This document is an exam summary that follows the script of Communication Networks lecture at ETH Zurich. The contribution to this is a short summary that includes the most important concepts, formulas, and algorithms. This summary was created during the spring semester 2018. Due to updates to the syllabus content, some material may no longer be relevant for future versions of the lecture. This work is published as CC BY-NC-SA.

I do not guarantee correctness or completeness, nor is this document endorsed by the lecturers. Feel free to point out any erratas.

Communication Networks

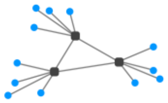
Yannick Merkli

FS 2018

1 Overview – Internet (Part 1)

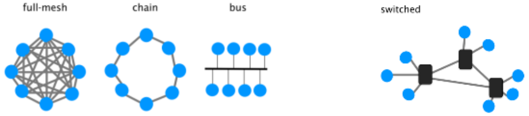
1.1 Different kinds of Networks in the Internet

Basic components of a network



- ● End-system
- ■ Switch / router
- — Link

Network designs



Requirements for networks

- Failure tolerable
- Feasible & cost-effective
- Adequate per-node capacity

ADSL (Asymmetric Digital Subscriber Line)

- Transmit digital data over telephone lines (Swisscom)
- 3 channels: downstream, upstream, 2-way phone channel
- Downstream approx. 10 faster than upstream (asymmetric…)

CATV (Cable Access Television)

- Transmit digital data over coaxial TV cable (Cablecom)
- 2 channels: downstream, upstream
- Downstream approx. 10 faster than upstream
- Shared between households

Ethernet (inside companies or universities)

- Twisted copper pair
- Symmetric (same speed in both directions)

Others

- Cellular, Satellite, FTTH, Fibers, Infiniband, …

1.2 Sharing

3 requirements for network topology:

1) Should tolerate failures (several paths between each source and dest.)
2) Provide adequate per-node capacity (# of links should not be too small)
3) Posses enough sharing to be feasible & cost-effective (# of links should not be too high)

Two approaches:

On demand	Reservation
Send data when you need to	Reserve the bandwidth you need in advance
Sharing at the packet level	Sharing at the flow level
	Good when P/A is small (~3), bad when P/A is big (>100)
Implement: packet-switching	Implement: circuit-switching

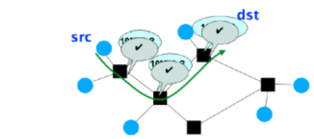
Utilization: $\boxed{U = A/P}$ (A = Average rate, P = Peak rate)

- Example: U = 10% if P = 100 Mbps and A = 10 Mbps
- With on-demand one achieves higher levels of utilization
- Reservation must serve P

1.2.1 Circuit-Switching

- Reserve a circuit path from source to destination
- Time-based or frequency-based multiplexing (to handle multiple users)
- Circuit needs to be established and teared down (takes time)

Resource Reservation Protocol



(1) src sends a reservation request for 10Mbps to dst
(2) Switches "establish a circuit"
(3) src starts sending data
(4) src sends a "teardown circuit" message

Advantages	Disadvantages
Predictable performance	Inefficient if traffic is bursty/ short
Simple & fast switching once circuit is established	Complex circuit setup/teardown which adds delay to transfer
	Requires new circuit upon failure (doesn't route around trouble)

1.2.2 Packet-Switching

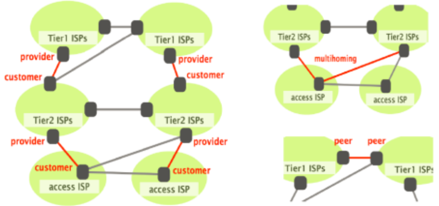
Advantages	Disadvantages
Efficient use of resources	Unpredictable performance
simpler to implement than circuit switching	Requires buffer management and congestion control (to absorb transient overload)
Routes around trouble	

Packet switching beats circuit switching with respect to resiliency and Efficiency. That's why in today's internet, we use **packet-switching.**

1.3 Hierarchical structure of the internet

Tier-1: international	Have no providers	~12
Tier-2: national	Provide transit to Tier-3s, have at least one provider	1000s
Tier-3: local	Do not provide transit services, have at least one provider	85-90%

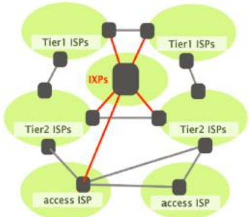
➔ Total ~50'000 networks



- **ISP (Internet Service Provider):** Swisscom, Cablecom, AT&T, …
- The different tiers have peer-to-peer connections to communicate with each other.
- Communication from a Tier-2 ISP to another Tier-2 ISP always goes over Tier-1 ISP or a direct peer-to-peer connection, but never over a Tier-3 ISP.

Internet eXchange Points (IXPs):

Interconnecting each network to its neighbors one-by-one is not cost Effective:



- Physical cost: of provisioning or renting physical links
- Bandwidth cost: a lot of links are not necessarily fully utilized
- Human cost: to manage each connection individually

IXPs solve these problems by letting many networks connect in one location!

1.4 Internet layers

Protocol: Convention which controls the links, the communication and the data transfer between two nodes in a network.

➔ Basic protocol functions include: send(), receive(), deliver()

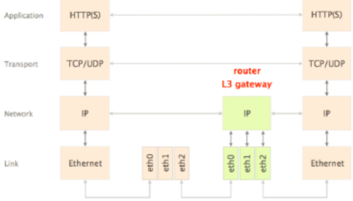
In practice, there exist a lot of network protocols (-> **Modularity!**)

Internet Layers:

To provide some structure to the design of network protocols, the protocols are organized in layers (as well as the hardware/software that implements these protocols).

Nr.	Layer	Service sends	Protocols
L5	Application	Network access (exchange **messages**)	HTTP,SMTP FTP, SIP,…
L4	Transport	End-to-end delivery (reliable or not) (transport **segments** between end systems)	TCP, UDP, SCTP
L3	Network	Global best effort delivery (transport **packets/ datagrams** in the network)	IP
L2	Link	Local best effort delivery (moves **frames** across a link)	Ethernet, Wifi, DSL, LTE
L1	Physical	Physical transfer of bits (moves **bits** across a medium)	Cooper cable, wireless,..

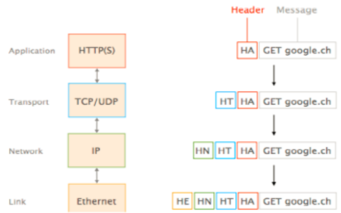
The official layer model called OSI model (Open Systems Intercon-nection model) features 7 layers. The bottom 4 layers are identical to this layer model. On top of these the session layer (L5), the presentation layer (L6) and the application layer (L7) are added.



Layers in different network Devices:

- **Host:** Application, Transport, Network, Link, Physical
- **Router** (L3 gateway): Network, Link, Physical
- **Switch** (L2 gateway): Link, Physical

Headers:
Each layer adds a new header to the sent message. This is the reason for the different names of the messages on the different layers (message, segment, packet, frame).



1.5 Characterization of the Internet

A network connection is characterized by its:

1.5.1 Delay

How long does the packet need? A packet gets delayed at each node of the transport.

$$\boxed{t_{delay} = t_{transmission} + t_{propagation} + t_{processing} + t_{queuing}}$$

1.5.1.1 Transmission delay

The amount of time required to push all of the bits onto the link.

$$\boxed{t_{transmission}[s] = \frac{packet\ size\ [bit]}{link\ bandwidth\ [bit/s]}}$$

1.5.1.2 Propagation delay

The amount of time required for a bit to travel to the end of the link.

$$\boxed{t_{propagation}[s] = \frac{link\ length\ [m]}{propagation\ speed\ [m/s]}}$$

Different transmission characteristics imply different tradeoffs in terms of which delay dominates:

$10^7 * 100B$ packtes	1Gbps link	Transmission delay dominates
$1 * 100B$ packtes	1Gbps link	Propagation delay dominates
$1 * 100B$ packtes	1Mbps link	Both matter

General rule:

- Large packets → transmission delay dominates
- Small packets → propagation delay dominates

→ in the internet, we can't know in advance which one matters!!

1.5.1.3 Queuing delay

The amount of time a packet waits (in a buffer) to be transmitted on a link. It is characterized with statistical measures - hard to evaluate.
It depends on the traffic pattern:

- arrival rate at the queue
- transmission rate of the outgoing link
- traffic burstiness

Average packet arrival rate	a	[packet/s]
Transmission rate of outgoing link	R	[bit/s]
Fixed packet length	L	[bit]
Average bit arrival rate	La	[bit/s]
Traffic intensity	La/R	[1]

- **Traffic intensity >1**: Queue increases without bound
- **Traffic intensity ≤1**: Queuing delay depends on burst size

1.5.2 Loss

Queuing delay upper bound:
In practice, queues are not infinite:

- There is an upper bound on queuing delay: $N * L/R$
  (N = queue size, L = packet size, R = transmission rate)

If queue is persistently overloaded, it will eventually drop packets (loss)

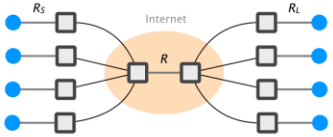
1.5.3 Throughput

$$\boxed{Average\ throughput\ [bit/s] = \frac{data\ size\ [bits]}{transfer\ time\ [s]}}$$

If a link has different transmission rates, one has to consider the bottleneck link (link with smallest troughput)

$$\boxed{Average\ throughput = \min(R_1, R_2, R_3, \ldots)}$$

Example:



if 4*min(R_S,R_L) > R the bottleneck is now in the core, providing each download R/4 of throughput

Bandwidth: $\boxed{bandwidth = \frac{amount\ of\ data}{delay}}$

2 Overview – Forwarding & Routing (Part 2)

2.1 Basics

	Forwarding	Routing
Goal	Directing packet to an outgoing link	Computing the paths packets will follow
Scope	Local	Network-wide
Implementation	Hardware (usually)	Software (always)
Timescale	Nanoseconds	Milliseconds

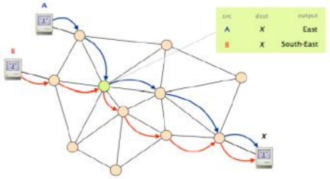
IP Packets [Header: (source IP, dest. IP) | payload]

- Payload: Actual message to transmit
- Routers forward IP packets hop-by-hop towards their destination. Upon packet reception, routers locally look up their forwarding table to know where to send it next.

