an upstream data channel and a 2-ways phone channel. The downstream data channel is usually much faster.

An alternative technology via cable TV are Cable Access Technologies (CATV). Many households (unlike ADSL) share the medium:

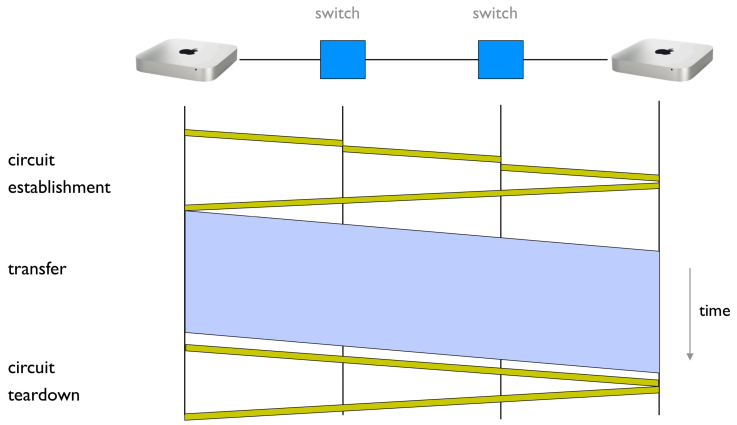


A common abstraction for networks are graphs.

1.2 Sharing of limited network resources

Limited network resources can be shared by making reservations (reserve the bandwidth you need in advance) or by on-demand sharing (send data when you need). Multiplexing for reservation-based sharing is done at the flow-level and at the packet-level for on-demand sharing. For reservation-based sharing, the peak rate (P) must be reserved. The level of utilization is the average rate (A) divided by the peak rate, i.e. A/P. On-demand reservation can usually achieve a higher level of utilization. When P/A is small (e.g. voice traffic), reservation makes sense. When P/A is big (i.e. very bursty traffic like data applications), capacity is wasted.

Reservation-based sharing is usually implemented using circuit-switching. Circuit switching relies on the resource reservation protocol that establishes a circuit within each switch. Therefore, a data transfer looks like this:



Efficiency heavily depends on how utilized the circuit is once established.

Advantages of circuit switching are predictable performance and simple & fast switching (once the circuit is established). Disadvantages are that it is inefficient if traffic is bursty / short, it needs a complex circuit setup / teardown (that adds delay) and it requires a new circuit upon failure.

In packet switching, data transfer is done using independent packets that contain the destination. Packet switching relies on buffers to absorb temporary overload. Packet switching can route around trouble.

Advantages of packet switching are that it provides an efficient use of resources, it's simpler to implement and it routes around trouble. Disadvantages are that the performance is unpredictable, and it requires buffer management and congestion control.

1.3 Communication

To exchange data, a set of network protocols is used. A protocol is like a conversational convention that says who should talk next and how they should respond. To provide structure to the design of network protocols, network designers organize protocols in layers. Internet communication can be decomposed in 5 independent layers:

1. Physical
2. Link
3. Network
4. Transport
5. Application

Each layer provides a service to the layer above (by using the services of the layer directly below it):

	layer	service provided
L5	Application	network access
L4	Transport	end-to-end delivery (reliable or not)
L3	Network	global best-effort delivery
L2	Link	local best-effort delivery
L1	Physical	physical transfer of bits

Each layer has a unit of data:

	layer	role
L5	Application	exchanges messages between processes
L4	Transport	transports segments between end-systems
L3	Network	moves packets around the network
L2	Link	moves frames across a link
L1	Physical	moves bits across a physical medium

And each layer (except for layer 3), is implemented with different protocols. The Internet Protocol (IP) acts as a unifying network layer. Each layer takes messages from the layer above and encapsulates with its own header and / or trailer. Hosts must implement all layers; intermediate devices may only implement certain layers. Routers act as a layer 3 gateway and implement layer 2 and layer 3. Switches act as a layer 2 gateway and implement only layer 2.

Each layer is implemented with different protocols and technologies (and in software, in hardware or both). The interfaces between layers should be durable. Interfaces that change often imply broken layering. The end-to-end principle says that endpoints still need to implement checks for reliability, no matter how reliable the communication system becomes. But implementing reliability in the network can still enhance performance (but doesn't reduce end-host complexity, does increase network complexity and can impose an overhead for apps that don't need it).

The fate-sharing principle says that when storing state in a distributed system, it should be co-located with entities that rely on that state. In that case, the state is lost if those entities fail, in which case it doesn't matter.

1.4 Characterization

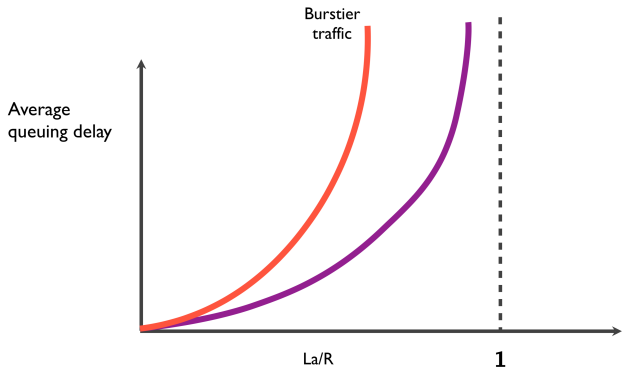
A network connection is characterized by its delay, loss rate and throughput. The delay is about how long it does take for a packet to reach the destination. The loss is about what fraction of packets sent to a destination are dropped and the throughput is about at what rate the destination is receiving data from the source.

Each packet suffers from several types of delays at each node along the path:

- **Transmission delay:** The amount of time required to push all of the bits onto the link. It is calculated as packet size (#bits) / link bandwidth (#bits/sec)

[The bandwidth limits of copper cables exists because of their behavior with high frequency signals where the impedance increases as frequency increases which makes it harder to differentiate the signal from noise. The velocity of the propagating wave does not limit the bandwidth. For fiber-optics, noise is a limiting factor as well, but way higher bandwidths can be achieved]

- **Propagation delay:** The amount of time required for a bit to travel to the end of the link. *[2/3 of the speed of light for fiber]*
- **Processing delay:** Due to switch internals, tends to be tiny.
- **Queuing delay:** The amount of time a packet waits (in a buffer) to be transmitted on a link. It can be only characterized with statistical measures because it varies from packet to packet.
  Let La (bit/sec) be the average bits arrival rate and R the transmission rate of the outgoing link. When the traffic intensity, La/R is greater than 1, the queue (and the queuing delay) will increase without bound. When La/R is smaller than 1, queueing delay can still happen (and depends on the burstiness of traffic):



Loss happens because in practice, queues are not infinite. If the queue is persistently overloaded, it will eventually drop packets. There's usually a correlation between avg. loss and latency / minimum RTT.

The throughput is the rate at which a host receives data. The average throughput (#bits/sec) is calculated as data size (#bits) / transfer time (sec). To compute the throughput, one has to consider the bottleneck link (and the intervening traffic).

Generally speaking, throughput increases & delay decreases, except for propagation delay (which has a lower bound due to the speed of light). But one can move the content closer to the user (content delivery networks) to reduce propagation delays.

1.5 Brief History of the Internet

In 1961-64, queueing-theoretic analysis of packet switching was done in which the value of statistical multiplexing was demonstrated. In 1965, the first computer network at MIT was built (connected with a telephone line). In 1969, the ARPANET was built using a packet switch (Interface Message Processor). In 1973, a paper on interconnecting networks was presented by Cerf / Kahn (that became TCP / IP). In 1982, the ISO / OSI reference model was released and in 1983, ARPANET transitioned to TCP / IP. Ethernet was developed in 1976 (and the spanning tree protocol in 1985). DNS was developed in 1983 and EGP / IGP (that became eBGP / iBGP) were developed in 1984 for hierarchical routing. In 1990, ARPANET ended.

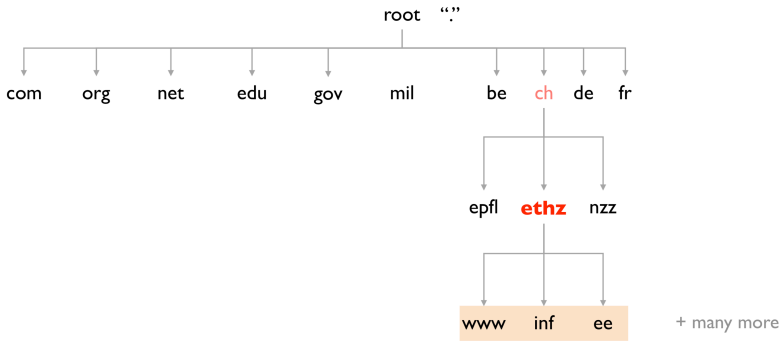
The birth of the web was in 1989 (by Tim Berners Lee at CERN), the first search engine was invented in 1993 (Excite) and Google reinvented search in 1998.

2 Applications

2.1 DNS

The Domain Name System (DNS) is a distributed database that enables translation between names (domain names) and addresses (IP addresses). Names can be mapped to more than one IP and an IP can be the target of multiple domain names.

DNS uses a hierarchical naming structure. The top-level domain (TLDs) sit at the top. Domains are subtrees and can have subtrees themselves:



The DNS system is hierarchically administrated. The root is managed by IANA, top-level domains are managed by different organizations, ch e.g. by the Swiss Education & Research Network (SWITCH).

The DNS infrastructure is hierarchally organized, as well. 13 root servers (distributed over the world) serve as root. To scale root servers, BGP anycast is used. Several locations can announce the same prefix and routing will deliver the packets to the "closest" location. TLDs server are managed professionally by private or non-profit organizations. The bottom of the hierarchy is managed by Internet Service Providers or locally.

Every server knows the address of the root servers (which is required for bootstrapping the systems), usually in a file named.root (https://www.internic.net/domain/named.root). To ensure availability, each domain must have at least a primary and a secondary DNS server. Besides availability, DNS queries can be load-balanced and on timeout, clients use alternate servers.

Using DNS relies on two components. The resolver software that triggers resolution process (and sends the request to a local DNS server) and a local DNS server (that is usually near the endhosts and either configured statically or dynamically via DHCP). DNS queries and replies use UDP (port 53), reliability is implemented by repeating requests. There are multiple types of records:

Records	Name	Value
A	hostname	IP address
NS	domain	DNS server name
MX	domain	Mail server name
CNAME	alias	canonical name
PTR	IP address	corresponding hostname

DNS resolution can either be recursive or iterative. When performing a recursive query, the client offloads the task of resolving to the server. When performing a iterative query, the server only returns the address of the next server to query. To reduce resolution times, DNS relies on caching. DNS servers cache responses to former queries (clients / applications do as well) and

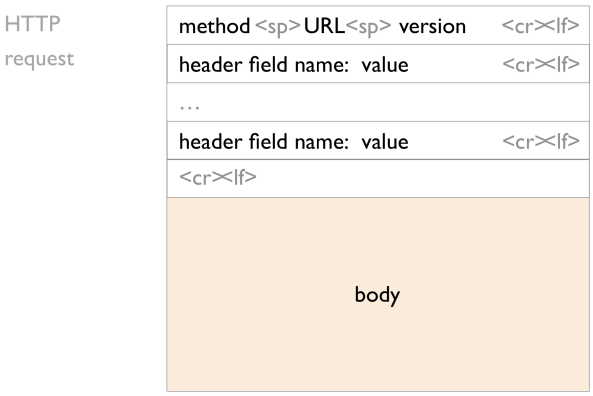
authoritative servers associate a lifetime to each record, the Time-To-Live (TTL). The records can only be cached for TTL seconds.

2.2 The Web

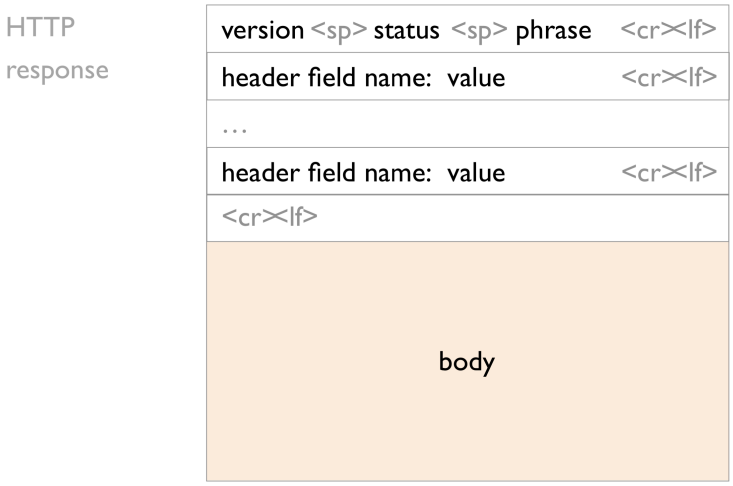
The web was founded in ~1990 by Tim Berners-Lee. His goal was to provide distributed access to data. The World Wide Web (WWW) is a distributed database of "pages" that are linked together via the Hypertext Transport Protocol (HTTP). It is made of three key components: Infrastructure (Clients / Browser, Servers, Proxies), Content (Objects that are organized in Web sites) and Implementation (URL, HTTP).

URLs are used to name content. A Uniform Resource Locator (URL) refers to an Internet resource and consists of `protocol://hostname[:port]/directory_path/resource`. The protocol can be for instance HTTP(S), FTP or SMTP. The hostname can be the DNS name or an IP address. Ports are optional, if no one is provided, the default of the protocols's standard (HTTP: 80, HTTPS: 443) is assumed. `directory_path/resource` is used to identify the resource on the destination.

HTTP is used to transport content. It is a rather simple synchronous request-response protocol. It is layered over a bidirectional byte stream (almost always TCP), text-based and stateless (i.e. no info about past client requests are maintained). HTTP clients make requests to servers that look like this:



The method can be GET (to return a resource), HEAD (to return headers only), POST (to send data to a server) and many more. The URL is provided relative to the server (e.g. /index.html). The version can be 1.0, 1.1 or 2.0. There are multiple request headers that are commonly used, e.g. Authorization info, acceptable document types / encoding, From (user email), If-Modified-Since, Referrer (cause of the request) or the user agent. The HTTP server answer the clients' requests, an answer looks like this:



The status is a 3-digit response code, where different ranges have different meanings:

	3 digit response code			reason phrase
Status	1XX	informational		
	2XX	success	200	OK
	3XX	redirection	301	Moved Permanently
			303	Moved Temporarily
			304	Not Modified
	4XX	client error	404	Not Found
	5XX	server error	505	Not Found

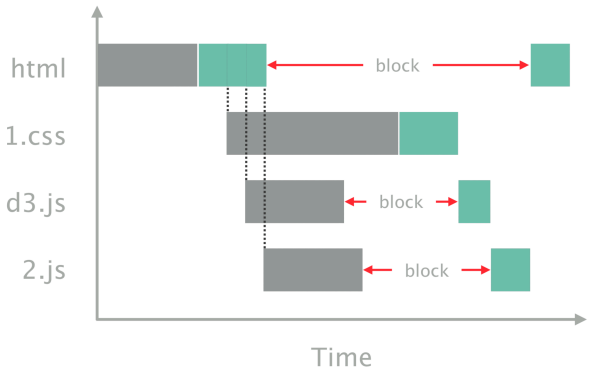
Some possible response headers are Location (for redirection), Allow (list of methods supported), Content encoding (e.g. gzip), Content-Length, Content-Type, Expires (for caching) or Last-Modified (for caching).

The advantages of a stateless protocol are that server-side scalability is much easier and failure handling is trivial. But some applications need state. HTTP makes the client maintain the state with cookies. The client stores a small state (on behalf of the server) and sends state in all future requests to the server.

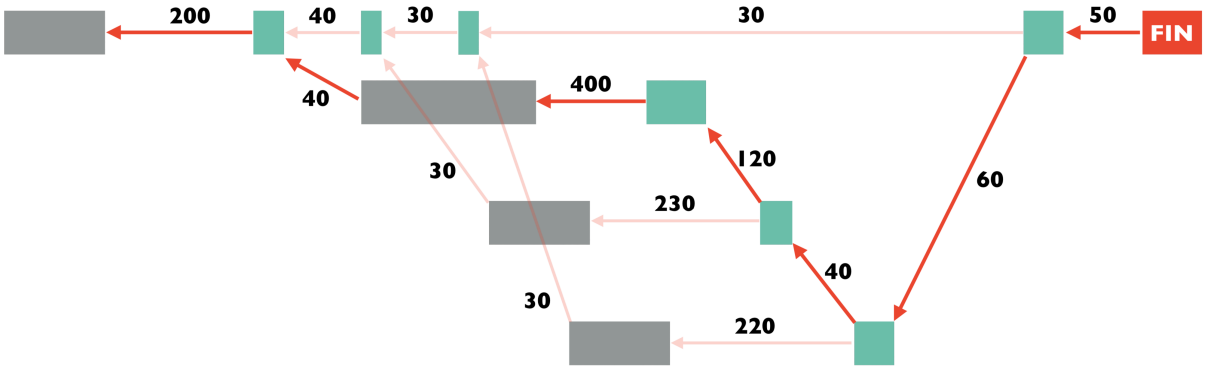
2.2.1 Performance

Web pages usually have a lot of external dependencies. Dependencies can interact with each other (e.g. a JavaScript file that relies on some CSS file). Because of that, a browser has to parse / execute dependencies in order (unless they are specified as `async`), which can cause blocking:

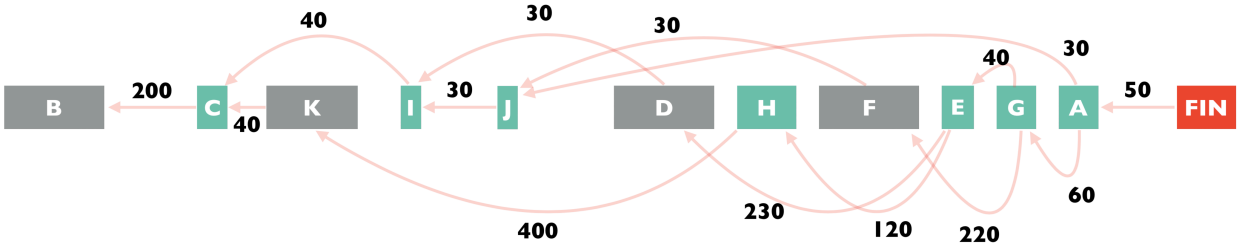
```
<html>
  <body onload="done();">
    <link src='1.css'>
    <script src='d3.js'></script>
    <script src='2.js'></script>
    <div id="main"></div>
  </body>
</html>
```



In a time-annotated dependency graph, the loading time is the longest path from start to finish:



To get the longest path in a systematic way, one can sort the graph topologically (when there is an edge from u to v, u should come before v) and process it in reverse-sort order. The longest path can then be calculated by calculating the maximum distance from start to each node (which is based on the distance to the previous node) step-by-step, e.g.:



d_B		0	d_H	d_K + 400	640
d_C	d_B + 200	200	d_F	d_J + 30	300
d_K	d_C + 40	240	d_E	d_H + 120, d_D + 230	760
d_I	d_C + 40	240	d_G	d_F + 220, d_E + 40	800
d_J	d_I + 30	270	d_A	d_G + 60, d_J + 30	860
d_D	d_I + 30	270	d_{FI}	d_A + 50	910

This is called critical path analysis. Speeding up any task on the critical path will speed up the end-to-end process or / and expose a different critical path.

There are many possibilities to speed up Web browsing:

- Simplify, restructure, redesign Web pages: Compress using gzip / more efficient image codecs like WebP, use In-line JS and CSS and tag `async` resources.
- Use faster computing devices: Javascript performance has significantly increased in the last years.
- Increase network bandwidth: Significant gains only up to a few Mbps.
- Make network RTTs smaller
- Simplify network protocols: Naïve HTTP opens one TCP connection for each object, which causes 2n RTTs for fetching n objects. A solution is to use multiple TCP connections in parallel. Another solution (default in HTTP/1.1) is to use persistent connections across multiple requests. The overhead of connection set-up and teardown is avoided, and it allows TCP to learn a more accurate RTT estimate (and therefore, a more precise timeout value). Furthermore, it allows the TCP congestion window to increase, to leverage a higher bandwidth.
  It's also possible to pipeline requests & replies asynchronously, on one connection. Requests and responses are batched to reduce the number of packets, multiple requests can be packed into one TCP segment. The different approaches lead to different RTTs (assuming n small objects):

	# RTTs
one-at-a-time	~2*n*
M concurrent	~2*n*/M
persistent	~*n*+1
pipelined	2

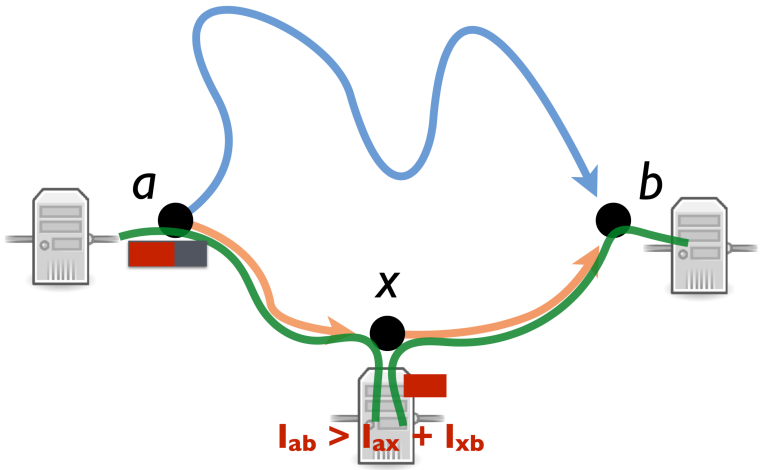
- Make network RTTs smaller
- Caching & replication: Highly popular content largely overlaps. But a significant portion of the HTTP objects are "uncachable" (dynamic data, scripts, cookies, SSL, advertising). HTTP enables a client to validate cached objects. The server hints when an object expires as well as the last modified date of an object. The client can conditionally request a resource using "If-Modified-Since" in the header of the HTTP request. The server can compare this against "last modified" and return "Not modified" or "OK" (with the latest version). Caching is performed at different locations:
  - Client: Browser cache
  - Close to the client: Forward proxy and Content Distribution Network (CDN). Forward proxies cache documents close to clients, decreasing network traffic, server load and latencies (typically done by ISPs or enterprises).
  - Close to the destination: Reverse proxy. A reverse proxy helps because many clients request the same information which increases servers and network's load. They cache documents close to servers, decreasing their load (usually done by the content providers).

  The idea behind replication is to duplicate popular content all around the globe. This spreads load on servers, places content closer to clients and also helps to speed up some uncacheable content. Often, content distribution networks (CDNs) are used for replication.

2.2.2 Content distribution network

CDNs spread the content servers globally, network the sites and the origin and direct clients to appropriate servers.

Because traffic in the internet not always follows the optimal route, overlay routing can be used to redirect traffic to an intermediate node, e.g. in the following situation:



To direct clients to the appropriate servers, a DNS-based solution or BGP Anycast can be used. BGP Anycast advertises the same IP prefix from different locations, which gives less flexibility and control. A DNS-based solution returns different IP addresses based on the client's geo-localization and possibly server load. If the client isn't using the local DNS resolver, EDNS Client Subnet (ECS) can be used to indicate the clients' location to the name server.

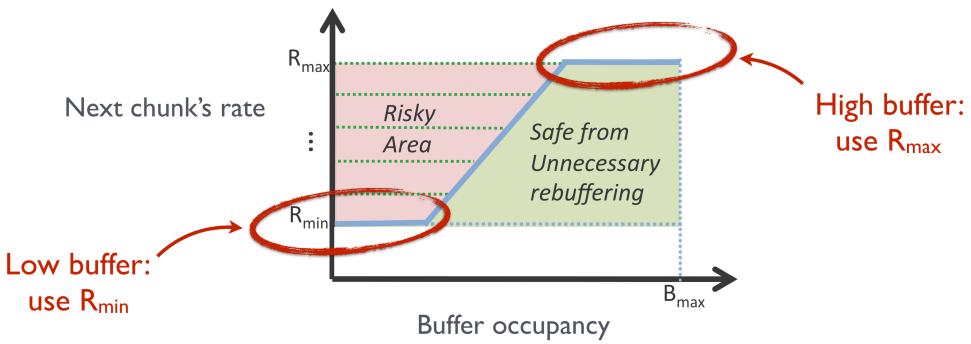
For replication, CDNs can use pull caching (cache as a direct result of clients requests) or push replication (cache when expecting a high access rate).

2.3 Internet Video

When streaming, one wants to achieve the highest quality without buffering. To do this, the video is encoded in multiple bitrates and replicated using a content delivery network. The video player picks the bitrate adaptively by estimating the available bandwidth and picking a lower bitrate.

For encoding, there is a "bitrate ladder" which indicates which bitrate is required for which resolution. This depends on the type of video, because certain videos (e.g. with little movement) don't require very high bitrates for high resolutions. Chunks of the video are created at each bitrate and the client gets metadata about chunks via a "Manifest".

For adaptation, a simple approach is to decrease the requested bitrate as soon as the network capacity is lower than the current rate. Another approach is to use buffer-based adaptation. When the buffer is nearly full, use a large rate. When it's nearly empty, use a small rate, i.e.:



A problem in this approach is the startup phase. One solution is to pick a rate based on the immediate past throughput.

3 Transport

3.1 Requirements

In the transport layer, we need:

- Data delivery to the correct application: IP just points towards the next protocol, the transport layer needs to demultiplex incoming data (done with ports)
- Files or byte-streams abstractions for applications: The network deals with packets and the transport layer needs to translate between streams and packets.
- Reliable transfer (if needed)
- Not overloading the receiver
- Not overloading the network

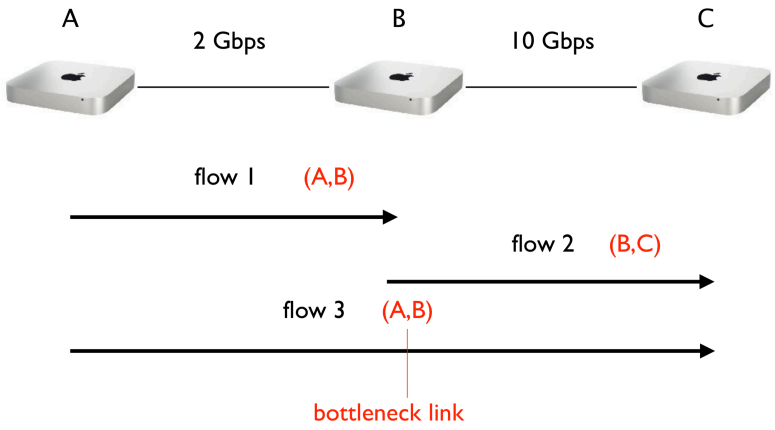
3.2 Reliable Transport

Reliable transfer has four goals:

- **Correctness:** Ensure data is delivered, in order, and untouched. More formally, a reliable transport design is correct if a packet is always resent if the previous packet was lost or corrupted. A packet may be resent at other times.
  A reliable transport mechanism is correct if and only if it resends all dropped or corrupted packets.
- **Timeliness:** Minimize the time until data is transferred. There is a clear tradeoff between timeliness and efficiency in the selection of the timeout value. Small timers can cause unnecessary retransmissions and large timers can cause a slow transmission. A better timeliness can be achieved by sending multiple packets at a time (not waiting for the ACK of the first) with a sequence number inside each packet. The sender and receiver have a buffer to store packets sent & not acknowledged / store out-of-sequence packets received. But the receiver can be overwhelmed, why a mechanism for flow control is needed. One way to do that is a sliding window. The sender keeps a list of the sequence # it can send (known as the sending window) and the receiver keeps a list of the acceptable sequence # (known as the receiving window). Sender and receiver negotiate the window size (with sending window <= receiving window).
  Assuming infinite buffers, the window should be the size of the bandwidth-delay product to maximize timeliness.
- **Efficiency:** Use the bandwidth optimally. The efficiency of the protocol essentially depends on two factors:
  - **Receiver feedback:** How much information does the sender get?
    ACKing individual packets provides detailed feedback (the sender knows the fate of each packet), but triggers unnecessary retransmission upon losses (loss of an ACK packet requires a retransmission). It is not sensitive to reordering and can be implemented as a simple window algorithm.
    With cumulative ACKs, the highest sequence number for which all the previous packets have been received, is ACKed. This enables recovery from lost ACKs but reordering causes confusion and a sender has incomplete information about which packets have arrived (which causes unnecessary retransmissions).
    Full Information Feedback lists all packets that have been received by sending the highest cumulative ACK plus any additional packets. Advantages are that the sender has complete information (with a more resilient form of individual ACKs) but there is a certain overhead (which lowers efficiency), e.g. when there are large gaps between the received packets.
    Behavior upon losses: How does the sender detect and react to losses?
    One way to detect losses is to use timers. But they can also be detected by relying on ACKs. With individual ACKs, a sender can infer that a packet is / may be lost, when he doesn't get the ACK for the packet but for k subsequent packets. With full information, missing packets (gaps) are explicit. With cumulative

ACKs, duplicated ACKs are a sign of isolated losses. Upon receiving k duplicates ACKs, a sender can trigger a resend, but has to decide what to resend (only the one after the duplicate ack number or all after it?)

- **Fairness:** Play well with concurrent communications. Exact notions of fairness are debatable, but a universally agreed upon minimal goal is to avoid starvation. Equal per flow is good enough for this. But simply dividing the available bandwidth doesn't work in practice since flows can see different bottlenecks, e.g.:



A max-min fair allocation is such that the lowest demand is maximized, then the second lowest demand is maximized, and so on… It can be easily computed:

1. Start with all flows at rate 0
2. Increate the flows until there is a new bottleneck in the network
3. Hold the fixed rate of the flows that are bottlenecked
4. Go to step 2 for the remaining flows

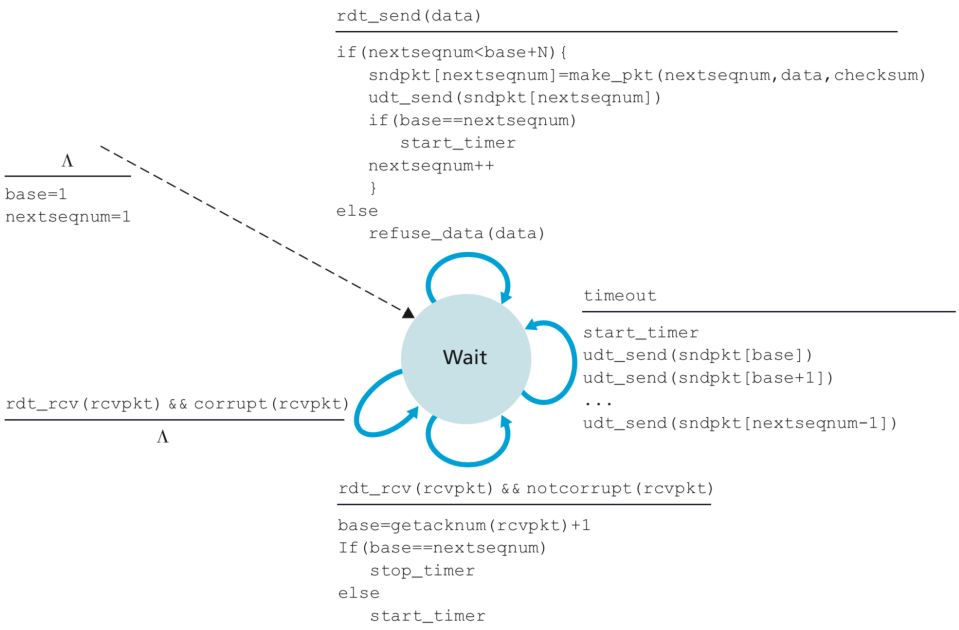
The max-min fair allocation can be approximated by slowly increasing W until a loss is detected.