Forwarding decision necessarily depends on the destination, but can also depend on other criteria:

- destination: mandatory
- source: requires n^2 state
- input port: traffic engineering
- other header

With **source- & destination based routing**, paths from different sources can differ!
With **destination-based routing**, paths from different source coincide once they overlap.



Once path to destination meet, they will never split. Set of paths to the destination produce a spanning-tree rooted at the destination:

- cover every router exactly once
- only one outgoing arrow at each router

Spanning-tree (Example for node X):

- covers every router exactly once
- Every router only has one outgoing arrow

2.1.1 Router/ Forwarding Table

Router: Device (L3 gateway) that is virtually split into two planes:

- Control plane: where the routing takes place, routing computes and populates the forwarding tables
- Data plane: uses forwarding table to transmit data

Forwarding table:

- Stored in a router or switch
- Content: which user is connected to which router output
- Used upon packet reception to know where to send it next

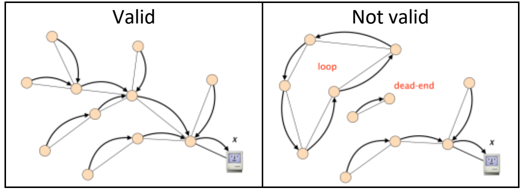
2.2 Verifying that a routing state is valid

Theorem: (sufficient and necessary condition):
A global forwarding state is valid if and only if

- There are no dead ends
- There are no loops

Algorithm for one destination:

1) Mark all outgoing ports with an arrow (based on forwarding table)
2) Eliminate all links with no arrow
3) State is valid iff the remaining graph is a spanning tree



2.3 How to compute a valid routing state

2.3.1 Use tree-like topologies

- Build a spanning tree for given topology and ignore all other links. Then flood the packets everywhere. But: Always flooding is wasteful
- More efficient if nodes remember where packets came from
  - If a flood packet from node A entered switch X on port 4, switch X can use port 4 to reach node A from now on.
  - Each switch decides whether to flood or not depending on who has communicated before.

Advantages	Disadvantages
Plug-and-play configuration free	Mandate a spanning-tree eliminates many links from topology
Automatically adapts to moving host	Slow to react to failures

In practice, operators tend to dislike spanning trees.

2.3.2 Rely on Global Network view

If each router knows the entire graph, one can find the shortest paths to any given destination with Dijkstra's algorithm.

Dijkstra's algorithm (used by OSPF) $\mathcal{O}(n^2)$

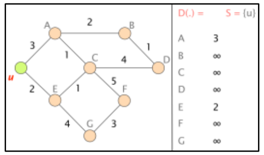
```
Initialization:
$S = \{u\}$
for all nodes $v$:
    if ($v$ is adjacent to $u$):
        $D(v) = c(u,v)$
    else:
        $D(v) = \infty$
```

```
Loop
while not all nodes in $S$:
    add $w$ with smallest $D(w)$ to $S$
    update $D(v)$ for all adjacent $v$:
        $D(v) = \min\{D(v), D(w) + c(w,v)\}$
```

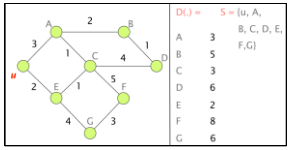
- u = node which runs the algorithm
- v, w = nodes of the graph (without u)
- S = set witch processed nodes
- $D(v)$ = smallest distance currently known by u to reach v
- $c(u,v)$ = cost / weight of the link between u and v

Example:

After initialization:



After Loop:



With the shortest paths u can directly compute its forwarding table

2.3.3 Rely on distributed computation

Distance-Vector (DV) Routing algorithm (Bellman-Ford)

1) Let $D_x(y)$ be the cost of the least-cost path known by x to reach y
2) Each node bundles these distances into one message (called a vector) that it repeatedly sends to all its neighbors (until convergence)
3) Each node updates its distances based on neighbors' vectors:

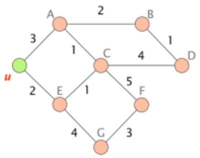
$$D_x(y) = \min_v\{c(x,v) + D_v(y)\} \quad \forall\ neighbors\ v$$

$c(x,v)$ = cost/ weight of the link between x and v

4) Over time $D_x(y)$ converges to the shortest-path distances and next-hops

```
Initialization:
    for all destinations y in N:
        $D_x(y) = c(x,y)$ //if y is not a neighbor: $c(x,y) = \infty$
    for each neighbor w:
        $D_w(y) = ?$ for all destinations y in N
    for each neighbor w:
        send distance-vector $D_x = [D_x: y \text{ in } N]$ to w
Loop:
    wait (until a link cost change or I receive a distance-vector)
    for each y in N:
        $D_x = \min_v[D_x: c(x,v) + D_v(y)]$
    If $D_x(y)$ changed for any destination y:
        Send distance vector $D_x = [D_x: y \text{ in } N]$ to all neighbors
```

Example: Shortest path from u to D (unfold the recursion and start at the end)

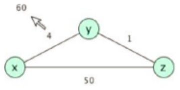


$d_B(D) = 1$
$d_C(D) = 4$
$d_A(D) = \min\{2 + d_B(D), 1 + d_C(D)\} = \min\{3, 5\} = 3$
$d_E(D) = \min\{1 + d_C(D), 4 + d_G(D)\} = = \min\{5, 13\} = 5$
$d_u(D) = \min\{3 + d_A(D), 2 + d_E(D)\} = \min\{6, 7\} = 6$

→ To reach node D from node u, the shortest path has distance 6 and the next-hop is node A.

Count-to-infinity problem:

Can happen when a link cost increases.



Initial distance vectors:
$\boldsymbol{d_x} = (0,4,5)$
$\boldsymbol{d_y} = (4,0,1)$
$\boldsymbol{d_z} = (5,1,0)$

Example:
y updates distance vector to (5+1,0,1) and sends it to neighbors.
z updates distance vector to (6+1,1,0) and sends it to its neighbors. d_y updates to (7 + 1,0,1). This pattern continues until z finds that its least cost path to x is via the direct link (for 44 iterations).

Poisoned reverse:

If z routes through y to get to x, then z will advertise to y that its distance to x is infinity. → used to avoid count-to-infinity problem between two neighboring nodes (does not work for loops involving three or more nodes, here the problem can be mitigated by setting a small infinity, e.g. 16)

3 Overview – Reliable Transport (Part 4)

Packets can get lost/ corrupted/ reordered/ delayed/ duplicated!

- In the internet reliability is ensured by the transport layer
- Possible errors caused by lower layers: Loss, delay, corruption, reordering or duplication of packets
- Four goals:
    - Correctness (ensure data delivered correctly)
    - Timeliness (minimize transfer time)
    - Efficiency (optimal use of bandwidth)
    - Fairness (play fair with other users)

Definition: A reliable transport design is correct if:

- a packet is always retransmitted if the previous packet was lost or corrupted
- a packet may be retransmitted at other times

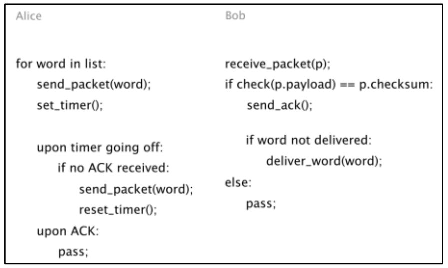
3.1 Designing a reliable transport protocol

3.1.1 Stop-and-wait protocol

Send one packet at a time

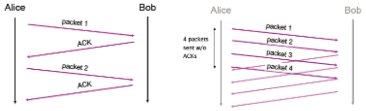
- Add header with sequence number and checksum to packet
- Receiver sends ACK (acknowledgement) for each received packet
- Sender sets timer for each transmitted packet and retransmits a packet if no ACK was received when timer goes off

Example:



```
Alice                                   Bob


for word in list:                       receive_packet(p);
    send_packet(word);                  if check(p.payload) == p.checksum:
    set_timer();                            send_ack();

                                            if word not delivered:
    upon timer going off:                       deliver_word(word);
        if no ACK received:             else:
            send_packet(word);              pass;
            reset_timer();
    upon ACK:
        pass;
```

Stop-and-wait protocols are limited by the propagation speed (c := speed of light) e.g. round trip east-west coast: 30ms. -> Solution: **Pipelining**



L := packet size, U := sender utilization, R := transmission rate

$$d_{Trans} = L/R$$

$$U_{Sender} = \frac{d_{Trans}}{RTT + D_{Trans}}$$

Send multiple packets

- Add sequence numbers to each packet
- Add buffers to the sender and receiver:

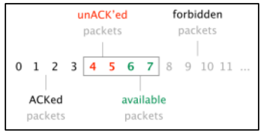
Sender	Store packets sent & not acknowledged in buffer
Receiver	Store out-of-sequence packets received in buffer

3.1.2 Sliding window

- Mechanism for flow control, needed to make sure sender does not send at a faster rate than the receiver can process.
- Sending window and receiving window
    - Sender keeps a list of the sequence # it can send: **sending window**
    - Receiver keeps a list of the acceptable sequence #: **receiving window**
- Sender and receiver negotiate the window size:

$$sending\ window \leq receiving\ window$$

- Example: (window of 4 packets)



3.1.3 Receiver Feedback

Individual ACKs

Advantages	Disadvantages
Fate of each packet known	Loss of ACK requires retransmission → Unnecessary retransmissions!
Simple window algorithm	
Not sensitive to reordering	

Cumulative ACKs

Whenever a packet arrives, the receiver sends back an ACK with the last consecutive received packet #

Advantages	Disadvantages
Recover from lost ACKs	Confused by reordering
	Incomplete information: which packets have arrived? → Unnecessary retransmissions!