Besides loss, packets can also get

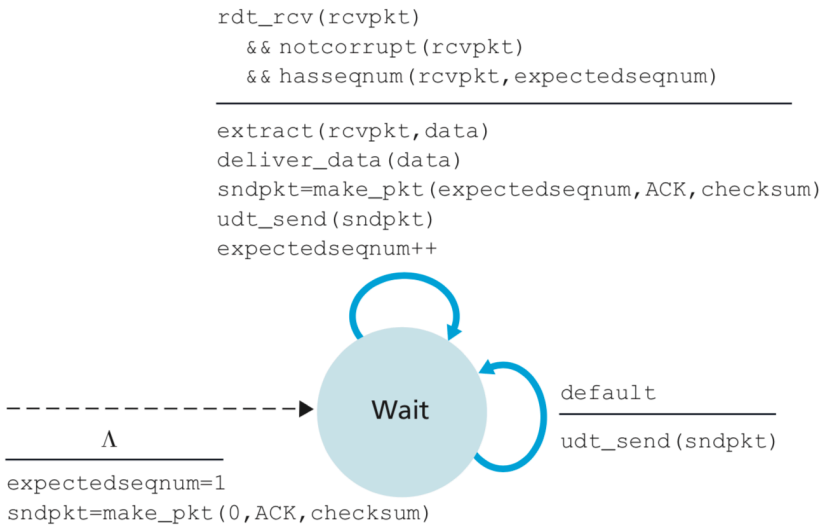
- corrupted (can be solved by relying on a checksum and treating corrupted packets as lost)
- reordered (no problem for individual ACKs and full feedback, creates duplicate ACKs for cumulative ACKs which can trigger retransmissions)
- delayed (can create useless timeouts)
- duplicated (leads to duplicate ACKs, which is unproblematic for individual ACKs / full feedback, but can cause problems for cumulative ACKs).

3.2.1 Go-Back-N

Go-Back-N (GBN) is a simple sliding window protocol using cumulative ACKs. The receiver delivers packets in-order to the upper layer. For each received segment, he ACKs the last in-order packet delivered (cumulative). The sender uses a single timer to detect loss which is reset at each ACK. Upon timeout, all W packets (starting with the lost one) are resent. The sender is implemented using the following finite state machine:



The receiver is implemented using this finite state machine:



3.2.2 Selective Repeat

Selective Repeat (SR) avoids unnecessary retransmissions by using per-packet ACKs. Each packet is acknowledged (no matter if it's in order or not). Out-of-order packets are buffered. A per-packet timer is used to detect loss and upon loss, the lost packet is resent. So the sender actions / events are:

1. *Data received from above.* When data is received from above, the SR sender checks the next available sequence number for the packet. If the sequence number is within the sender's window, the data is packetized and sent; otherwise it is either buffered or returned to the upper layer for later transmission, as in GBN.
2. *Timeout*. Timers are again used to protect against lost packets. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout. A single hardware timer can be used to mimic the operation of multiple logical timers [Varghese 1997].
3. *ACK received.* If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. If the packet's sequence number is equal to `send_base`, the window base is moved forward to the unacknowledged packet with the smallest sequence number. If the window moves and there are untransmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

And the receiver actions / events:

1. *Packet with sequence number in* `[rcv_base, rcv_base+N-1]` *is correctly received.* In this case, the received packet falls within the receiver's window and a selective ACK packet is returned to the sender. If the packet was not previously received, it is buffered. If this packet has a sequence number equal to the base of the receive window (`rcv_base` in Figure 3.22), then this packet, and any previously buffered and consecutively numbered (beginning with `rcv_base`) packets are delivered to the upper layer. The receive window is then moved forward by the number of packets delivered to the upper layer. As an example, consider Figure 3.26. When a packet with a sequence number of `rcv_base=2` is received, it and packets 3, 4, and 5 can be delivered to the upper layer.
2. *Packet with sequence number in* `[rcv_base-N, rcv_base-1]` *is correctly received.* In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged.
3. *Otherwise.* Ignore the packet.

3.3 TCP

TCP (Transmission Control Protocol) provides:

- Data delivery to the correct application: Demultiplexing using application identifiers (ports)
- Files or byte-streams abstractions for applications: Segmentation and reassembly
- Reliable transfer (if needed): ACKs, checksums
- Not overloading the receiver: "Flow control" via receiver window
- Not overloading the network: "Congestion control" via sender window

In the Internet, reliability is ensured by the end hosts, not by the network. The goals are to keep the network simple and dumb (to make it relatively "easy" to build and operate a network) and to keep applications as network "unaware" as possible. Reliability is implemented in-between the application and the network, in the layer 4 (just above the network layer).

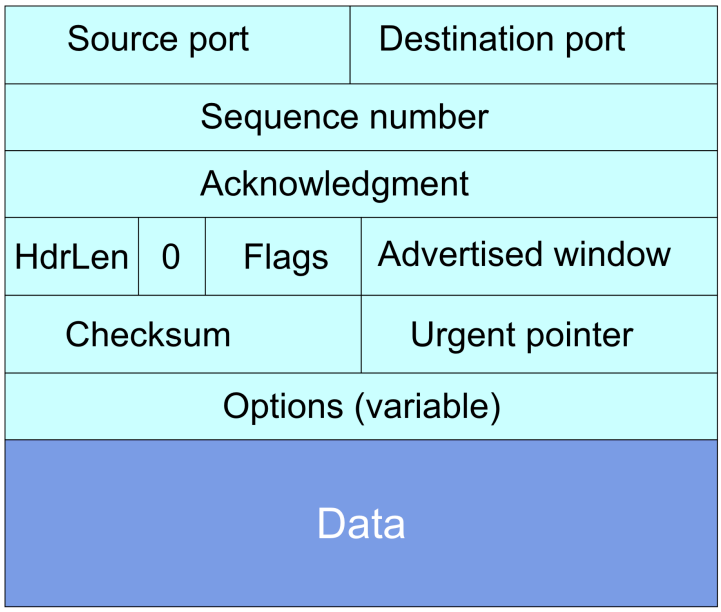
Reliability requires keeping state. Each bytestream is called a connection or a session (each with their own connection state). The state is in the hosts, not in the network.

Transport protocols do not provide delay and / or bandwidth guarantees (cannot be offered by transport, would require support at IP level) or sessions that survive change-of-IP-address (except Multipath TCP).

The basic components of reliability are:

- ACKs: TCP uses byte sequence numbers to identify payloads
- Checksums: TCP does checksum over TCP header + data and a pseudo header that contains fields of the IP header (e.g. sender / recipient)
- Timeouts and retransmissions: TCP retransmit based on timeouts and duplicate ACKs. The timeout is based on an estimate of the RTT.

TCP uses cumulative acknowledgements but selective ACKs (full information ACKs) are also supported. TCP uses a single timer that is set after each payload is ACKed. When the timer goes off, the payload is resent, and the timeout period doubled. The TCP header contains these fields:



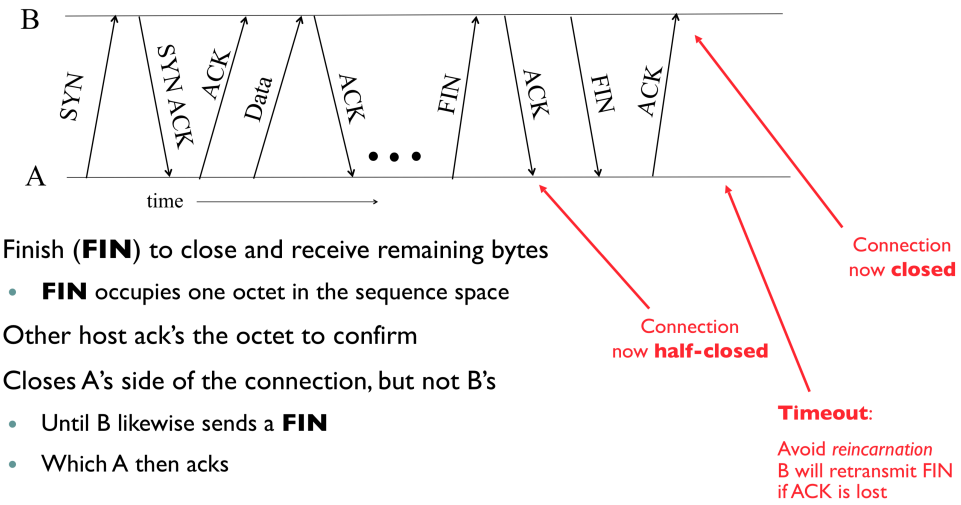
A TCP segment contains several bytes. The IP packet isn't allowed to be bigger than the Maximum Transmission Unit (MTU) whereas the TCP packet (IP packet with a TCP header and data inside) isn't allowed to be larger than Maximum Segment Size (MSS) bytes. The TCP header is greater or equal to 20 bytes, the MSS is calculated as MSS=MTU-(IP header)-(TCP header).

TCP uses byte values for sequence numbers. The initial sequence number (ISN) is chosen at random (for security reasons; otherwise an attacker could inject packets easily) and afterwards, the sequence number is increased by the number of bytes in a packet. Therefore seq. no - ISN gives the byte offset in the stream. The ACK sequence number is the number of the next expected byte.

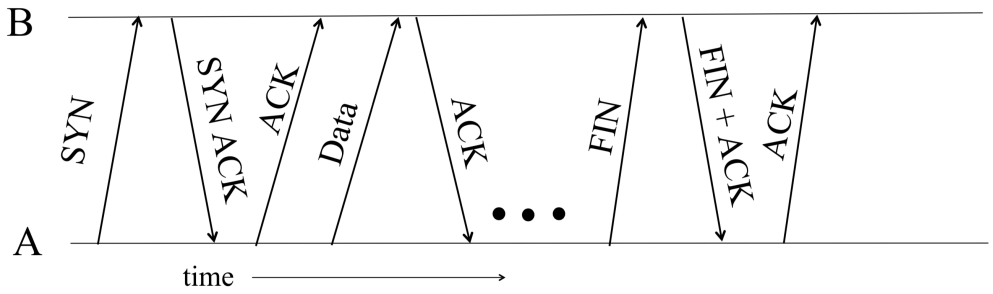
The receiver sends the advertises Window W which means the sender can send W bytes beyond the next expected byte to prevent the sender from overflowing the buffer. If W/RTT is lower than the bandwidth, a transfer has speed W/RTT.

For connection establishment, TCP uses a 3-way handshake. Each host tells its ISN to the other. The first host sends a SYN (for "synchronize sequence numbers"), the second host returns a SYN acknowledgement (SYN ACK) and the first host sends an ACK to acknowledge the SYN ACK. If no SYN-ACK is received, the sender can retransmit the SYN after a timer (3 seconds according to the RFC).

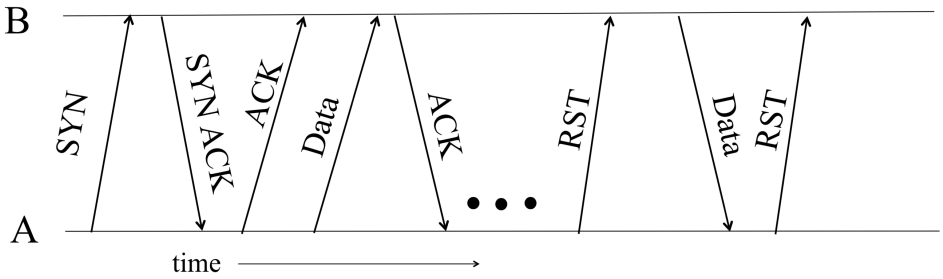
In a normal connection termination, both ends send a FIN (one side at a time) which gets ACKed.



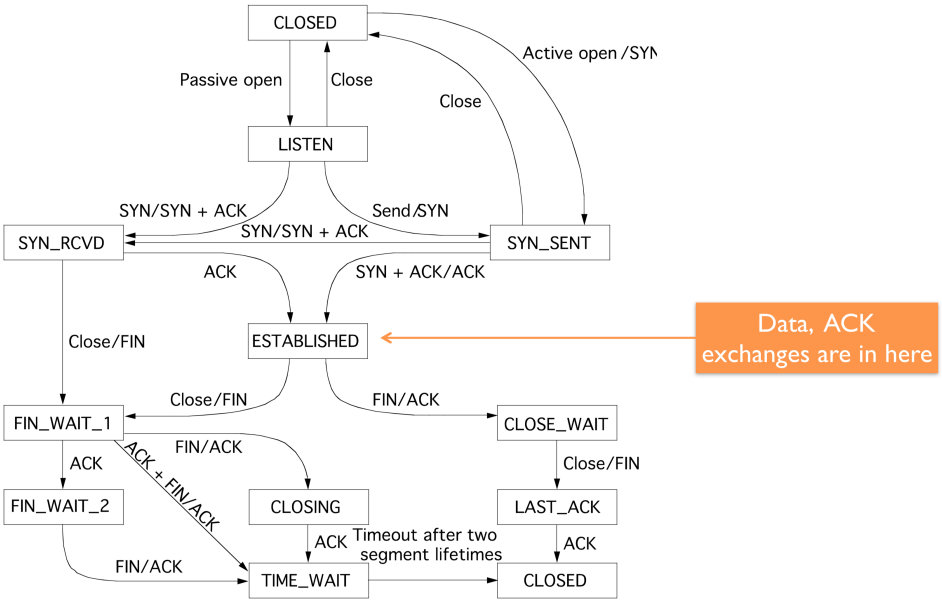
B can also set FIN with their ACK of A's FIN:



In an abrupt termination (e.g. because a process crashed), a RST is sent which isn't confirmed. If more data arrives, another RST is sent:

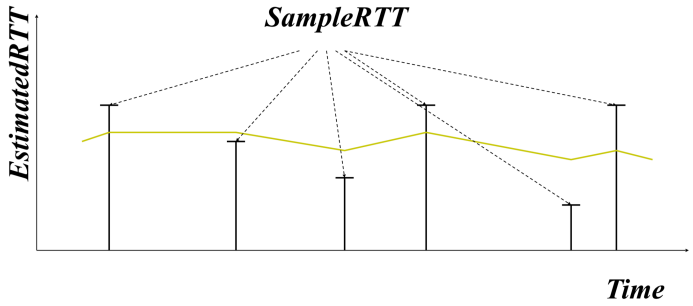


The TCP state diagram looks like this:



To estimate the RTT (for the timeout), exponential averaging of RTT samples is used:

$$SampleRTT = AckRcvdTime - SendPacketTime$$
$$EstimatedRTT = \alpha \times EstimatedRTT + (1-\alpha) \times SampleRTT$$
$$0 < \alpha \leq 1$$



For the calculation, a system needs to differentiate between real ACKs and ACKs of retransmitted packets. In the Karn/Partridge Algorithm, SampleRTT is only for original transmissions measured. The timeout value (RTO) is calculated as 2 * EstimatedRTT. Every time the RTO timer expires, it is doubled. When a new measurement comes in (i.e. a new successful original transmission), RTO is collapsed back to 2 * EstimatedRTT.

The Jacobson/Karels improvement also calculates the exponential average of the deviation and calculates the RTO as EstimatedRTT + 4 * EstimatedDeviation.

In practice, a minimum RTO value (1 second according to the RFC, 200ms in Linux) is used.

Cumulative ACKs are also used as a sign of an isolated loss. On receiving k (k=3 for TCP) duplicate ACKs, the packet is resent. This is known as “fast retransmit”.

3.3.1 Congestion Control

Congestion collapse is the phenomenon when sudden load increases the round-trip time, causing retransmissions, increasing the round-trip time, …

Knee point after which

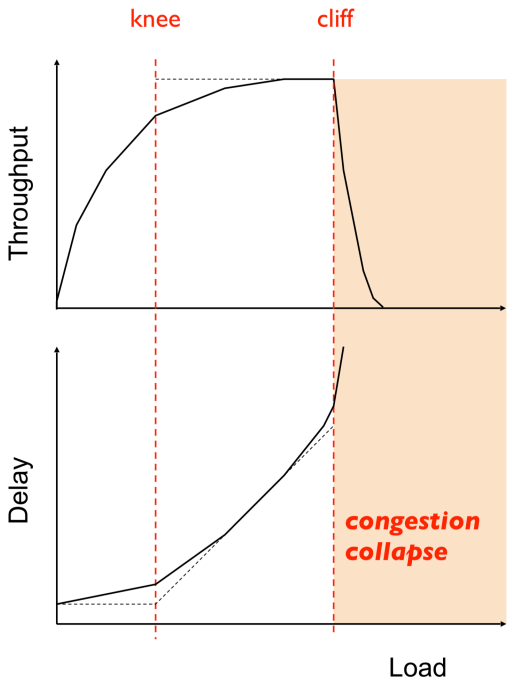
throughput increases slowly

delay increases quickly

Cliff point after which

throughput decreases quickly

delay tends to infinity



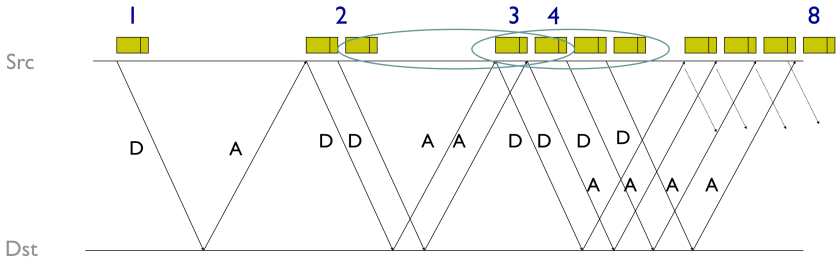
Congestion control aims at solving three problems:

1. Bandwidth estimation: How to adjust the bandwidth of a single flow to the bottleneck bandwidth?
2. Bandwidth adaptation: How to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth?
3. Fairness: How to share bandwidth "fairly" among flows, without overloading the network?

The difference to flow control is that flow control prevents one fast sender from overloading a slow receiver (with a receiving window) whereas congestion control prevents a set of senders from overloading the network with a congestion window. The sender window is set as minimum(congestion window, receiving window).

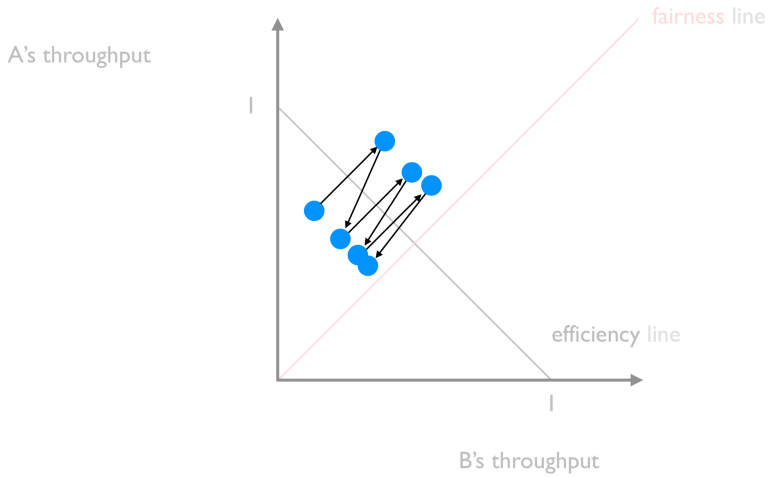
To detect congestion, packet loss is measured. Duplicate ACKs are a mild congestion signal whereas timeouts are a severe congestion signal. To quickly get a first-order estimate of the available bandwidth (known as slow start), CWND (congestion window) is set to 1 initially and increased by 1 upon receipt of an ACK. This corresponds to an exponential increase of CWND:

This increase phase, known as slow start,
corresponds to an… exponential increase of CWND!



slow start is called like this only because of starting point

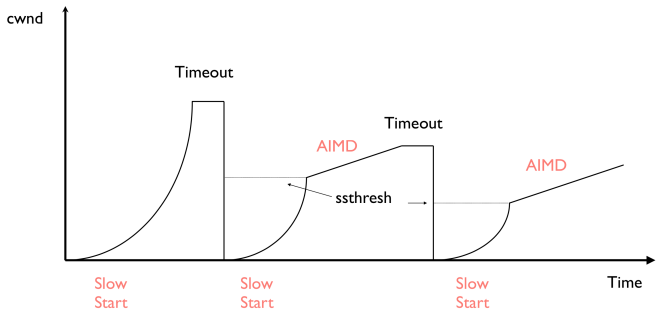
Afterwards, TCP uses additive increase/multiplicative decrease for adaptation of the congestion window. This is the only option that converges to fairness end efficiency:



The intuition behind this is that during increase, both flows gain bandwidth at the same rate and during decrease, the faster flow releases more.

To implement this, CWND is incremented by 1/CWND after each ACK. There exists a slow start threshold that defines when a sender leaves slow start and starts with AIMD. On timeout, this threshold is set to CWND/2.

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

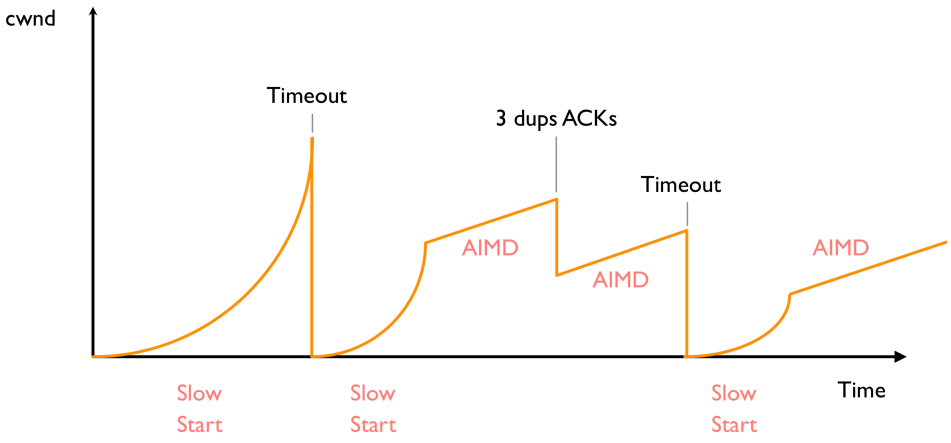


To prevent this, TCP switches back to AIMD without going all the way back to 0 after a fast retransmit (3 duplicate ACKs). The CWND is halfed instead.

The implementation in pseudo code looks like this:

Initially:
```
cwnd = 1
ssthresh = infinite
```
New ACK received:
```
if (cwnd < ssthresh):
    /* Slow Start*/
    cwnd = cwnd + 1
else:
    /* Congestion Avoidance */
    cwnd = cwnd + 1/cwnd
dup_ack = 0
```
Timeout:
```
/* Multiplicative decrease */
ssthresh = cwnd/2
cwnd = 1
```
Duplicate ACKs received:
```
dup_ack ++;
if (dup_ack >= 3):
    /* Fast Recovery */
    ssthresh = cwnd/2
    cwnd = ssthresh
```

Congestion control makes TCP throughput look like a “sawtooth”:



3.3.2 Sockets

Sockets are the application ←→ transport interface. Sockets are an operating system abstraction whereas ports are a networking abstraction. A socket is a software abstraction by which an application process exchanges network messages with the (transport layer in the) operating

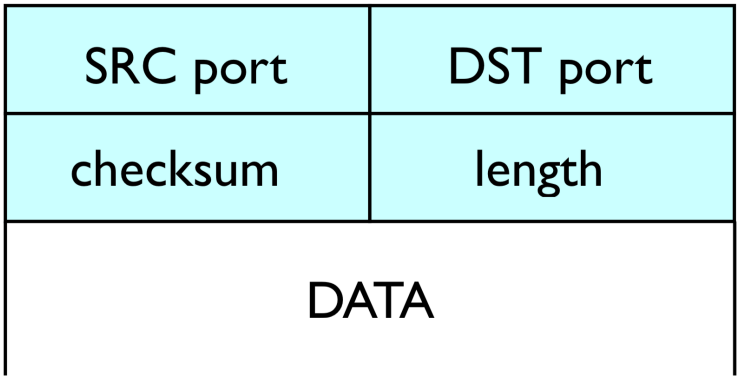
system. The type of UDP sockets is SOCK_DGRAM whereas the type of TCP sockets is SOCK_STREAM.

For UDP ports (SOCK_DGRAM), the OS stores (local port, local IP address) whereas for TCP ports (SOCK_STREAM), (local port, local IP, remote port, remote IP). When a server has a port he listens on, the socket is "cloned" on a connection request with the destination port / destination IP of the client.

Ports solve the problem of which app (socket) gets which packets. The OS stores the mapping between sockets and ports. "Well known" ports are between 0-1023, ephemeral ports (usually) between 1024-65535 and are given to clients (at random).

3.3.3 UDP

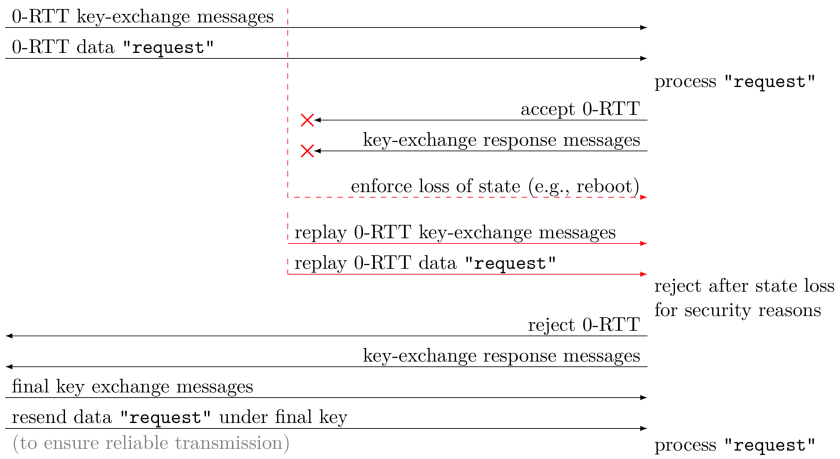
UDP (User Datagram Protocol) only provides data delivery to the correct application and optionally checksums. A UDP message is therefore very simple:



There are some reasons to use UDP. The overhead and delays of ordered, reliable delivery are avoided, there are no delays for connection establishment, there is finer control over what data is sent and when, no connection state / buffers / timers leading to greater scalability and the per-packet overhead is because of the small headers minimal. Some applications are DNS, Gaming or VoIP.

3.4 QUIC

QUIC stands for Quick UDP Internet Connections. HTTP/2 introduced multiple streams over one transport connection. But "head of line blocking" can occur where the last byte is not read by the application which causes the windows to not move forward. QUIC solves these issues by managing the windows per stream. Furthermore, it eliminates the "hello" by using transport cookies (and combines connection / TLS setup). But 0-RTT data transfer can enable replay attacks, where an attacker intercepts the 0-RTT acceptance and enforces loss of state at the server. When an attacker then replay the 0-RTT message, it is rejected by the server (because he has no state anymore) and a new key-exchange message is sent. This is accepted by the client, causing a second transmission of the data.



The problem is very hard to solve (global / temporal consistency on server side would be needed). But one can use only idempotent operations with 0-RTT or don't provide reliability with 0-RTT.

To enable mobility, a "connection ID" separate from the IP address is added.

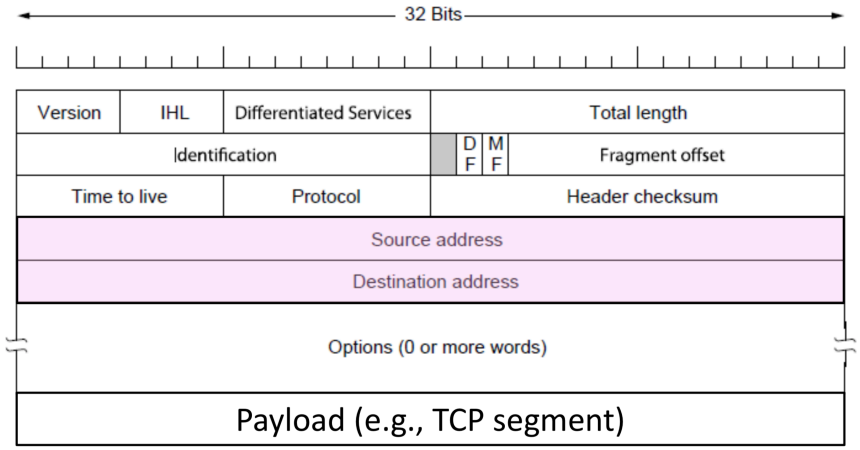
4 Network

The network layer builds on the link layer and provides service to the transport layer. Routers send packets over multiple networks. The challenges of the network layer are to scale to a global internet, heterogeneity (with IP for internetworking) and bandwidth control (lowest cost routing / Quality of Service).

Routing is the process of deciding in which direction to send traffic (control plane of network). It is network wide (global) and expensive. Forwarding is the process of sending a packet on its way (data plane of network). It is a node process (local) and fast.

There are two network service models. Datagrams / connectionless service (which IP implements) or virtual circuits / connection-oriented service (like a telephone call). Both models are implemented with store-and-forward packet switching where routers receive a complete packet, storing it temporarily (in an internal buffer; typically a FIFO queue) if necessary, before forwarding it. Statistical multiplexing is used to share the link bandwidth over time.

The Internet Protocol is the network layer of the internet. IPv4 carries a 32 bit src / dst address in each packet, the whole packet looks like this:



The process of connecting different networks together is called internetworking. Network may differ in a lot of ways, for instance the service model (datagrams, virtual circuits), addressing, QoS, packet sizes or security.