Full information feedback

Every ACK contains: highest cumulative ACK & any additional packets

Advantages	Disadvantages
Complete Information	Large overhead possible -> limited efficiency

3.1.4 Fairness

When n entities are using our transport mechanism, we want a fair allocation of the available bandwidth.

- Avoid starvation
- Try to reach approximately equal flow per user

Max-min fair allocation (maximize the minimal demands)

1) Start with all flows at rate 0
2) Increase the flows until there is a new bottleneck
3) Hold the bottlenecked flows at the reached rate
4) Go to step 2) for the remaining flows

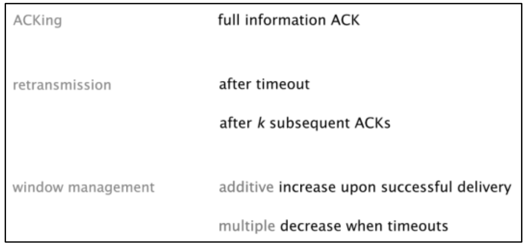
➔ Can be approximated by slowly increasing the window size until a loss is detected

3.1.5 Dealing with possible errors

- Loss
  - Using timers
  - Relying on ACKs
    - Individual ACKs: implicit
    - Cumulative ACKs: only partially possible
    - Full information feedback: explicit
- Delay
  - Can create unnecessary timeouts
- Corruption
  - Rely on checksum
  - Treat corrupted packets as loss
- Reordering (depends on ACKing mechanism)
  - Individual ACKs: no problem
  - Cumulative ACKs: needs correct order to work
  - Full information feedback: no problem
- Duplication (depends on ACKing mechanism)
  - Individual ACKs: no problem
  - Cumulative ACKs: problematic
  - Full information feedback: no problem

3.1.6 Example

Correct, timely, efficient and fair transport protocol (TCP)



ACKing	full information ACK
retransmission	after timeout
	after k subsequent ACKs
window management	additive increase upon successful delivery
	multiple decrease when timeouts

3.2 Go-Back-N (GBN)

Sliding-window protocol: sender is allowed to transmit multiple packets (when available) without waiting for an ACK but is constrained to a maximum of N unacknowledged packets in the pipeline.

- Sender:
  - Has a window of size N
  - Reception of an ACK with sequence number x indicates that all packets up to and including x were delivered (**cumulative ACK**)
  - Uses a single timer to detect loss, reset at each new ACK.
  - If timeout occurs, resend all packets that have been sent but not yet ACK'd (-> Sender has to go back N steps at max)
- Receiver (simple, no buffer needed):
  - If packet x was received after packet $x-1$, send ACK (the ACK points to the next expected packet, seqno $x+1$)
  - Else discard packet and resend ACK for the most recently received in-order packet
  - Timer is restarted upon receipt of any ACK

3.3 Selective Repeat (SR)

- Avoids unnecessary retransmissions
- Sender & Receiver both have a window size of N

Events and actions at sender:

- Data received from above (i.e. packet to send)
  - check next available sequence number for packet
  - if within sender's window, put data into packet and send
  - else, buffer data or return it to upper layer for later transmission
- Time-out
  - each packet must have its own logical timer, since only single packet will be transmitted on time-out
- ACK received
  - mark packet as received if in window
  - if sequence number equal to window base, move window forward to the unacknowledged packet with the smallest sequence number
  - if window moves and there are now untransmitted packets that fall into it, transmit those packets

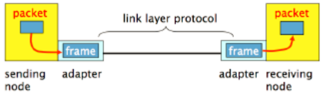
Events and actions at receiver:

- Packet with sequence number in receiver window is received
  - send selective ACK to sender
  - if not previously received, buffer packet
  - if sequence number equal to base of receiver window, deliver it and the subsequent previously buffered and consecutively numbered packets to upper layer and move receiver window forward by number of packets delivered to the upper layer
- Packet with sequence number smaller than the base of the receiver window is received
  - generate and send ACK even though packet has been previously acknowledged
- Otherwise
  - ignore packet

4 The Link Layer

What is a link?

Network adapters communicate together through the medium



Sender	Receiver
Encapsulates packets in a frame	Look for errors, flow control, …
Add error correction, flow control, …	Extracts packet and passes it to the network layer

encoding:	representing the 0s and 1s
framing:	encapsulate packet into a frame adding header and trailer
error detection:	detects errors with checksum
error correction:	optionally correct errors
flow control:	pace sending and receiving node

4.1 How do we identify link adapters?

4.1.1 MAC-Addresses

- MAC = Medium Access Control
- MAC addresses
  - identify the sender and receiver adapters
  - are uniquely assigned (hard-coded into each adapter)
  - use **48 bits**, e.g. 34:36:3b:d2:8a:86 (hexadecimal)
    - First 24 bits: vendor (assigned by IEEE) e.g. 34:36:3b = Apple Inc.
    - Second 24 bits: adapter specific (assigned by vendor) e.g. d2:8a:86 = Laurent's MacBook
- **Broadcast address:** ff:ff:ff:ff:ff:ff (all bits set to 1)

➔ Enables to send frame to all adapters on a link

- Two different adapter modes:
  - Default mode: decapsulate only frames addressed to a local MAC address or the broadcast address
  - Promiscuous mode: decapsualte everything

4.1.2 Why don't we simply use IP addresses

- Links can support any protocol (not just IP) different addresses on different kind of links
- Adapters may move to different locations -> cannot assign static IP address, it has to change
- Adapters must be identified during bootstrap? -> need to talk to an adapter to give it an IP address

4.2 Bootstrapping an adapter

Two problems need to be solved when you bootstrap an adapter:

1) Who am I? How do I acquire an IP address?

➔ MAC-to-IP binding: **DHCP**

2) Who are you? Given an IP address reachable on a link, how do I find out what MAC to use?

➔ IP-to-MAC binding: **ARP**

4.2.1 Dynamic Host Configuration Protocol (DHCP)

How does an adapter get an IP address?

1) Host with no IP address sends IP request to everyone on the link:

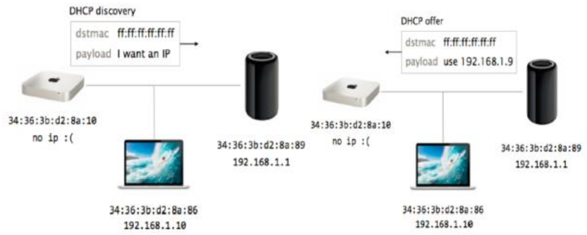
[src MAC = client_MAC, dst MAC = broadcast]	
[src IP = 0.0.0.0, dst IP = broadcast]	[payload: DHCP discovery]

2) If a DHCP server is connected to the same link, it broadcasts an IP address and the source MAC address

[src MAC = DHCP_server_MAC, dst MAC = broadcast]	
[src IP = DHCP_server_IP, dst IP = broadcast]	[payload: DHCP offer for *IP*]

3) Host receives IP address and uses it from now on.

Note: DHCP offer is in broadcast or unicast (both correct)



4.2.2 Address Resolution Protocol (ARP)

Given an IP address reachable on a link, how do I find out what MAC address to use?

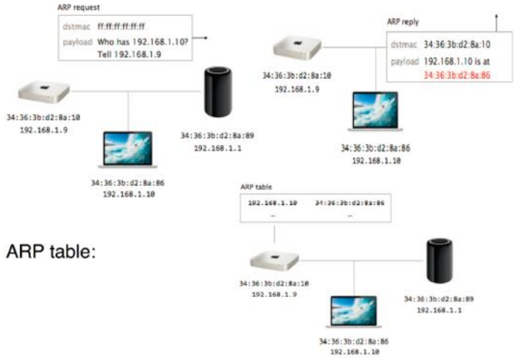
1) **ARP request**: Device A broadcasts "Who has 192.168.1.2? Tell 192.168.1.1 at MAC address 34:36:3b:d2:8a:10"

[src MAC = client_MAC, dst MAC = broadcast]	[payload: Who has…? Tell …]

2) **ARP reply**: Device B updates its ARP table with the information of device A and answers directly to device A: "192.168.1.2 is at 34:36:3b:d2:8a:86"

[src MAC = last_hop_MAC, dst MAC = client_MAC]	[payload: *IP* is at *MAC*]

3) Device A stores the received information in its ARP table and can communicate with device B from now on



Example:

SRC MAC address	DST MAC address	Message type	Message content
44:36:3b:12:ba:12	ff:ff:ff:ff:ff:ff	DHCP discovery	I need an IP address
34:36:3b:d2:8a:89	44:36:3b:12:ba:12	DHCP offer	use 192.168.1.37
44:36:3b:12:ba:12	ff:ff:ff:ff:ff:ff	ARP request	Who has 192.168.1.35 Tell 192.168.1.37
34:36:3b:d2:8a:10	44:36:3b:12:ba:12	ARP reply	192.168.1.35 is at 34:36:3b:d2:8a:10

4.3 Multi-Access: How do we share a network medium?

How can two or more hosts communicate at the same time over a shared medium?