IP is the "narrow" waist of the internet because it supports many different links below and apps above. IPv4 uses 32-bit addresses that are written in "dotted quad" notation (four 8-bit numbers separated by dots). Addresses are allocated in blocks called prefixes where addresses in an L-bit prefix have the same top L bits. The first and last IP prefix aren't usable (network identifier / broadcast address), so a L-bit prefix has $2^{32-L} - 2$ host addresses. Prefixes are sometimes also specified using a mask. ANDing the address and the mask gives the prefix:

	Address	82.130.102.49
		01010010.10000010.01100110. 00110001
		11111111.11111111.11111111. 00000000
and	Mask	255.255.255.0
	Prefix	01010010.10000010.01100110. 00000000
		82.130.102.0 better written: 82.130.102/24

There are public IP addresses (that are valid destinations in the global internet and must be allocated before use) and private IP addresses (RFC 1918). Private IP addresses can be used freely within private networks, the ranges are 10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16. The allocation of public IP addresses follows a hierarchical process. The Internet Assigned Numbers Authority (IANA) delegates to regional internet registries (RIRs) and RIRs delegate to companies in their region (which assign them to their customers / hosts).

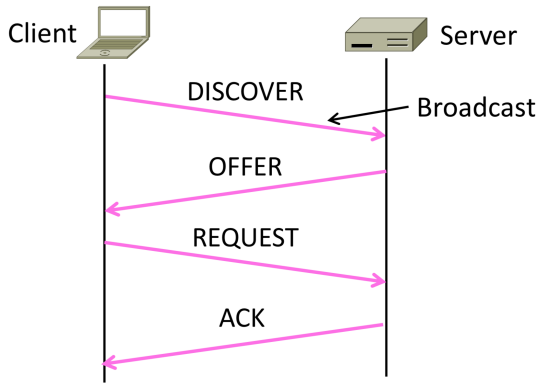
4.1 IP Forwarding

IP addresses on one network belong to the same prefix. Nodes use a table that lists the next hop for IP prefixes. Prefixes in the table might overlap (this combines hierarchy with flexibility). The longest matching prefix forwarding rule specifies that for each packet, the longest (most specific) prefix that contains the destination address must be used. 0.0.0.0/0 is a default route that catches all IP addresses. Other aspects of forwarding are:

- Decrement TTL value (to protect against loops)
- Header checksum checking (reliability)
- Large packet fragmentation (split to fit on next link)
- Send congestion signals

4.2 DHCP

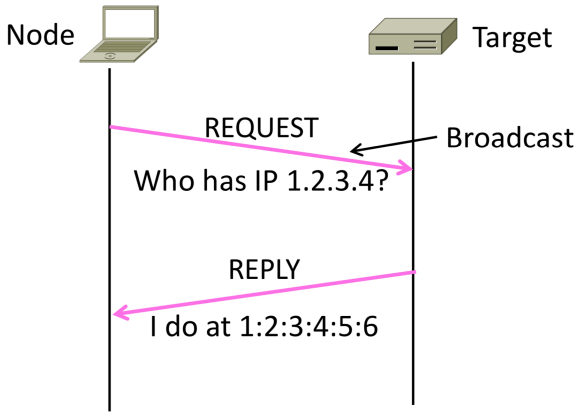
The dynamic host configuration protocol (DHCP) is a protocol for automatically configuring addresses. It leases IP addresses to nodes and provides other parameters too (network prefix, address of local router aka. "default gateway", DNS server, time server, …). It uses UDP ports 67 and 68. First, a node sends a broadcast message that is delivered to all nodes in the network. The whole process looks then like this:



To renew an existing lease, only a REQUEST, followed by an ACK is used.

4.3 ARP

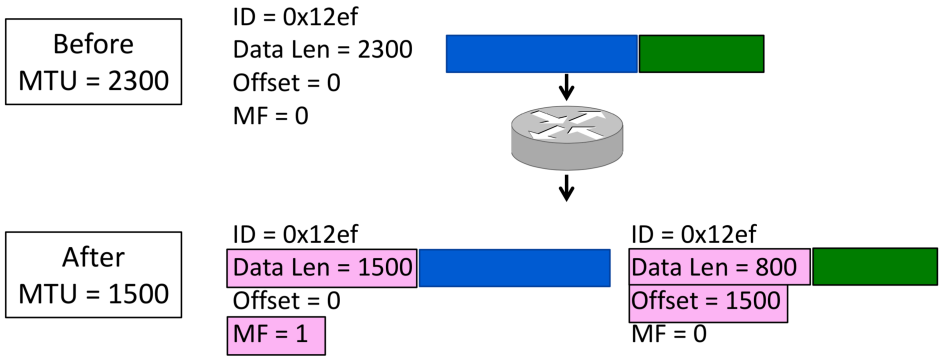
To send a frame over the local link, a node needs the link layer address (MAC address). The address resolution protocol (ARP) is used for this. A node sends a REQUEST as broadcast and gets a REPLY:



4.4 Packet Fragmentation

To connect networks with different maximum packet sizes, one needs to split up packets or discover the largest size to use. Different networks have different maximum packet sizes / MTUs (e.g. Ethernet 1.5K and WiFi 2.3K). The classic solution (fragmentation) is to split up large packets in the network if they are too big to send. The newer solution (discovery), that is employed today by IP ("path MTU discovery"), is to find the largest packet that fits on the network path and use it.

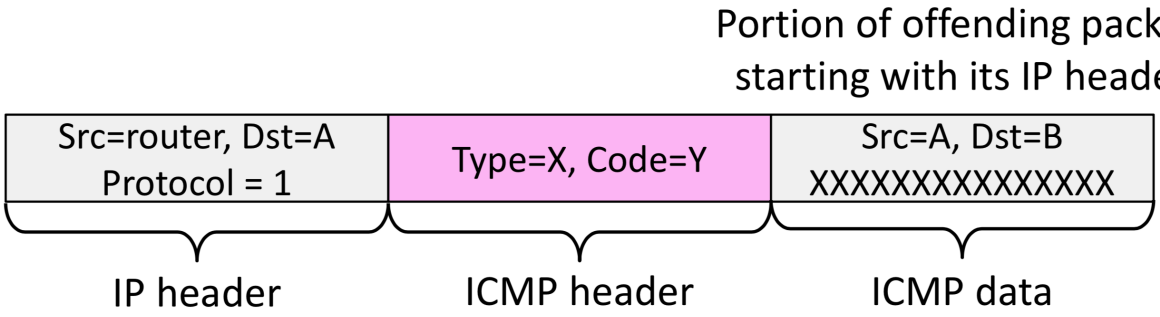
In IPv4 Fragmentation, routers fragment packets that are too large to forward. The receiving host reassembles the packets. The header fields "total length" (length of this datagram), "identification" (used to identify a datagram; links fragmented pieces together), "DF" (don't fragment; if set to 1, the packet is never fragmented), "MF" (if set to 1, more fragments follow this fragment) and "fragment offset" (number of data bytes preceding the fragment). For instance:



Path MTU discovery discovers the MTU by setting the DF flag. If the packet is too large, a router will provide feedback to the client (via ICMP) and tells what size would have fit.

4.5 ICMP

The Internet Control Message Protocol (ICMP) is a companion to IP and sits on top of it. It provides error report and testing. When a router encounters an error while forwarding, it sends an ICMP error report back to the IP source address. A ICMP message looks like this:



Some example messages are “Dest. Unreachable (Net or Host)”, “Dest. Unreachable (Fragment)”, “Time Exceeded (Transit)” or “Echo Request or Reply”.

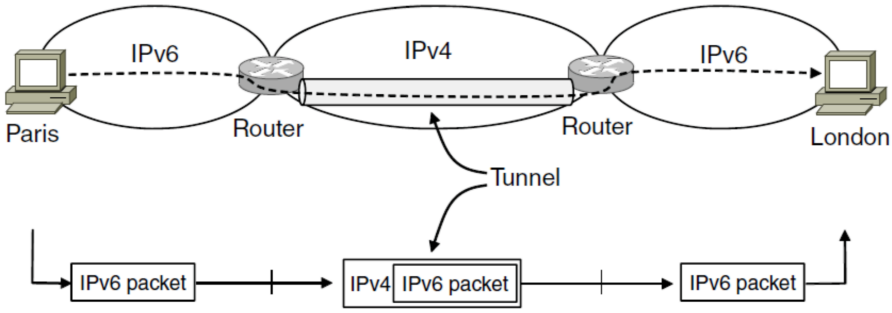
The IP header contains a TTL (Time to live) field to protect against forwarding loops. It is decremented on every router hop (and for every second the packet is kept in the router buffer) and a ICMP error is sent if it hits zero. Traceroute uses TTL / ICMP by sending probe packets with increasing TTL, starting from 1. The ICMP errors identify the routers on the path.

4.6 IPv6

To solve the IPv4 address space exhaustion, IPv6 was introduced. It features large addresses (128 bits) that are notated as 8 groups of 4 hex digits (16 bits). Leading zeros (per group) and groups of zeros (one time) can be omitted, e.g.:

Ex: 2001:0db8:0000:0000:0000:ff00:0042:8329
→ 2001:db8::ff00:42:8329

Because IPv6 is fundamentally incompatible with IPv4, the deployment is a big problem. One approach is tunneling where IPv6 is carried over IPv4, which allows to connect native IPv6 islands that are connected via IPv4:



The tunnel acts as a single link across the IPv4 network.

4.7 NAT

Network Address Translation (NAT) is widely used at the edges of the network. A NAT box connects an internal network to an external network. Many internal hosts are connected using few external addresses and there is a middlebox that "translates addresses". NAT works by keeping an internal / external table that typically uses IP address + TCP port. On a connect request from the internal network, the source IP / port is rewritten (and a new translation is created in the table). When a packet arrives at the external IP, the table is consulted, and the destination IP / port is rewritten.

Downsides of NAT are that connectivity has been broken (can only send incoming packets after an outgoing connection is set up) which causes difficulties for servers / peer-to-peer apps at home. It doesn't work that well when there are no connections (UDP apps) and it breaks apps that unwisely expose their IP addresses as ASCII within packet data (FTP). But it relieves much IP address pressure, it's easy to deploy and useful.

For enabling peer-to-peer applications, several NAT traversal techniques are used.

4.8 Routing

Routing adapts to equipment failures. There are other techniques with a different timescale / adaptation:

Mechanism	Timescale / Adaptation
Load-sensitive routing	Seconds / Traffic hotspots
Routing	Minutes / Equipment failures
Traffic Engineering	Hours / Network load
Network Provisioning	Months / Network customers

Different routing is used for different delivery models (Unicast, Broadcast, Multicast). In routing algorithms, there is a decentralized, distributed setting. All nodes are alike, there is no controller. Nodes only know what they learn by exchanging messages with neighbors and nodes operate concurrently.

Shortest path routing tries to find the "best" paths (shortest path) with link costs. "Best" can refer to latency, bandwidth, money or hops. This is approximated by a cost function where for each link, a cost is assigned. The best nodes between two nodes is then the path that has the lowest total cost (in case of ties, one is randomly picked). Subpaths of shortest paths are also shortest paths. Therefore, only the destination is needed to follow shortest paths and each node only needs to send to the next hop. Because of this, the forwarding table at a node only lists the next hop for each destination.

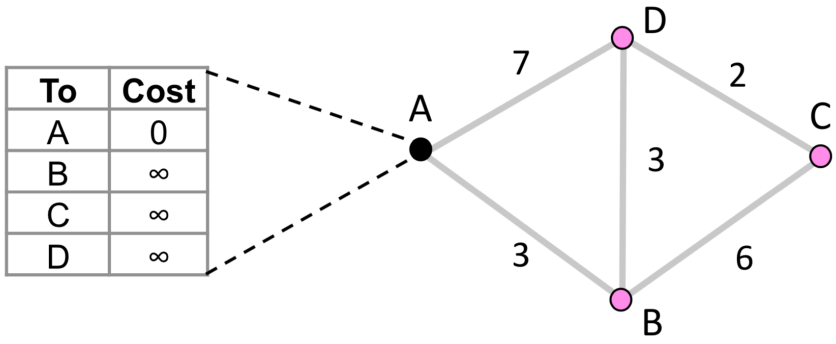
The shortest paths from a source can be calculated with Dijkstra's algorithm:

- Initialization: Mark all nodes tentative, set distances from source to 0 for source and infinity for all other nodes.
- While tentative nodes remain:
  - Extract the node N with the lowest distance.
  - Add link to N to the shortest path tree.
  - Relax the distances of neighbors of N by lowering any better distance estimates.

The runtime depends on the data structure (time for extracting min-cost node), but is generally superlinear in network size. The algorithm requires the complete topology.

4.8.1 Distance Vector Routing

Distance vector routing is a simple, early routing approach that is for instance used in RIP. It is based on a distributed Bellmann-Ford algorithm. In this approach, each node maintains a vector of distances (and next hops) to all destinations. The vector is initialized with 0 cost to self and infinity to other destinations. It is periodically sent to the neighbors. The vector is updated for each destination by selecting the shortest distance heard (after adding the cost of the neighbor link). For instance, with the initial vector:

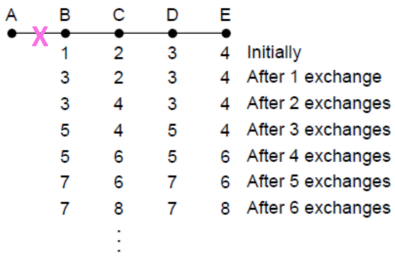


In the first round, A updates the cost to B / D after getting their vector:

To	B says	D says		B +3	D +7		A learns Cost	Next
A	∞	∞		∞	∞		0	--
B	0	∞	→	3	∞	→	3	B
C	∞	∞		∞	∞		∞	--
D	∞	0		∞	7		7	D

= learned better route

When a node fails, there are no more exchanges and the other nodes forget the node. But partitions are a problem and can lead to the "count to infinity" scenario. In this scenario, a node chooses (B in the following example) an alternative route with him in the path (which he doesn't know) after a link failure. The other node (C in the following example) then updates the cost and this keeps going to "infinity":



There are heuristics to address this issue, for instance "Split horizon with poisoned reverse": Split horizon means that routes learned from a neighbor are omitted in updates sent to that neighbor. Poisoned reverse is an implementation of that were the metric is set to infinity for routes learned from a neighbor that are sent back to that neighbor. But there are still topologies where these heuristics don't work.

RIP (Routing Information Protocol) is a distance vector protocol with hop count as metric. Infinity is 16 hops (which limits the network size). The routers send vectors every 30 seconds.

4.9 Link State Routing

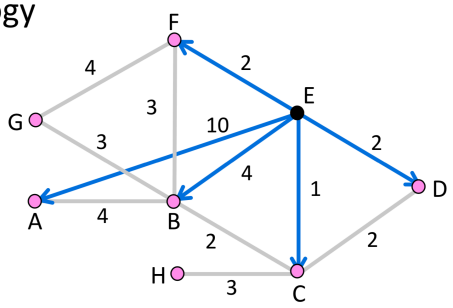
Link-State routing trades more computation than distance vector for better dynamics. It is widely used in practice, e.g. in OSPF and IS-IS. Link-State algorithms proceed in two phases:

1. Nodes flood topology in the form of link state packets (that describes their portion of the topology, i.e. which link costs to other nodes they have):

Each node floods link state packet (LSP) that describes their portion of the topology

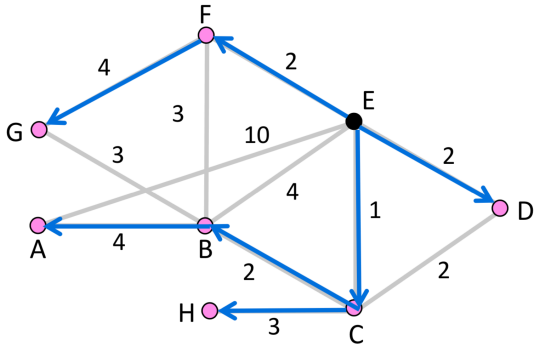
Node E's LSP flooded to A, B, C, D, and F

Seq. #	
A	10
B	4
C	1
D	2
F	2



2. Each node computes its own forwarding table (by running Dijkstra or equivalent). By combining all link state packets, each node has the full topology and simply runs Dijkstra. The forwarding table is computed from sink / source tree (union of all shortest path towards / from the node). For example:

Source Tree for E (from Dijkstra)



E's Forwarding Table

To	Next
A	C
B	C
C	C
D	D
E	--
F	F
G	F
H	C

On a change, the network is flooded with updated LSPs and routes are re-computed. In case of a link failure, both nodes notice and send updated LSPs. In case of a node failure, all neighbors notice a link has failed which causes all links to the node to be removed.

The LSPs include an age and are forgotten when they aren't refreshed for some time. The comparison with Distance vector algorithms looks like this:

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

One form of multipath routing is to extend the shortest path model by keeping a set if there are ties. The source / sink "tree" then becomes a directed acyclic graph. With ECMP (Equal Cost Multipath Routing), the packets from a given source / destination pair should be forwarded on the same path. A source/destination pair is called a flow, the flow identifier is mapped to a single next flow.

4.9.1 Flooding

A simple flooding method is to send an incoming message on to all other neighbors and to remember the message (using source and sequence number) so that it is only sent once over each link. To make flooding reliable, stop-and-wait is used where the receiver acknowledges, and the sender resends if needed.

4.9.2 Security

An attacker only needs to control one router / link to perform an attack. He then can intercept traffic or do a DoS attack. The solution is to use cryptographic authentication i.e. to send authenticated announcements and cryptographically protect topology information.

4.9.3 OSPF

OSPF splits an AS into different areas. Each area independently runs the link-state routing algorithm. Areas are connected exclusively via a special backbone area. There are internal routers (that belong to a single area), backbone routers (with a connection to the backbone area), area border routers (belong to the backbone area and at least one other area) and AS boundary routers (routers that are connected to other ASes, internal or backbone routers). Area border routers / AS boundary routers distribute additional LSAs of their connected area / AS. OSPF supports equal-cost multipath.

IS-IS is very similar to OSPF:

	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

4.10 Hierarchical Routing / BGP

IP prefixes are one technique to scale routing. Another is to route first to network regions and then to prefixes. The penalty is longer paths because outside a region, nodes have one route to all hosts within the region.

4.10.1 IP Prefix Aggregation / Subnets

Multiple subnets can be joined (aggregated) to reduce the size of forwarding tables. It's also possible to internally split one less specific prefix into multiple more specific prefixes (subnets) and to only send one prefix to the rest of the internet.

4.10.2 BGP

For Inter-domain routing, path-vector protocols like BGP (the Border Gateway Protocol) are used.

The internet is a network of networks, referred to as Autonomous Systems (AS). ASes exchange information about the IP prefixes they can reach (directly or indirectly) using BGP. BGP needs to solve three key challenges: Scalability (huge number of networks / prefixes), privacy (networks don't want to divulge internal topologies / their business relations) and policy enforcement (networks needs to control where to send and receive traffic). BGP relies on path-vector routing to support flexible routing policies and avoid count-to-infinity. The entire AS-level path is advertised instead of distances. Each AS prepends itself to the path when it propagates announcements.

The internet topology is shaped according to business relationships. ASes connect only if they have a business relationship. There are 2 main business relationships: customer/provider and peer/peer. A customer pays a provider to get internet connectivity. The amount paid is based on peak usage, usually according to the 95th percentile rule (top 5% values are removed, bill is then according to the highest value). Peers don't pay each other for connectivity, they do it out of common interest (e.g. providers that exchange a lot of traffic with each other). Generally, a path is only allowed if all ASes on the path have a financial incentive. For this reason, peers do not transit traffic between each other (the "middle" peer would not get any money). Customers do not transit traffic between their providers.

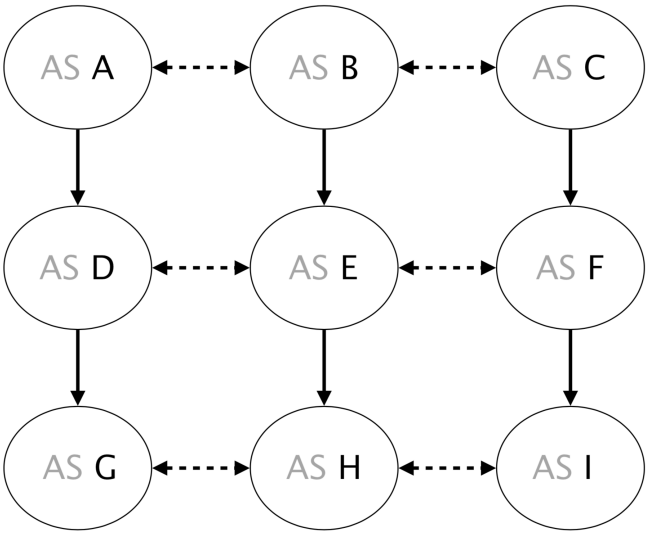
These policies are defined by constraining which BGP routes are selected (which path to use?) and which are exported (which path to advertise?). The selection controls outbound traffic. The rule is, that for a destination p, prefer routes coming from:

- Customers over
- Peers over
- Providers

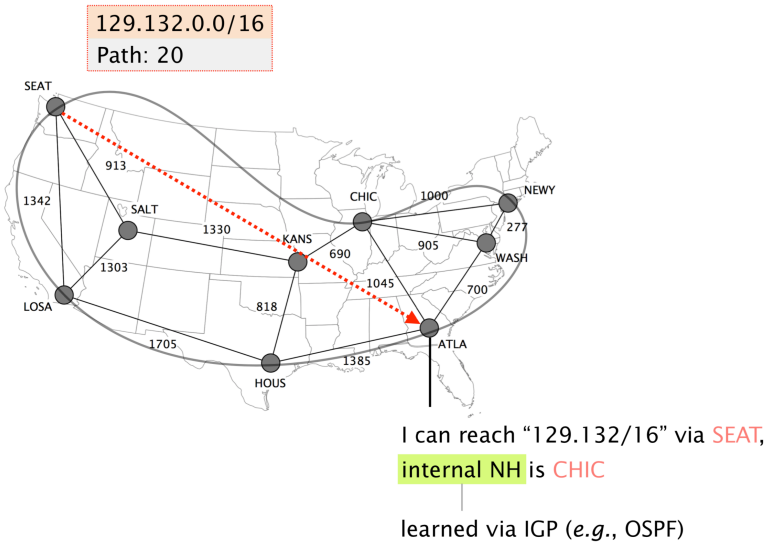
Business relationships conditions route exportation. Routes coming from customers are propagated to everyone else, routes coming from peers / providers are only propagated to customers:

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

Because of this policies, Tier-1s must be connected through a full-mesh of peer links. The following network is partitioned (G can't communicate with I):



There are two types of BGP sessions. External BGP (eBGP) sessions connect border routers in different ASes. They are used to learn routes to external destinations. Internal BGP (iBGP) sessions connect the routers in the same AS. They are used to disseminate externally-learned routes internally. There still needs to be an Intra-Domain Routing Protocol like OSPF so that internal routers can reach each other:

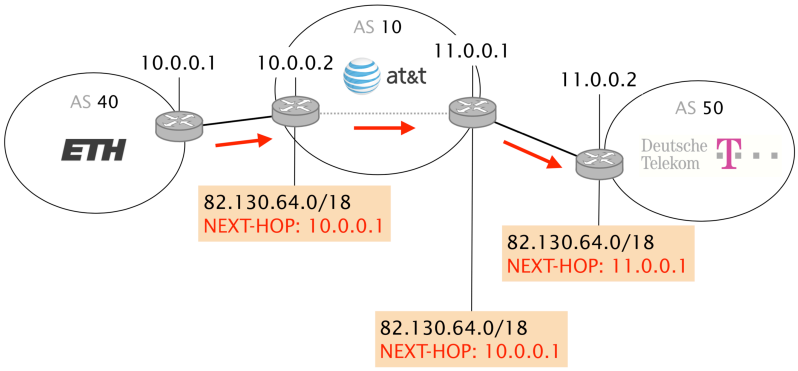


The internally disseminated routes (using iBGP) are then announced externally again using eBGP sessions. BGP is a rather simple protocol that is composed of four basic messages:

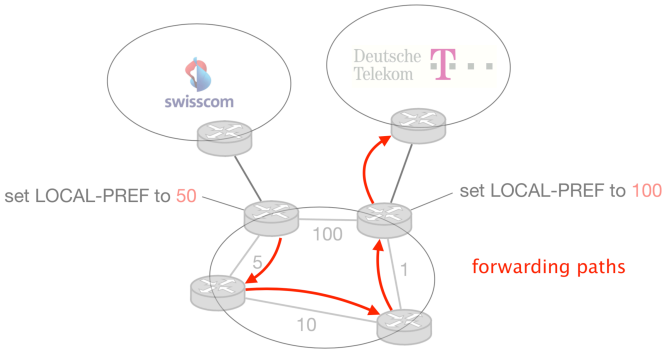
type	used to...
OPEN	establish TCP-based BGP sessions
NOTIFICATION	report unusual conditions
UPDATE	inform neighbor of a new best route a change in the best route the removal of the best route
KEEPALIVE	inform neighbor that the connection is alive

BGP UPDATEs carry an IP prefix and a set of attributes that describe route properties. They are either local (only seen on iBGP) or global (seen on iBGP and eBGP). Some attributes are:

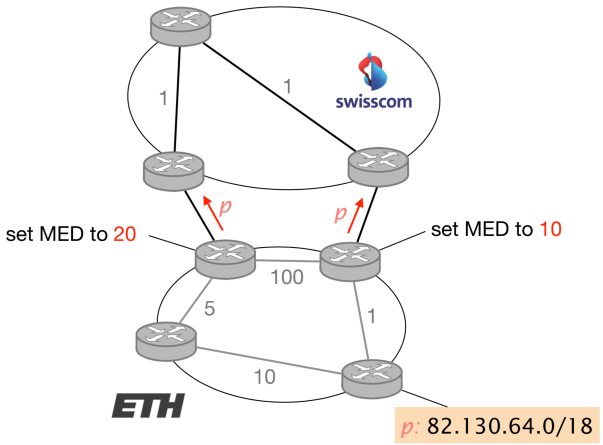
- **NEXT-HOP:** A global attribute which indicates where to send the traffic next. It is set when the route enters an AS and doesn't change within the AS.



- **AS-PATH:** A global attribute that lists all the ASes a route has traversed (in reverse order)
- **LOCAL-PREF**: A local attribute (set at the border) that represents how "preferred" a route is. If a route to an AS has a higher LOCAL-PREF, all routers use this route to the AS (even if another egress would be closer):



- **MED**: The MED (Multi-Exit Discriminator) is a global attribute which encodes the relative "proximity" of a prefix with respect to the announcer. A lower MED value indicates closeness and is preferred over a higher value. For instance, if ETH would want to get traffic to p over the right router / route, it would set the MED to a lower value than the left one:



The MED value can be ignored (Swisscom could just set its LOCAL-PREF for the left router higher and send all traffic over this way). The network which is sending the traffic always has the final word when it comes to deciding where to forward, the network which is receiving the traffic can just influence remote decision, not control them. The MED can furthermore only be used to influence decisions when there exist multiple connections towards the same AS. It cannot be used to influence over which AS incoming traffic arrives.

Given the set of all acceptable routes for each prefix, the BGP decision process elects a single route (BPG is therefore a "single path protocol"). The rules are to prefer routes…

with higher LOCAL-PREF

with shorter AS-PATH length

with lower MED

learned via eBGP instead of iBGP

with lower IGP metric to the next-hop

with smaller egress IP address (tie-break)

The reason to prefer routes learned via eBGP and with lower IGP metric is to direct traffic as quickly as possible out of the AS (early exit routing / hot potato routing). This leads to assymetric routing where traffic often doesn't flow on the same path in both directions.

To implement their selection policy, operators define input filters which manipulate the LOCAL-PREF. Routes learned from customers have the highest LP, then those from peers and then those from providers. To implement the export rules, a mix of import / export filters is used.

BGP suffers from many rampant problems:

- **Reachability:** Connectivity doesn't imply reachability.
- **Security:** There are no security considerations in the BGP specifications. ASes can advertise any prefixes, arbitrarily modify route content or forward traffic along different paths.
- **Convergence:** With arbitrary policies, BGP may fail to converge. But if all AS policies follow the customer/peer/provider rules, it is guaranteed to converge.
- **Performance:** BGP path selection is mostly economical and not based on accurate performance criteria.
- **Anomalies:** BGP configuration is hard to get right and often manually configured (which can cause misconfigurations).
- **Relevance:** The world of BGP policies is rapidly changing because ISPs are mainly eyeballs talking to content networks. Transit becomes less important and less profitable.

4.10.3 Security

BGP doesn't validate the origin of advertisements which allows attackers to hijack IP addresses. An attacker can announce a more specific prefix and gets the traffic for it. Furthermore, the content of advertisements is not validated which allows to create bogus AS paths (e.g. some that appear shorter). Routes can also be missing / inconsistent (not announced at all peering points), for instance to trick neighbors into "cold potato".

There's a proposal for secure BGP (BGPsec) with origin authentication and cryptographic signatures. Address attestations are signed and route attestations are signed by each AS as the route traverses the network.

Besides the control plane, the data plane can also be attacked. Packets can simply be dropped or traffic can be slowed down. They can also be sent in a different direction.

4.11 SCION

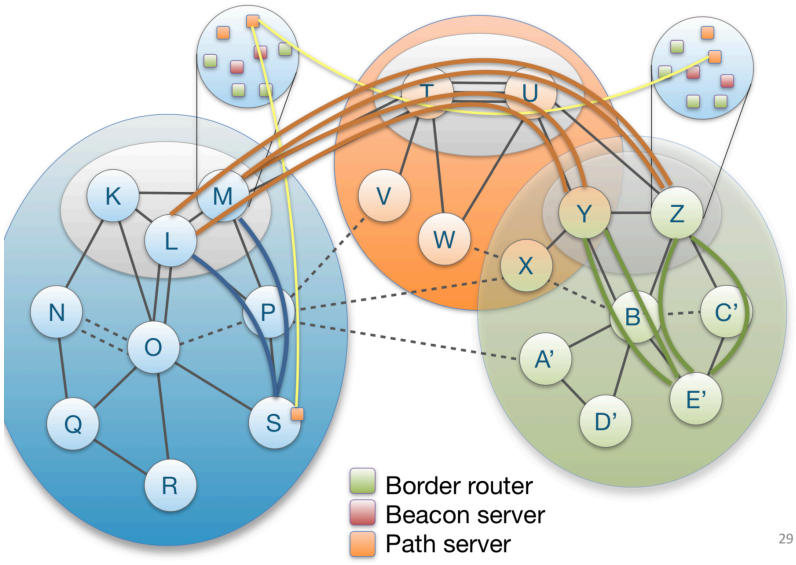
BGP / BGPsec have fundamental limitations that SCION tries to solve. There are frequent periods of unavailability when paths change, slow convergence, the protocol is susceptible to attacks / misconfigurations, there is poor path predictability / reproducibility, almost no path choice by end points and it uses very few trust roots (single points of failure).

SCION groups ASes as isolation domains (ISDs). An ISD core consists of ASes that manage the ISD ("Core ASes"). The control plane is organized hierarchically with an Inter-ISD control plane and an Intra-ISD control plane. Each ISD defines their TRC (Trust Root Configuration) which contains the root cryptographic keys to verify ISD operations.