Three different techniques:

- Divide the channel into pieces (time or frequency)
- Take turns (e.g. token passing)
- Random access (allow collisions, detect them, recover)

4.3.1 CSMA (Carrier Sense Multiple Access)

Carrier-sense	*Listen* before speaking, don't interrupt
Collision detection	*Stop* if someone else starts talking ensure everyone is aware of the collision
Randomness	*Don't* talk again right away

4.3.2 CSMA/CD

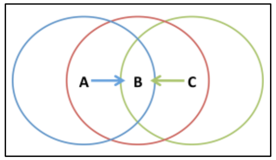
- No synchronization or feedback required
- Continue sensing the channel while transmitting
- If someone else starts transmitting, stop own transmission immediately
- To ensure that nobody started transmitting, a message needs to last at least for RTT (RTT = 2 * "max. latency between two hosts")
- Upon collision: Start retransmission after a random time
- CSMA/CD imposes limits on the network length:
  For this reason, Ethernet imposes a minimum packet size (512 bits)

4.3.3 Collision Detection

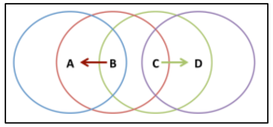
Wired network	Compare transmitted with received signal
Wireless network	- reception shuts off while transmitting - broadcast isnt perfect (limited range -> only local detection) - leads to use of Collision AvoidanceU (instead of detection)

4.3.4 Problems with CSMA/ CD and wireless transmission

- Sensing not possible while transmitting, because incoming signal is extremely weak compared to the transmitted signal
- Hidden Terminal Problem
  A and C can't see each other, both send to B simultaneously



- Exposed Terminal Problem
  C wants to send to D, listens to the channel and falsely assumes that it cannot (because B is already sending to A)



Solution: Use CSMA/CA instead

4.3.5 CSMA/CA (extra control messages)

- Relies on extra control messages

Control message	Request to send (RTS) A and C would ask for the right to transmit
	Clear to send (CTS) B would only give it to one of them
acknowledgments	Confirm correct reception If no ACK, sender assumes collision

- Upon collision: Start retransmission after a random time

4.4 Ethernet

Properties

- Dominant wired LAN technology
- Unreliable:
  - Receiving adapter sends no acknowledgements
  - Packets passed to the network layer can contain gaps which can be filled by the transport protocol (TCP)
- Connectionless:
  - No handshaking required between sender and receiver

Traditional Ethernet

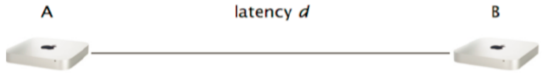
- 1 wired between 2 hosts
- CSMA/CD
- Minimum packet size of 512 bits = 64 bytes

$$Network\ length\ [m] = \frac{min\_frame\_size\ * speed\ of\ light}{2 * bandwidth}$$

$$Network\ length = 768m\ for\ 100Mb/s$$

Modern Ethernet

- Full duplex communication
  - 2 wires between 2 host
  - One wire used for transmission, the other for reception
  - ➔ No collisions possible!
- CSMA/CD is only needed for half-duplex communications
- 64 byte restriction not needed anymore, however IEEE chose to keep it
- Multiple Access Protocols are still important for Wireless

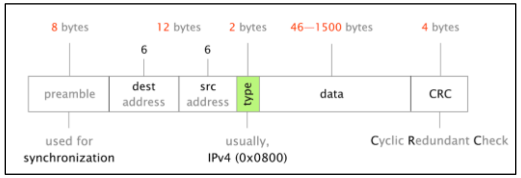


Suppose: - A sends a packet at time t
- B sees an idle just before $t + d$ and sends a packet

A sender can detect a collision only after $t + 2d$

=> Minimum packet-size = 512bits

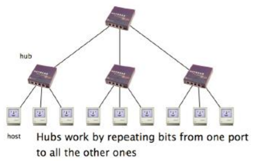
Ethernet header



4.5 Interconnection of Segments at Link Layer

4.5.1 Hub (historic)

- Sends each bit everywhere
- limited to one LAN technology – can't interconnect different rates/ formats
- limited number of nodes & distances (max 2500m Ethernet)


hub
host
Hubs work by repeating bits from one port to all the other ones

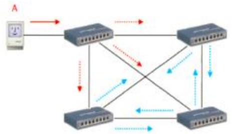
4.5.2 Switch

Switches connect two or more LANs together at the Link layer, acting as L2 gateways. Switches **don't** have a MAC, they are *transparent.*

Switches are "store-and-forward" devices, they

- extract the destination MAC from the frame
- look up the MAC in a table (using exact match)
- forward the frame on the appropriate interface

While flooding enables automatic discovery of hosts, it also creates problems when the networks has loops:



→ Each frame leads to the creation of at least two new frames!

→ exponential increase, with no TTL to remove looping frames

How a forwarding table is built

Switches are plug-and-play devices, they build their **forwarding tables** on their own:

- When a frame arrives:
  - inspect the source MAC-Address
  - associate the address with the port
  - store the mapping in the switch table
  - launch a timer to eventually forget the mapping
- In case of misses, switches simply floods the frames.
- When a frame arrives with an unknown destination
  - forward the frame out of all interfaces except for the source interface

Flooding enables automatic discovery of hosts, it also creates problems when the network has loops

Problem	Loops create major problems
Solution	Reduce the network to one logical spanning-tree
	Upon failure, automatically rebuild the spanning-tree

→ How to build a spanning tree? Spanning tree protocol!

4.5.3 Spanning tree protocol

- Reduce the network to one logical spanning tree
- Upon failure, automatically rebuild a spanning tree

Procedure:

1) Elect a root switch (the one with the lowest identifier)
2) Determine if each interface is on the shortest path from the root & disable it if not

- Detailed procedure with BPDUs (Bridge Protocol Data Unit):
  - Each switch X iteratively sends **BPDU(Y,d,X)** to all neighbors
    - Y = switch ID of switch which X considers to be root
    - d = # hops X needs to reach the root
    - X = switch ID of sending switch

1) Initially each switch proposes itself as root
   → sends $(X, 0, X)$ on all its ports
2) Upon receiving (Y, d, X) each switch checks if Y is a better root (has a lower switch ID). If this is the case, it considers Y as the new root and floods the updated BPDU.
3) Switches compute their distance to the root for each port (simply add 1 to received distance, if shorter, flood)
4) Switches disable ports which are not on the shortest-path

- To prevent failures (any switch, link or port can fail):
  - Root switch continuously sends BPDU messages and all the other switches forward this message
  - Failures are detected through timeout. If switch X did not receive a message from the root after some time (timeout), it claims to be the root.

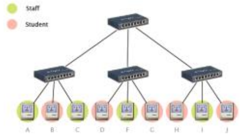
4.5.4 Virtual Local Area Network (VLAN)

As the network scales, network operators like to segment their LANs:

- Improves security ("build islands" and set firewalls around them, reduced visbility) -> smaller attack surface
- Improves performance (limit the overhead of broadcast traffic, i.e. ARP)
- Improves logistics (separates traffic by role, e.g. staff, student, visitors,..)

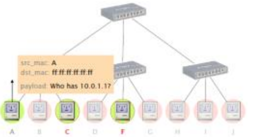
Physically rewiring is a major pain -> do in software!

Definition: A VLAN logically identifies a set of ports attached to one (or more) Ethernet switches forming one broadcast domain.

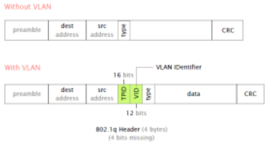


Switches need configuration tables telling them which VLANs are accessible via which interface.

Example: A (Staff VLAN) sends a broadcast frame (i.e. an ARP request). That frame should only be received by staff members (C,F)

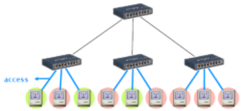


To identify VLAN, switches add a new header when forwarding traffic to another switch (802.1q header).



With VLANs, Ethernet links are divided into two sets:

- Access links: only belong to one VLAN (do not carry a 802.1q header)
- Trunk links: carry traffic for more than one VLAN (thus carry 802.1q tagged frames)



Each switch runs one MAC learning algorithm for each VLAN.

- When a switch receives a frame with an unknown or a broadcast destination, it forwards it over all the ports that belong to the same VLAN.
- When a switch learns a source address on a port it associates it to the VLAN of this port and only uses it when forwarding frames on this VLAN.

Switches can also compute per-VLAN spanning-tree allowing a distinct SPT for each VLAN. This allows the operators to use more of their links.

5 The Network Layer

IP (Internet Protocol): Used on layer 3, global best-effort delivery

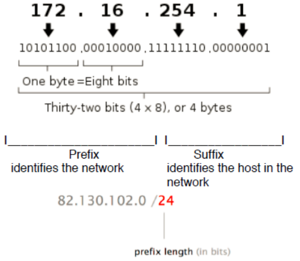
5.1 IP Addresses (IPv4)

IP address: 32 bit number associated to a network interface

Smallest IP address: 0.0.0.0

Largest IP address: 255.255.255.255

➔ This IP address notation is called "dotted-quad notation"



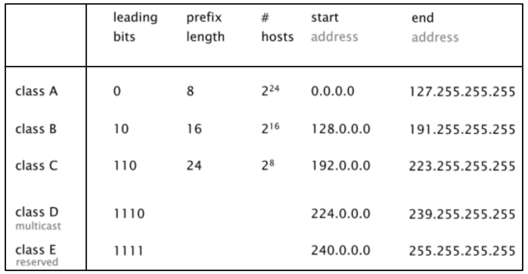
Structure of an IP address:

- Prefix (network address) and suffix (host address)
- Slash notation used to indicate prefix length (in bits)
  e.g. 82.130.102.162/24
  → Prefix = 82.130.102 = 01010010.10000010.01100110
  → Suffix = 162 = 10100010
- /24 means that 8 bits are left for the host addresses
  → 256 hosts possible in /24 networks
  However, the first and the last IP address are not used (IPv4):
  - First: identifies a network (e.g. 82.130.102.0)
  - Last: used as broadcast address (e.g. 82.130.102.255)
  → Only 254 usable hosts possible in /24 networks
- Sometimes the prefix is indicated using an address and a mask
  e.g. address: 82.130.102.162, mask: 255.255.255.0
  → ANDing the address and the mask yields the prefix
- Calculation example with 82.130.0.0/17

- # addressable hosts:	$2^{(32-17)} - 2 = 32'766$
- Prefix mask:	255.255.128.0
- Network address:	82.130.0.0
- Broadcast address:	82.130.127.255
- First host address:	82.130.0.1
- Last host address:	82.130.127.254

Routers: Use the prefix of an IP (network address) to forward packets, the host part is only used in local networks. This hierarchical addressing allows to add new hosts without changing or adding forwarding rules to every single router in a network.

Originally, there were only 5 allocation sizes (classful networking):



	leading bits	prefix length	# hosts	start address	end address
class A	0	8	2^{24}	0.0.0.0	127.255.255.255
class B	10	16	2^{16}	128.0.0.0	191.255.255.255
class C	110	24	2^{8}	192.0.0.0	223.255.255.255
class D multicast	1110			224.0.0.0	239.255.255.255
class E reserved	1111			240.0.0.0	255.255.255.255

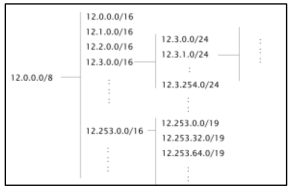
→ Class C was too small for most companies, therefore they requested class B. This quickly led to IP address exhaustion.

→ Problem solved with CIDR

CIDR (Classless Inter-Domain Routing):

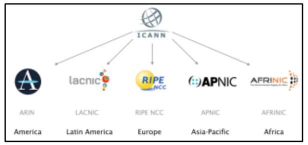
- Flexible division between network and host addresses
- Must specify address and mask (classful networking communicated this in the first address bits)
- Maximal IP address waste bounded to 50%
  e.g. a company needs 255 addresses, this is too much for /24 (only 254 addresses) so they get /23 (510 addresses)

Hierarchical structure of IP addresses (today):



Allocation of IP addresses is also hierarchical:

- ICANN (Internet Corporation for Assigned Names and Numbers) allocates large prefixes to Regional Internet Registries (RIR)

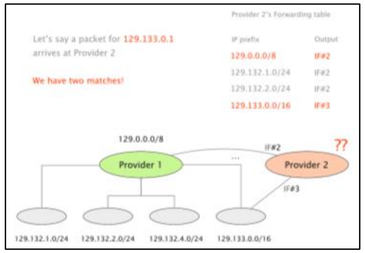


- RIRs then allocate parts of these prefixes to ISPs (Swisscom, …)

5.1.1 IP Forwarding

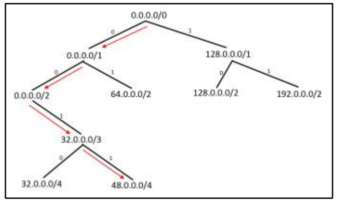
Forwarding table:

- Each router maintains a forwarding table which contains the IP prefixes and matches them with its interfaces (interface = port of a router).
- Upon reception of a package, a router performs an IP lookup to find the matching prefix and it forwards the packet through the corresponding interface.
- CIDR makes forwarding harder, since one packet can match many prefixes → To resolve ambiguity, forwarding is done along the most specific prefix (i.e. the longer one)
  → Longest Prefix Matching
- Example:



IP tree:

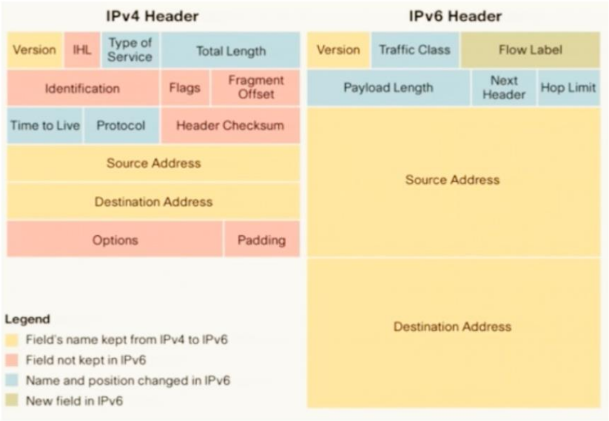
- Instead of a list containing one entry per prefix an IP tree can be used → more efficient lookups
- Example: Destination IP address = 51.0.0.2
  (Binary IP = 00110011.00000000.00000000.00000010)



→ Forward packet over interface associated with 48.0.0.0/4

- Faster tree lookups can be achieved by:
  - compressing the tree
  - adding a cache and saving frequent destinations (only cache lookup necessary…)

5.1.2 IP Header



Most important IPv4 header elements:

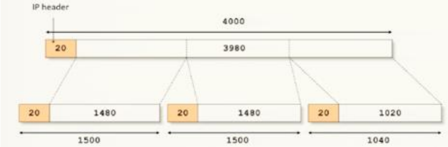
- Version: "4" = IPv4 or "6" = IPv6
- Type of Service (ToS): allows different packets to be treated differently (e.g. low delay for voice, high BW for video, …)
- Identification, Flags & Fragment offset: used to organize fragmentation of large packets (not used today anymore)
- Time To Live (TTL): Number which is decremented by 1 at each router, the packet gets discarded if it reaches 0
  (Linux/Mac TTL: 64, Windows TTL: 128)
- Protocol: higher level protocol used ("6" = TCP, "17" = UDP)
- Source / Destination IP addresses: 32 bit addresses

IPv6 header:

- With respect to IPv4, IPv6 is simpler
- 128 bit addresses instead of 32 bit addresses
- The header only consists of:
  Version, traffic class, flow label, payload length, next header, hop limit, source address, destination address
  → Only what is necessary & more secure!

5.1.3 IPv4

Second layer of the packet is used for fragmentation of the packet



Fragmentation slows the router, so it's mostly deactivated.

Christmas tree attack: Crush a router by sending a packet with all bits set to one in the options.

Bits which are not in use anymore are tried to use for other things.

Default TTL: windows: 128 unix: 64

Problem with IPv4 options: all of them must be processed by each router which is slow In IPv6, only one optional header must be processed.

5.1.4 IPv6

got rid of everything what wasn't necessary → not compatible with the old systems → difficult implementation

IPv6 address have a slightly different notation:

- Addresses are 128bit long → we switch to a HEX notation
- In general, IPv6 addresses are represented by eight colon-separated blocks of up to four hexadecimal digits each.
- In a block, leading zeros can be omitted
- we can use the "::" symbol to compress one or more consecutive zero blocks (can only be used **once** in a single IPv6 address!)

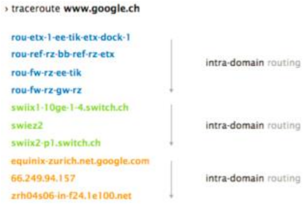
Ex: 2001:0db8:0000:0000:0000:ff00:0042:8329 → 2001:db8::ff00:42:8329

flow label: source and destination/ allows routers to identify packets which belong to the same flow.

Note: IPv6 no longer has a broadcast address!

5.2 Routing

- Inter Domain routing:
  Find paths between networks
- Intra Domain routing:
  Find paths within networks



```
› traceroute www.google.ch

rou-etx-1-ee-tik-etx-dock-1
rou-ref-rz-bb-ref-rz-etx
rou-fw-rz-ee-tik                      intra-domain routing
rou-fw-rz-gw-rz
swiix1-10ge-1-4.switch.ch
swiez2                                intra-domain routing
swiix2-p1.switch.ch
equinix-zurich.net.google.com
66.249.94.157                         intra-domain routing
zrh04s06-in-f24.1e100.net
```

5.3 Intra Domain Routing

Find a good path within a network:

A good path is a path that minimizes some network-wide metric.

Good path: A path that minimizes some network-wide metric (typically delay, load, loss or cost)

→ Assign to each link a weight (usually static) and compute shortest path to each destination. Weights can be:

- Proportional to the distance (minimizes end-to-end delay)
- Inversely proportional to link capacity (throughput maximized) e.g. 10 Gbps links get weight 3 and 1 Mpbs links weight 120

5.3.1 How do routers compute shortest paths?

(see section 2: "How to Compute a Valid Routing State")

- Use tree-like topologies (spanning-tree, LAN) -> not good, no redundancy
- Rely on a global network view: Link state, SDN
- Rely on distributed computation: Distance-vector, BGP

In practice tree-based forwarding is only used within LAN.