Intra-ISD path exploration works by beaconing. Core ASes initiate the path-segment construction beacons (PCBs) which traverse the ISD as a flood to reach downstream ASes. Each AS receives multiple PCBs representing path segments to a core AS. There exist border routers / beacon servers. Border routers forward the beacons to the beacon servers, which coordinate the periodic resending of the beacons. Beacons use signatures.

For Inter-ISD path exploration, the core ASes exchange beacons. Path servers offer the lookup service. The core ASes operate the core path server infrastructure and each non-core AS runs local path servers that servers up-path segments to local clients and resolves / caches responses of remote AS lookups. An AS selects the path segments to announce as down-path segments (traffic from the core "downwards") and uploads them to the core path server.

The path lookup works as follows: A host contacts RAINS (similar to DNS) with a name and gets back the ISD / AS / local address. The host contacts the local path server with the ISD / AS and gets back the up-path, core-path and down-path segments. The host combines path segments (and therefore knows the whole path in advance) to obtain end-to-end paths that are added to packets. For different ISDs, the core path server may need to contact the core path server of the other ISD for the down-path segment (if it is not cached).



These paths mimic current internet paths (up-path = paths up to tier-1 ISP, core-path = tier-1 close to destination and down-path = tier-1 ISP to lower-tier ISPs).

Each AS assigns a unique integer identifier to each interface. These are used for internal routing protocols to find routes from ingress SCION border router to egress SCION border router.

SCION can be used for DDoS defense. There's end-system high-speed source authentication, multi-path communication enables circumventing congested areas, hidden paths prevents flooding of last-mile links and there's a global QoS system.

4.12 Virtual Circuits

In the virtual circuit model, there are three phases:

1. Connection establishment, circuit is set up (path is chosen, circuit information is stored in routers)
2. Data transfer, circuit is used (packets are forwarded along the path)
3. Connection teardown, the circuit is deleted

Packets only contain a short label to identify the circuit. These labels don't have a global meaning, they are only unique for a link. Each router has a forwarding table that is keyed by the circuit.

MPLS (Multi-Protocol Label Switching) is a virtual-circuit like technology that is widely used by ISPs. The ISP sets up circuits inside his backbone ahead of time. At ingress, the MPLS label is added to an IP packet and removed at egress.

Datagrams / virtual circuits have different strengths:

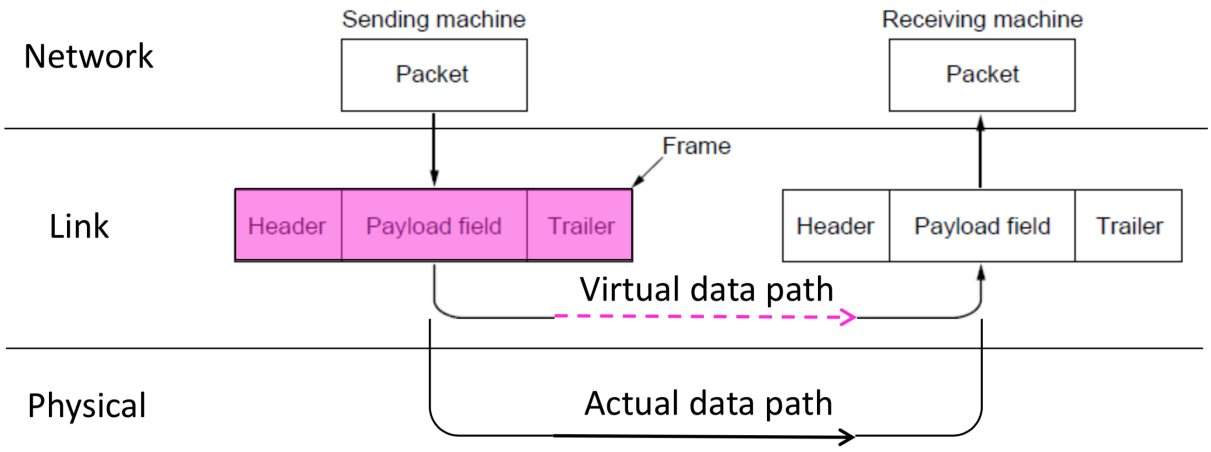
Issue	Datagrams	Virtual Circuits
Setup phase	Not needed	Required
Router state	Per destination	Per connection
Addresses	Packet carries full address	Packet carries short label
Routing	Per packet	Per circuit
Failures	Easier to mask	Difficult to mask
Quality of service	Difficult to add	Easier to add

4.12.1 MPLS

For packet forwarding, routers can directly forward IP packets, or one can build an internal network based on Multiprotocol Label Switching (MPLS). The second approach has the advantage that internal routers do not need to understand IP and have smaller routing tables / faster switching. It's possible to set up multiple routes between a pair of nodes (which enables traffic engineering) and allows ISPs to set up Virtual Private Networks (VPNs) by construction a virtual circuit between customer's networks. When using MPLS, ingress routers receive packets without MPLS labels and insert the appropriate label. Intermediate routers inspect the labels, modify them and forward the packet. Egress routers remove the label and forward the packet according to the layer-3 header. Forwarding equivalence classes (FEC) are groups of packets that take the same label-switched path through the MPLS network. To set up the labels, an IGP can be used (only works for distance-vector protocols) or a separate Label Distribution Protocol.

5 Link Layer

The link layer concerns how to transfer messages over one or more connected links. Messages are frames of limited size. The link layer adds a "virtual data path" between hosts:



5.1 Framing

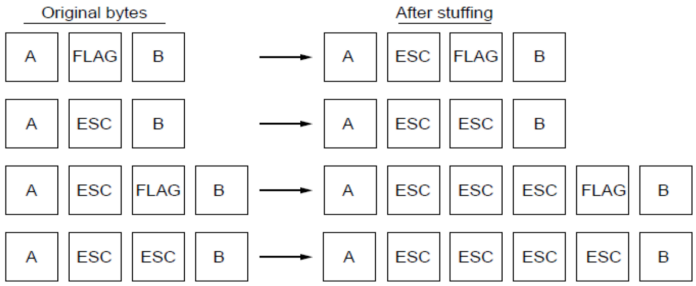
The physical layer gives a stream of bits that the link layer needs to interpret as a sequence of frames. There are different framing methods.

5.1.1 Byte Count

A very simple method is to start each frame with a length field. But it's very difficult to re-synchronize after a framing error.

5.1.2 Byte Stuffing

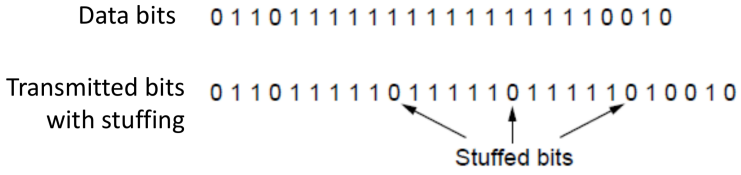
In this approach, there exists a special flag byte value that means start / end of frame. The flag inside the frame needs to get escaped with an escape code (but the escape code needs to get escaped as well):



Any unescaped FLAG is then the start / end of a frame.

5.1.3 Bit Stuffing

In this approach, six consecutive 1s is called a flag. On transmit, a 0 is inserted after five 1s and on receive, a 0 after five 1s is deleted.



5.2 Error Coding

Some bits will be received in error due to noise. The errors can be detected / corrected with codes (or the lost frames can get retransmitted). The approach is to add redundancy (check bits). The hamming distance is the number of bit flips needed to change $D + R_1$ to $D + R_2$. The hamming distance of a code is the minimum distance between any pair of codewords. For a code of hamming distance d+1, up to d errors will always be detected. For a code of hamming distance 2d+1, up to d errors can always be corrected by mapping to the closest codeword.

A very simple error detection is to use a parity bit. 1 check bit is added. It is the sum (modulo 2 or XOR) of the data bits. The code has a hamming distance of 2 and can therefore detect 1 error.

Another approach is to use checksum. The internet checksum works by arranging data in 16-bit words, putting zero in the checksum position and adding the words together. Any Carryover is added as well and the sum is negated:

Sending:

1. Arrange data in 16-bit words
2. Put zero in checksum position, add
3. Add any carryover back to get 16 bits
4. Negate (complement) to get sum

```
  0001
  f203
  f4f5
  f6f7
+(0000)
------
 2ddf0
   ↓
  ddf0
+    2
------
  ddf2
   ↓
  220d
```

Receiving / controlling the checksum works very similar:

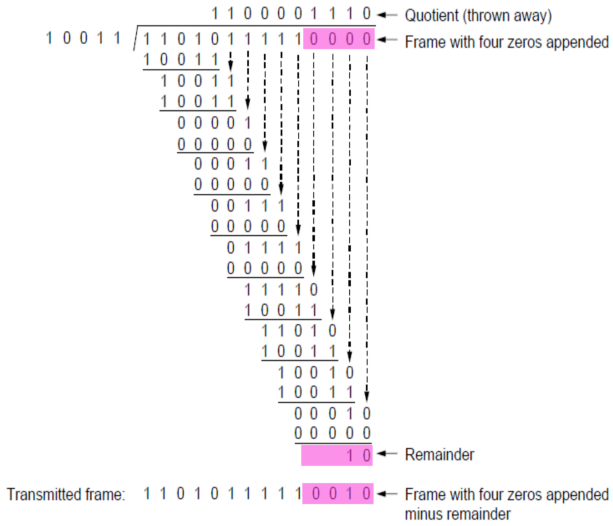
Receiving:

1. Arrange data in 16-bit words
2. Checksum will be non-zero, add
3. Add any carryover back to get 16 bits
4. Negate the result and check it is 0

```
  0001
  f203
  f4f5
  f6f7
+ 220d
------
 2fffd
   ↓
  fffd
+    2
------
  ffff
   ↓
  0000
```

This checksum has (in the worst case) as well a hamming code of 2, but the errors have to be 16-bit apart.

Cyclic Redundancy Check (CRC) gives an even stronger protection. Given n data bits, k check bits are generated such that the n+k bits are evenly divisible by a generator C. CRCs are based on mathematics of finite fields, in which "numbers" represent polynomials. For instance, 10011010 is $x^7 + x^4 + x^3 + x^1$. The send procedure is to extend the n data bits with k zeros, divide by the generator value C, keep the remainder and adjust the k check bits by the remainder.



The receive procedure is to divide and check for zero remainder. CRC-32 (where C consists of 32 bits) has a hamming distance of 4.

The hamming code gives a method for construction a code with a distance of 3. We have $n = 2^k - k - 1$, e.g. n=4 and k=3. Check bits is put in positions p that are powers of 2, starting with position 1. Check bit in position p is the parity sum of all bits where the (binary) index has a 1 in position p.

Example: data=0101, 3 check bits
- 7 bit code, check bit positions 1, 2, 4
- Check 1 covers positions 1, 3, 5, 7
  - Bit positions xx1
- Check 2 covers positions 2, 3, 6, 7
  - Bit positions x1x
- Check 4 covers positions 4, 5, 6, 7
  - Bit positions 1xx

$\overline{7}\ \overline{6}\ \overline{5}\ \overline{4}\ \overline{3}\ \overline{2}\ \overline{1}$

To decode, the check bits are recomputed (with the parity sum including the check bit) and arranged as a binary number. The value (syndrome) tells the error position, a value of zero means no error.

$\longrightarrow$ 1 1 1 0 0 1 0
7 6 5 4 3 2 1

p_1= 0+0+1+1 = 0, p_2= 1+0+1+1 = 1,
p_4= 0+1+1+1 = 1

Syndrome = 1 1 0, flip position 6
Data = 1 0 1 0 (correct after flip!)

Error correction is needed when errors are expected (a small number of errors that are correctable) or when there is no time for a retransmission. Error detection is more efficient when errors are not expected and when errors are large when they do occur.

Instead of correcting errors, they can also be detected and the frame retransmitted (Automatic Repeat reQuest, ARQ). ARQ is often used when errors are common or must be corrected (e.g. TCP / WiFi). The receiver automatically acknowledges correct frames with an ACK, the sender automatically resends after a timeout with no ACK.

5.3 Multiplexing

Multiplexing is the network word for sharing of a resource. There is time division multiplexing (TDM) where users take turn on a fixed schedule. There's also frequency division multiplexing (FDM) where different users are put on different frequency bands. TV / radio stations use FDM, GSM allocates calls using TDM within FDM.

Network traffic is bursty so TDM / FDM is inefficient. There are different multiple access protocols / medium access (MAC) protocols:

5.3.1 Randomized

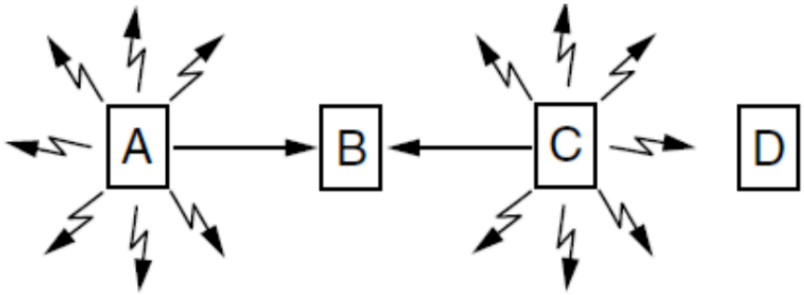
In this multiple access protocol, nodes randomize their resource access attempts. This protocol is good for low load situations. The basic idea is very simple. A node just sends when it has traffic, if there was a collision (no ACK received), a node waits a random time and resends.

5.3.2 CSMA/CD

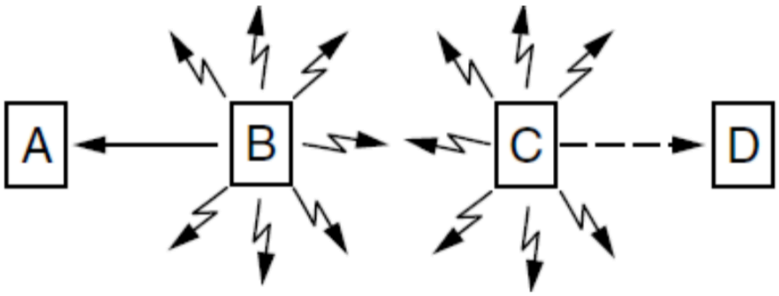
CSMA/CD stands for Carrier Sense Multiple Access with Collision Detection. In this approach, a node listens for activity before sending. Collisions are still possible because a node can listen and hear nothing when another node is sending because of delay. If a collision is detected while sending, a jam signal is sent so that the sender notices the collision. There is a minimum frame size (Ethernet 64 bytes) so that a node can't finish the transmission before he notices the collision / gets the jam signal. Binary exponential backoff is used. In this approach, a sender waits 0 or 1 frame times on the 1st collision, from 0 to 3 times on the second collision, etc… (the interval is always doubled).

5.3.3 WiFi

CSMA/CD isn't possible for WiFi because nodes may have different areas of coverage and nodes can't hear while sending. There is the problem of hidden terminals (A / C in the following example) that can't hear each other but still collide (at B in the example):

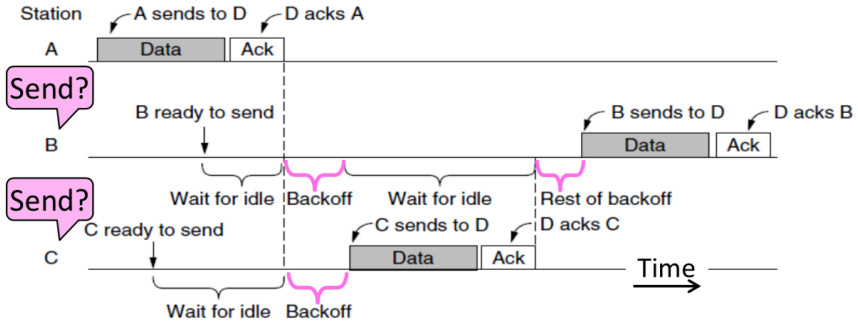


Exposed terminals can hear each other yet don't collide:



A solution is MACA: A sender node transmits an RTS (Request-To-Send with the frame length). The receiver replies with a CTS (Clear-To-Send with the frame length). The sender transmits the frame while nodes hearing the CTS stay silent. MACA solves the hidden terminal / exposed terminal problem.