5.3.2 Link-state protocols

- Routers build a precise map of the network by flooding local view to everyone:
  - Routers keep track of incident links and costs
  - Each router broadcasts its own link-state
  - Routers run Dijkstra's algorithm on the corresponding graph (see section 2: "How to Compute a Valid Routing State")
- Flooding: Done as in L2 learning, except that it's reliable
  - Starting node sends its link-state on all its links
  - Next nodes do the same, except for the links where the in-formation was received
  - All nodes are ensured to receive the latest version of all link-states:
    - Problems: Packet loss, out of order arrival
    - Solutions: ACK & retrans., Seq. No., TTL for link-states
- Conditions for flooding:

Topology change	link or node failure/recovery
Configuration change	change of the cost of a link
Periodically	refresh the link-state information

- Link-state protocols detect topology changes by sending periodically "Hello" to other routers. 2 parameters:
  - How often a "Hello" is sent (e.g. every 30s)
  - After how many misses a link is considered dead
  - ➔ Nonetheless, inconsistencies between the different nodes can arise (loops or dead ends)

5.3.2.1 Discover Problems in a network

By default, Link-State protocols detect topology changes using software-based beaconing. Routers periodically exchange "Hello" in both directions (30s, declare the the router for dead after 3 misses, this very long, some cases can detect this in <50ms)

Tradeoff

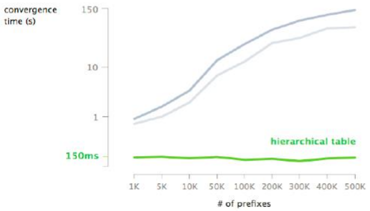
- detection speed
- bandwidth and CPU overhead
- false positive/negatives

During network changes, the link-state database of each node might differ:
→ black holes (router is not aware of the problem and keeps sending traffic on a broken link)
→ forwarding loops, during reconfiguration of a network

5.3.2.2 Convergence

Process during which the routers seek to actively regain a consistent view of the network. Multiple factors influence convergence time:

Factor	Time router takes to...	Time	Improvements
Detection	Realizing that a link or a neighbor is down	~ms	Smaller timers
Flooding	Flooding the news to entire network	~ms	High priority flooding
Computation	Recomputing shortest-path using Dijkstra	~ms	Incremental algo-rithms (don't consider the full network)
Table update	Updating their forwarding table	Long! ~min	Better table design



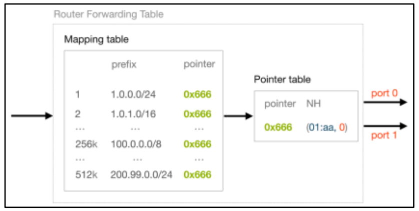
The problem is that forwarding tables are flat

- Entries do not share any information even if they are identical
- Upon failure, all of them have to be updated inefficient, but also unnecessary

Solution → Hierarchical table

5.3.2.3 Router Forwarding Table

For more efficient forwarding table updates, the routing forwarding table gets replaced by a mapping table and a pointer table. This way only one entry in the pointer table needs to be updated in case a router crashes (instead of all entries which were directed towards this router):



5.3.2.4 Link-state protocols

OPSF: Open shortest-path first	**IS-IS:** Intermediate Systems Intermediate Systems
• Used in many enterprise & ISPs • Works on top of IP • Only route IPv4 by default	• Used mostly in large ISPs • Work on top of link-layer • Network protocol agnostic

5.3.3 Distance-Vector protocols

- Bellman-Ford algorithm (see section 2: "How to Compute a Valid Routing State")
- Situations that cause nodes to send new distance-vectors:

Configuration change	Link or node failure/ recovery
Topology change	Link cost change
Periodically	Refresh the link-state information every e.g. 30min account for possible data corruption

- Cost of a link changes (see example in slides):
  - Edge gets a smaller weight -> converges fast
  - Edge gets a higher weight -> converges slowly
  - ➔ Called "count-to-infinity" problem, in easy networks this problem can be resolved by "poisoned reverse"
- **Poisoned reverse:**
  If router X uses the adjacent router Q to reach router Z on its shortest path, router X announces router Q an infinite cost to reach router Z. This way Q will never include the route via X (and itself) in its own shortest path (no loops…).
  - ➔ See example in slides
- Poisoned reverse does only work for networks with few nodes, however it can be used for more complex networks if one de-fines infinity to be small (e.g. 16). Then infinity is reached quite fast and emerging loops get stopped quickly.

5.3.4 Link-State vs. Distance-Vector routing

	Message Complexity	Convergence Speed	Robustness

Link-State
n := #Nodes
E := #links
	$O(nE)$ message sent	Relatively fast	Node can advertise in-correct link cost
			Nodes compute their own table

Distance-Vector	Between neighbors only	Slow	Node can advertise in-correct path cost Errors propagate

5.4 Inter-Domain routing

AS (Autonomous System):

- The Internet is a network of networks. One single network of these inter-connected networks is called Autonomous System.
- Each AS has an identification number (encoded in 16 bits)

5.4.1 BGP

- Used by the different AS to communicate with each other. They exchange information about the IP prefixes they can reach directly or indirectly.

BGP needs to solve three key challenges

1	Scalability	There is a huge # of networks and prefixes 600k pre-fixes, >50000 networks, millions of routers
2	Privacy	Networks don't want to divulge internal topologies or their business relationships
3	Policy En-forcement	Networks need to control where to send/ receive traf-fic without an Internet-wide notion of link cost metric

Link state routing does not solve these challenges:

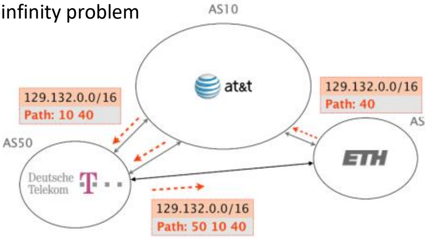
- Floods topology information -> *high processing overhead*
- Requires each node to compute the entire path -> *high processing overhead*
- Minimizes some notion of total distance -> *works only if the policy is shared and uniform*

Distance-Vector routing is on the right track, but not really there yet

Pros	Hide details of the network topology nodes determine only "next-hop" for each destination
Cons	It still minimizes some common distance impossible to achieve in an inter domain setting
	It converges slowly counting-to-infinity problem

5.4.1.1 Path-vector routing

- BGP relies on path-vector routing:
  - Similar to distance-vector routing
  - Advertises entire path instead of distances (loop detection…)
  - Supports flexible routing policies
  - Avoids count-to-infinity problem
- Example:



➔ ETH sees itself in the path and discards the route

- BGP routers repeat the following steps:
  - Receive routes from neighbors
  - Select best route for each prefix
  - Export the best route to all neighbors

5.4.1.2 Local routing policies

Each AS can apply local routing policies. Each AS is free to:

- Select and use any path (preferably the cheapest one) e.g. ETH could reach AT&T over Swisscom because it's cheaper (even if a direct link between ETH and AT&T existed)
- Decide which path to export (if any) to which neighbor e.g. Swisscom could hide its connection to AT&T to ETH, to stop ETH from directing all AT&T traffic over Swisscom.

5.4.2 BGP Policies – follow the money

Principle: BGP is a "follow the money" protocol

The internet topology is shaped according to **business relationships.**

Intuition: 2 ASes connect only if they have a business relationship. There are 2 main business relationships today:

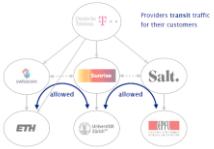
1. Customer/ provider
2. Peer/ peer
3. May less important ones (siblings, backups,…)

Customer/ provider:

- Customers pay providers to get internet connectivity
- 95th percentile rule: the provider records the # bytes sent or received from each customer every 5 minutes. At the end of a month these recordings get sorted in decreasing order and the top 5% are removed (to compensate for the bursty behavior of the internet). The customer is billed with respect to the highest remaining value.
- Most ISPs offer discount traffic unit prices when customers precommit to a certain volume. If the volume is exceeded, the customers pay for the extra volume. This way providers can estimate the amount of traffic for each link and can dimension them appropriately. Additionally, they have a secure income.

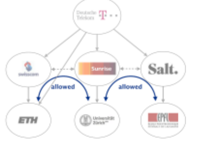
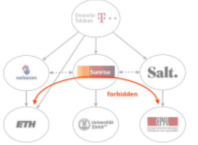
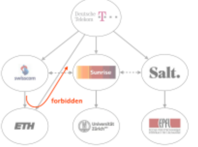
Peer/ peer:

- Peers don't pay each other for connectivity out of common interest.
  Example: Two tier2 ISPs exchange a lot of data. For every exchange they must pay their tier1 ISP. To save money they can establish a direct link between each other (for a fix cost) and then exchange data for free (cheaper for both).



5.4.2.1 Route Selection

Which path to use? -> control outbound traffic

	Providers transit traffic for their direct customers
	Peers do not transit traffic if none of their customers is involved
	Customers do not transit traffic between their providers

RegEx for valid paths: $(c2p)^{\{0-n\}}(p2p)^{\{0-1\}}(p2c)^{\{0-m\}}$ ({0-n}: 0 or 1 or… n)

To control traffic flow each AS needs to select which BGP routes it wants to use and to decide which BGP routes it wants to export:

- **Route selection:**
  Routes coming from customers are preferred over routes from peers, which are again preferred over routes from providers.

  BGP route selection: Customer > Peer > Provider

  This policy is implemented at the AS via the LOCAL-PREF (i.e. give Customer a LP 300, peer: LP 200, provider: LP 100)

5.4.2.2 Route Exportation

		Send to:		
		Customer	Peer	Provider
From:	Customer	✓	✓	✓
	Peer	✓	✗	✗
	Provider	✓	✗	✗

Customers pay → They want to be connected to everybody (receive data from everybody and send data to everybody).

Because of all these BGP policies it is necessary that the tier1 ISPs are connected through a full-mesh of peer links. Otherwise the net-work would be partitioned (not everybody could reach everybody).

5.4.3 BGP Protocol

BGP is used for two different applications:

- **eBGP (external BGP):**
  - Connects gateway routers of different ASes
  - Needed to reach external destinations from within an AS
  - BGP advertisements (BGP UPDATEs) are exchanged between routers to communicate (BGP info not sent with user packets)
- **iBGP (internal BGP):**
  - Connects routers within an AS
  - Used to spread externally learned routes internally.
  - For iBGP so called interior gateway protocols (IGP) are used, e.g. OSPF (a link-state protocol, see section 5: "Intra-Domain Routing").
  - Example: The gateway router X within AS1 is connected to AS2. The gateway router X internally spreads the information on how to reach AS2. This way all internal routers can send the packets destined to AS2 directly to X (iBGP). Other gateway routers within AS1 provide the information that they can reach AS2 also to the ASes they are connected to (eBGP). Therefore, everybody knows where to send packets to reach anybody (and nobody knows too much).

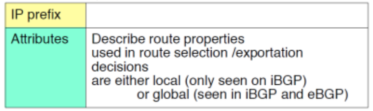
5.4.3.1 BGP messages

BGP uses 4 basic messages

Type	Used to…
OPEN	Establish TCP-based BGP session
NOTIFICATION	Report unusual conditions
UPDATE	Inform neighbors of: - a new best route - a change in the best route
KEEPALIVE	Inform neighbors that the connection is alive

5.4.3.2 BGP UPDATE

BGP UPDATEs carry an IP prefix together with a set of attributes. These attributes describe route properties, are used in route selection/ exportation decisions and are either local or global.

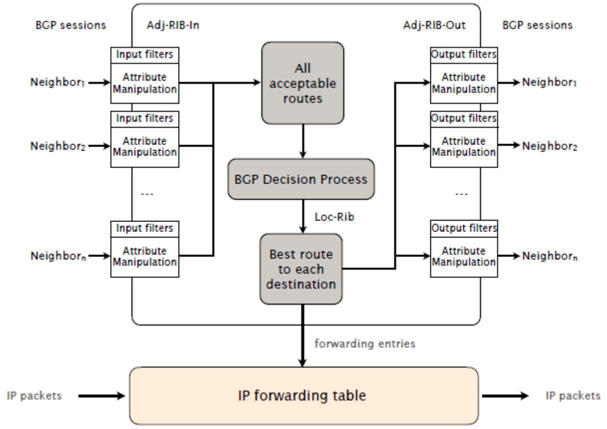


IP prefix	
Attributes	Describe route properties used in route selection /exportation decisions are either local (only seen on iBGP) or global (seen in iBGP and eBGP)

Attributes	Usage
NEXT-HOP	egress point identification - global attribute, indicates where to send the traffic next - contains IP address of the NEXT-HOP router - is set when the route enters an AS, it does not change within the AS
AS-PATH	- global attribute that lists all the ASes a route has Traversed (in reverse order!) - A new AS is added to *the beginning* of the list - loop avoidance, inbound/ outbound traffic control - Ex: Example: a BGP UPDATE containing the attribute “AS-PATH: 50 10 40” last passed AS50
LOCAL-PREF	- represents how “preferred” a route to an external AS is (in case there are routes over multiple providers) - outbound traffic control: local attribute sets at the border - “Good” routes get a high LOCAL-PREF value
MED	(Multiple Exit Discriminator) (global): - Inbound traffic control: set by customers to tell their provider over which of multiple possible external routes they like to receive traffic. - “Good” routes get a low MED value - In case all routes cost the same for the provider, it will choose the route for which it received the lowest MED from the customer. - In case the provider has different LOCAL-PREFs for the routes to the customer, it will ignore the received MED and just use the route with the highest LOCAL-PREF.

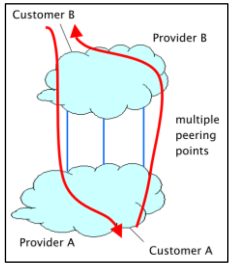
➔ The network which is sending the traffic always has the final word when it comes to deciding where to forward. The network which is receiving the traffic can just influence remote decisions (but can’t control them).

Each BGP router processes UPDATEs according to a precise pipeline:

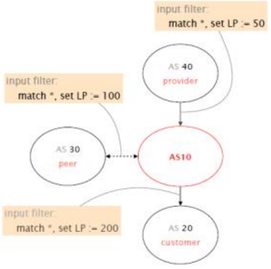


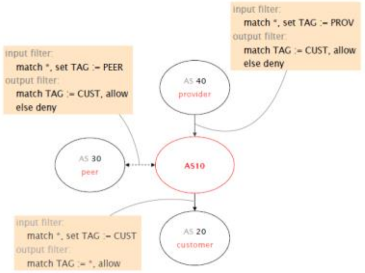
BGP elects **a single route** (BGP is a single path protocol) for each prefix according to the following criteria (in case the criteria are equal, look at the next criterion):

1. Higher LOCAL-PREF
2. Shorter AS-PATH length
3. Lower MED
4. Learned via eBGP instead of iBGP
5. Lower IGP metric to NEXT-HOP
6. Smaller source IP address (tie-break)



ASes are selfish:
The criteria 4 and 5 aim at directing traffic as quickly as possible out of the AS (early exit routing or “hot potato routing”). ASes do this to safe money (needs less network resources). This leads to asymmetric routing (packets don’t take the same path when being sent between two ASes).





5.4.4 BGP Problems

Reachability
Because of BGP policies reachability in a network graph is not guaranteed simply because a path between two nodes exist.
Example: A customer C may have two providers P1 & P2 which have no peer link and which don’t share a common provider. Hence, the P1 & P2 are connected only via their customer C. However, BGP prohibits the communication between P1 & P2 via their customer C.

Security (see BGP security)

- ASes can advertise any prefix (even if they don’t own them…)
- ASes can arbitrarily modify route content (e.g. the AS-PATH)
- ASes can forward traffic along different paths than the advertised ones.

Convergence
With arbitrary policies, BGP may fail to converge (oscillations). Determining whether a finite BGP network converges is PSPACE-hard. Determining whether an infinite BGP network converges is Turing-complete.

➔ It is not possible to guarantee the absence of oscillations in a given BGP configuration.

➔ However, if all AS policies follow the “Gao-Rexford” rules, BGP is guaranteed to converge (e.g. oscillations require cycles in the network, however cycles make no sense from an economic perspective…).

Performance
BGP path selection is mostly economical and not based on accurate performance criteria.
Example: BGP tends to select the path which crosses the least amount of ASes, however there might be a physically shorter path which crosses 1 or 2 more ASes. Therefore, the selected path does not have the shortest possible RTT (round-trip time).

Anomalies
BGP configuration is hard to get right, possible reasons are:

- BGP is both “bloated” (many options) and underspecified
- BGP is often manually configured (error-prone)
- BGP is not abstract enough (e.g. configurations of a single router can influence an AS-wide policy).

Relevance
Rapid changes: Many customers buy direct links to content networks (e.g. Netflix, YouTube, Facebook, …), hence heavy traffic is routed around the ISPs. Therefore, ISPs get less important and earn less money. IXPs on the other hand get more important.

6 The Transport Layer

6.1 Overview

6.1.1 What do we need in the transport layer

Functionality implemented in network:

- Keep minimal (easy to build, broadly applicable)

Functionality implemented in the application:

- Keep minimal (easy to write)
- Restricted to application-specific functionality

Functionality implemented in the “network stack”

- The shared networking code on the host
- This relieves burden from both app and network
- The transport layer is a key component here

We need to provide service in between these two layers:

- Application layer:
  - Communication for specific applications
  - e.g., HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP)
- Network layer:
  - global communication between hosts
  - hides details of the link technology
  - e.g. Internet Protocol (IP)

6.1.2 What problems should be solved here?

- Data delivering, to the correct application
  - IP just points towards next protocol
  - Transport needs to demultiplex incoming data (ports)
- Files or bytestreams abstractions for the applications
  - Network deals with packets
  - Transport layer needs to translate between them
- Reliable transfer (if needed)
- Not overloading the receiver
- Not overloading the network → fairness

6.1.3 What is needed to address these?

- Demultiplexing: identifier for application process
  - Going from host-to-host (IP) to process-toprocess
- Translating between bytestreams and packets:
  - Do segmentation and reassembly
- Reliability: ACKs and all that stuff
- Corruption: Checksum
- Not overloading receiver: "Flow Control"
  - Limit data in receiver's buffer
- Not overloading network: "Congestion Control"

6.1.4 Connections (or sessions)

Reliability requires keeping state

- Sender: packets sent but not ACKed, and related timers
- Receiver: noncontiguous packets

Each bytestream is called a connection or session

- Each with their own connection state
- State is in hosts, not network!

Example: I am using HTTP to access content on two different hosts, and I'm also ssh'ing into another host. How many sessions is this? - 2??

6.2 Sockets and Ports

6.2.1 Sockets: an operating system abstraction

A socket is a software abstraction by which an application process exchanges network messages with the (transport layer in the) operating system.

3 basic functions:

- Instantiation: socketID = socket(..., socket.TYPE)
- Sendto: socketID.sendto(message, …)
- Recvfrom: socketID.recvfrom(...)

Two important types of sockets:

- UDP socket: TYPE is SOCK_DGRAM
- TCP socket: TYPE is SOCK_STREAM

6.2.2 Ports: a network abstraction

- Not a physical interface on a switch, but a logical interface on a host
- Identifies which packets belongs to which application (socket)
- Source and destination port numbers are contained in the transport header (16 bit for each of them)
- OS stores mapping between sockets and ports:

Ports: in packets	Socket: in OS

- Port numbers:
  - 0 – 1023: agreement which services run which ports (e.g. ssh: 22, http: 80, https: 443)
  - 1024 – 65535: Temporary ports (randomly assigned to the clients)

➔ The operating system knows the mapping between sockets and ports!

6.2.3 Multiplexing & Demultiplexing

Host receives IP datagrams

- Each datagram has source and destination IP address,
- Each segment has source and destination port number

Host uses IP addresses and port numbers to direct the segment to appropriate socket (→ images p.661-663)

6.2.4 Connection Mappings

For UDP ports (SOCK_DGRAM):

- OS stores (local port, local IP address) <--> socket

For TCP ports (SOCK_STREAM):

- OS stores (local port, local IP, remote port, remote IP) <--> socket

Why the difference? Because TCP is connection oriented -> need to know the receiver. Problem: mobile connections (e.g. smartphone) -> local IP changes -> TCP will close session!

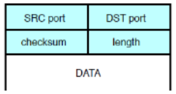
6.3 UDP (User Datagram Protocol)

Lightweight communication between processes

- Avoid overhead and delays of ordered, reliable delivery
- Send messages to and receive them from a socket

UDP described in RFC 768 – (1980!)