802.11 uses CSMA/CA. Collisions are avoided by inserting small random gaps / backoffs:

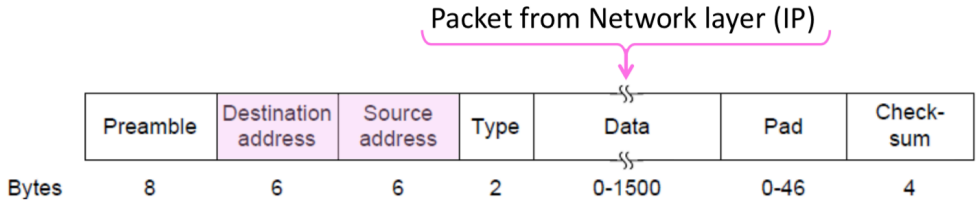


5.3.4 Contention-Free

Contention-Free Multiple Access is based on turns, not randomization. An order in which nodes get a chance to send (or pass) is defined. An example is token ring where nodes are arranged in a ring and a token rotates as "permission to send" to each node in a turn. This has the advantage that the overhead is fixed with no collisions and service is predictable. But the complexity is very high and the overload is high at low load.

5.4 Ethernet

Ethernet has addresses to identify the sender / receiver, CRC-32 for error detection, no ACKs or retransmission and the start of the frame is identified with a physical layer preamble:



Modern Ethernet is based on switches (and hubs, that simply broadcast frames), not multiple access. Hosts are wired to Ethernet with twisted pair. A switch uses the frame addresses to connect the input port to the right output port, multiple frames may be switched in parallel. A switch needs buffers so multiple inputs can send to one output.

Switches use a technique called backward learning. To fill the table, it looks at the source address of input frames. To forward, it sends to the correct port, or else broadcasts to all ports.

5.4.1 Spanning Tree

A topology may have redundancy which causes loops. This could cause forwarding loops. Switches collectively find a spanning tree (a subset of links that is a tree and reaches all switches) for the topology. Switches forward as normal but only on the spanning tree. A broadcast will go up to the root of the tree and down all the branches.

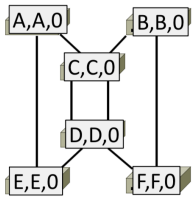
The spanning tree algorithm works like this:

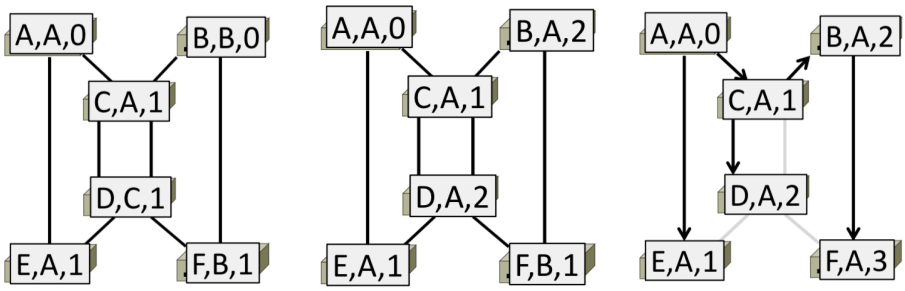
1. Elect a root node of the tree (switch with the lowest address)
2. Grow tree as shortest distances from the root (using lowest address to break distance ties)
3. Turn off ports for forwarding if they are not on the spanning tree.

Each switch initially believes it's the root of the tree and sends periodic updates to neighbors with its address, address of the root and the distance (in hops) to the root. Switches favors ports with shorter distances to the lowest root. An example run looks like this:

1st round, sending:
- A sends (A, A, 0) to say it is root
- B, C, D, E, and F do likewise

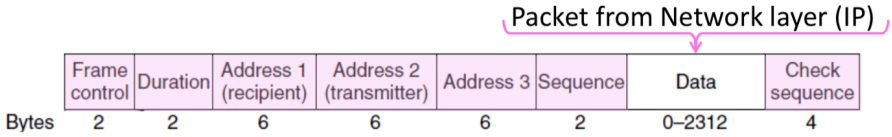
1st round, receiving:
- A still thinks is it (A, A, 0)
- B still thinks (B, B, 0)
- C updates to (C, A, 1)
- D updates to (D, C, 1)
- E updates to (E, A, 1)
- F updates to (F, B, 1)





5.5 802.11

An 802.11 frame looks like this:



Frames are ACKed and retransmitted with ARQ. There are three addresses due to access points and errors are detected with a 32-bit CRC.

6 Physical Layer

The physical layer concerns how signals are used to transfer message bits over a link. Wires etc. carry analog signal but we want to send digital bits.

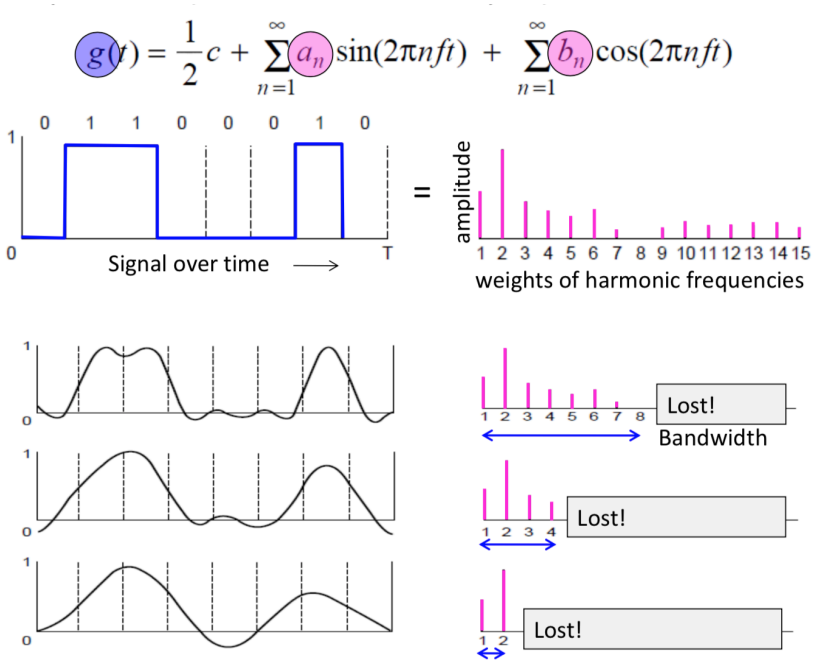
6.1 Properties of Media

A simple link model is that a link has a rate in bits/second and a delay / latency in seconds that is related to the length. The message latency is the transmission delay + the propagation delay. Media propagate signals that carry bits of information.

Twisted Pair wires are very common and used in LANs and telephone lines. Twists can reduce the radiated signal or reduce the effect of external interference signal. Coaxial cables are also common and provide better shielding for better performance. Fiber are long, thin, pure strands of glass and provide enormous bandwidth over long distances. There is multi-mode fiber (shorter links and cheaper) and single-mode (up to 100km). With wireless, a sender radiates a signal over a region (in many directions). Nearby signals interfere at a receiver; the use needs to be coordinated.

6.2 Simple Signal Propagation

A signal over time can be represented by its frequency components. Fewer frequencies (=less bandwidth) degrades the signal:



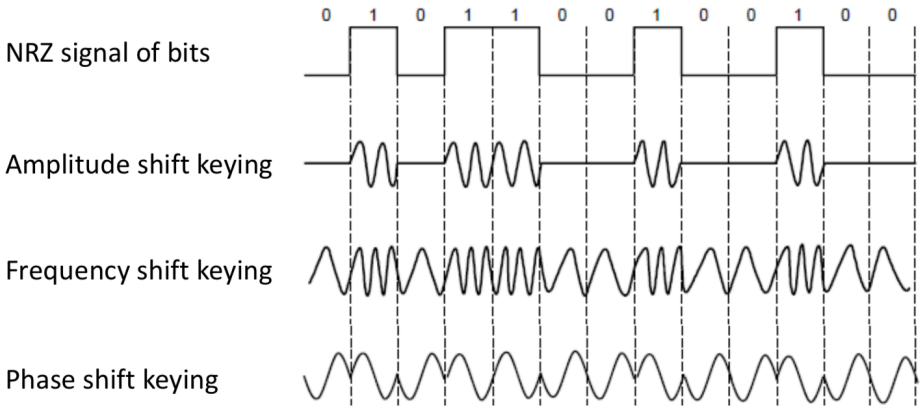
Frequencies above a cutoff are highly attenuated. Light propagates with very low loss in three very wide frequency bands.

6.3 Modulation Schemes

Modulation deals with how bits are represented. In NRZ (Non-Return to Zero), a high voltage (+V) represents a 1 and a low voltage (-V) a 0. More signal levels can be used, e.g. 4 levels which allows 2 bits per symbol.

To decode the bits, a receiver needs frequent signal transitions (otherwise he can't know for instance how many zeros a low voltage was). For instance, there's 4B/5B that maps every 4 data bits into 5 code bits without long runs of zeros. Or there's NRZI (Non-Return to Zero Inverted) which inverts the signal level on a 1 and keeps it on a 0.

These described techniques are baseband modulation where the signal is sent directly on a wire. They do not propagate well on fiber / wireless. Passband modulation carries a signal by modulating a carrier (a signal oscillating at a desired frequency). The carrier can be modulated by changing the amplitude, frequency or phase.



6.4 Fundamental Limits

There's the Nyquist limit and the Shannon capacity that deal with the question how rapidly it is possible to send information over a link. The Shannon capacity says that how many levels one can distinguish depends on S/N (the signal-to-noise ration). For wires / fibers, the SNR can be engineered for the data rate but for wireless, the bandwidth is given but the SNR varies greatly.

7 Algorithms

The maximum-flow problem in networks deals with how much traffic a network can carry. The min-cut problem deals with the reliability of a network (how many edges do I have to remove to break it?). There exists the Max-flow / min-cut duality

7.1 Linear Programming

A linear program contains variables (flows on edges, distance from start, …), an objective (maximize flow s to t, minimize distance s to t, …) and constraints (flow on edge <= capacity on edge, …). The variables must be real numbers, the objective must be maximize / minimize f(x1, …, xn) where f is a linear function. The constraints are of the form: linear combination of variables <= 0.

Many problems can be expressed as linear programs (but specialized algorithms will often be better). For instance, for max flow one can define f as the variable of the flow from s to t, f_{uv} as the variable of the flow from u to v, the objective as maximize f and the constraints as:

Capacity: $f_{uv} \leq c_{uv}$

Conservation: in-flow = out-flow

$\sum_{w \to u} f_{wu} = \sum_{u \to v} f_{uv} \quad \forall\ u - \{s,t\}$

$\sum_{s \to v} f_{sv} = f$

$\sum_{v \to t} f_{vt} = f$

$f_{uv} \geq 0$

For shortest path, one can define as a variable x_{uv} that indicates if the edge u → v is on the shortest path. The objective is to minimize the sum of $x_{uv} w_{uv}$ over edges u → v. The constraints are:

Path s→t is connected:

$\sum_{u \to v} x_{uv} - \sum_{v \to w} x_{vw} = 0 \quad \forall v - \{s, t\}$

$= 1 \quad [v = t]$

$= -1 \quad [v = s]$

Also: $x_{uv} \in \{0, 1\}$?

$x_{uv} \in [0, 1]$ is enough!

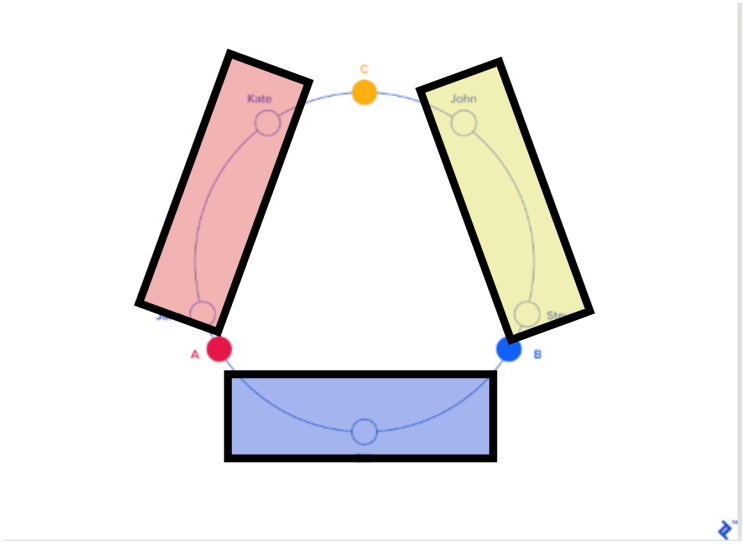
Lucky in this case.

An alternative formulation is to define d_u as s → u distance and for each u → v, $d_v \leq d_u + w_{uv}$. Then, d_t has to get maximized!

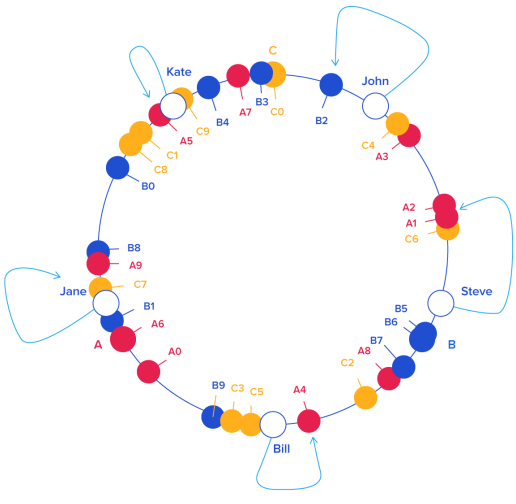
Integer linear programs are linear programs where the variables are integers. The general class of ILPs is NP-hard (in contrast to linear problems that are solvable in polynomial time).

7.2 Randomness

A common problem is to distribute requests to different servers. A simple approach is to use round-robin (sequential distribution) which can be problematic if the requests follow a pattern (every n-th is a very hard task and we have n servers). Another approach is to send it to the least loaded server which incurs the overhead of querying / tracking. A server can also be picked uniformly at random. To improve the expected maximum load on any server, two (or k) can be picked at random, their load can be queried and the less loaded one can be chosen. Sometimes completely random selectin is harmful (persistent session). Then, one can calculate a hash over the user-id. Consistent hashing can be used to prevent reassignments of sessions. In this approach, one hashes both machine and data and each machine deals with the data points in the clockwise direction, before the next machine.



To improve load balancing, virtual nodes can be introduced where the load is better distributed (with a small number of machines):



7.2.1 Bloom filters

Bloom filters are a form of yes / no hash tables, where collisions are ignored. They are used for membership testing. k hash functions are used and “yes” is the answer iff the entry in all hashed places is “yes”. Counting bloom filters allow a delete without recreating the filter afresh. On insert, each place is incremented by 1 and decremented on delete. On lookup, “yes” is the answer iff every entry is non-zero.