- IP plus port numbers to support (de)multiplexing
- Optional error checking on the packet contents
- (checksum field = 0 means "don't verify checksum")



Advantages of UDP:

- Fine control over what data is sent at which point in time (as soon as an application writes to the socket UDP will package data and send a packet)
- No delay for connection establishment (UDP just starts sending)
- No connection state (no buffers, sequence #s, ...) -> easier handling
- Small packet header (UDP header is only 64 bit = 8 byte)

Applications:

- Streaming applications: Retransmission is often pointless (already too late), e.g. telephone calls, video conferences, gaming, etc.
  - ➔ Modern streaming protocols however use TCP (and HTTP)
- Simple query protocols like DNS (Domain Name System): Connection establishment would double the cost…

6.4 TCP (Transmission Control Protocol)

Connection-oriented, reliable, full-duplex transport service

- Multiplexing/Demultiplexing among processes
- Reliability: Dealing with possible errors (e.g. retransmission of lost and corrupted packets, reordering, duplication, …)
- Full duplex stream-of-bytes service (sends/ receives a stream of bytes)
- Connection-orientation: Explicit set-up and tear-down of TCP sessions
- Flow control: ensure that sender doesn't overwhelm receiver
- Congestion control: Dynamic adaption to network path's capacity
  - ➔ What UDP offers and more

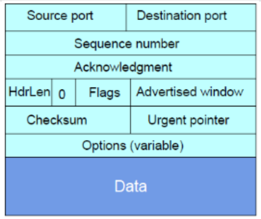
6.4.1 TCP Features

Reliability:

- ACKs and sequence numbers to identify payloads
- Checksums to check for data corruption
- Single timer set after each payload is ACKed (for next payload):
  - If timer goes off, the payload is resent and the timer is set again (with the double timeout period)
  - Timeout period based on estimate of RTT
- Retransmission in case of lost/corrupted data:
  - TCP retransmits based on timeouts and duplicate ACKs
  - Duplicated ACKs can be used to trigger a fast retrans-mission
- Sliding window for flow control (allow W contiguous bytes to be in flight)
- Cumulative ACKs (Selective ACKs (full information) also supported)

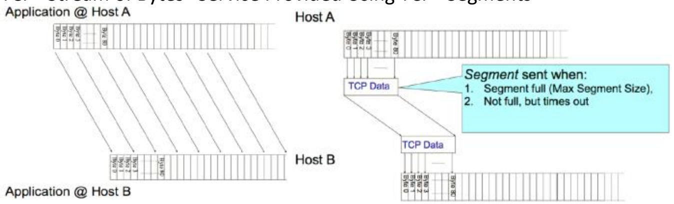
6.4.2 TCP Header

- Sequence number: Starting byte offset of data carried in this segment
- Acknowledgment: gives seqno just beyond highest seqno received in order
  → "What Byte is Next"
- HdrLen: Number of 4-byte words in TCP header; 5 = no options
- 0: "Must Be Zero" 6 bits reserved
- Urgent pointer: Used with URG flag to indicate urgent data
- Flags: SYN ACK FIN RST PSH URG



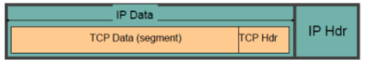
6.4.3 Segments and Sequence Numbers

TCP "Stream of Bytes" Service Provided Using TCP "Segments"



Segment sent when:
1. Segment full (Max Segment Size),
2. Not full, but times out

6.4.4 TCP Segment



IP packet

- No bigger than Maximum Transmission Unit (MTU)
- E.g., up to 1500 bytes with Ethernet

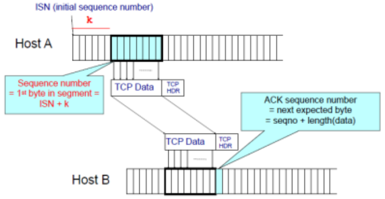
TCP packet

- IP packet with a TCP header and data inside: TCP header ≥ 20 bytes long

TCP segment

- No more than Maximum Segment Size (MSS) bytes
- E.g., up to 1460 consecutive bytes from the stream
- MSS = MTU – (IP header) – (TCP header)

Sequence numbers



Sequence number = 1st byte in segment = ISN + k

ACK sequence number = next expected byte = seqno + length(data)

6.4.5 ACKing and Sequence numbers

- Sender sends packet
  - Data starts with sequence number X
  - Packet contains B bytes: X, X+1, X+2, ...X+B-1
- Upon receipt of packet, receiver sends an ACK
  - If all data prior to X already received:
    - ACK acknowledges X+B (because that is next expected byte)
  - If highest contiguous byte received is smaller value Y
    - ACK acknowledges Y+1
    - Even if this has been ACKed before

Normal Pattern:

- Sender: seqno=X, length=B
- Receiver: ACK=X+B
- Sender: seqno=X+B, length=B
- Receiver: ACK=X+2B

…

Seqno of next packet is same as last ACK field

6.4.6 Sliding Window Flow Control

Advertised Window: W -> Can send W bytes beyond the next expected byte

→ Limits number of bytes sender can have in flight

- W (in bytes), which we assume is constant
- RTT (in sec), which we assume is constant
- B := Bandwidth (in bytes/sec)

Data transfer speed

- If W/RTT < B, the transfer has speed W/RTT
- If W/RTT > B, the transfer has speed B

Advertised window limits rate

- Sender can send no faster than W/RTT bytes/sec
- Receiver only advertises more space when it has consumed old arriving data - Receiver uses W to prevent sender from overflowing buffer

→ In original TCP design, that was the sole protocol mechanism controlling sender's rate. Whats missing? → fairness

6.4.7 Implementing Sliding Window

Both sender & receiver maintain a window

Sender: not yet ACK'ed	Receiver: not yet delivered to application

Left edge of window:

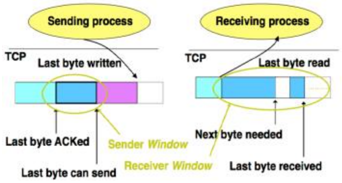
- Sender: beginning of unacknowledged data
- Receiver: beginning of undelivered data

For the sender: Window size = maximum amount of data in flight

For the receiver: Window size = maximum amount of undelivered data

Sliding Window: Allow a larger amount of data "in flight"

- Allow sender to get ahead of the receiver



- .... Though not too far ahead

Sliding window summary:

- Sender: window advances when new data ack'd
- Receiver: window advances as receiving process consumes data

Receiver advertises to the sender where the receiver window currently ends ("righthand edge")

- Sender agrees not to exceed this amount
- It makes sure by setting its own window size to a value that can't send beyond the receiver's righthand edge

6.5 TCP Connection Establishment & ISN

6.5.1 ISN

Sequence number for the very first byte -> E.g., Why not just use ISN = 0?

Practical issue:

- IP addresses and port #s uniquely identify a connection
- Eventually, though, these port #s do get used again
- ... small chance an old packet is still in flight

TCP therefore requires changing ISN

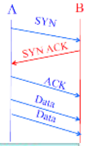
- Set from 32-bit clock that ticks every 4 microseconds
- ... only wraps around once every 4.55 hours

To establish a connection, hosts exchange ISNs

6.5.2 Establishing a TCP Connection

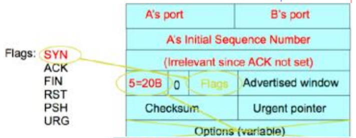
Each host tells its ISN to the other host. Three-way handshake to establish connection:

1) Host A sends a SYN (open; "synchronize sequence numbers")
2) Host B returns a SYN acknowledgment (SYN ACK)
3) Host A sends an ACK to acknowledge the SYN ACK

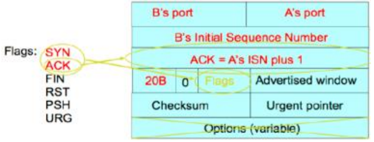


Detailed procedure:

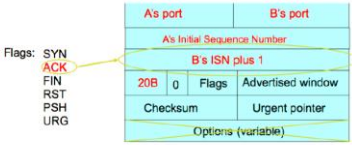
1) A's initial SYN packet
   A tells B it wants to open a connection



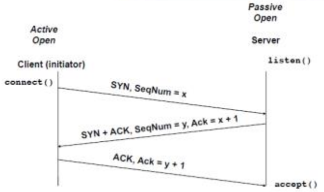
2) B's SYN ACK packet
   B tells A it accepts, and is ready to hear the next byte… upon receiving this packet, A can start sending data



3) A's ACK of the SYN ACK
   A tells B it's likewise okay to start sending… upon receiving this packet, B can start sending data



Timing Diagram: 3-way handshake



6.5.3 What if the SYN packet gets lost?

Suppose the SYN packet gets lost

- Packet is lost inside the network, or Server discards the packet (e.g., listen queue is full)

Eventually, no SYN-ACK arrives

- Sender sets a timer and waits for the SYN-ACK
- ... and retransmits the SYN if needed

How should the TCP sender set the timer?

- Sender has no idea how far away the receiver is
- Hard to guess a reasonable length of time to wait
- SHOULD (RFCs 1122 & 2988) use default of 3 seconds
  - Other implementations instead use 6 seconds

6.5.4 SYN loss and Web downloads

User clicks on a hypertext link

- Browser creates a socket and does a "connect"
- The "connect" triggers the OS to transmit a SYN

If the SYN is lost...

- 3-6 seconds of delay: can be very long
- User may become impatient and click the hyperlink again, or click reload

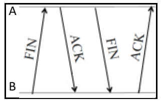
User triggers an "abort" of the "connect"

- Browser creates a new socket and another "connect"
- Essentially, forces a faster send of a new SYN packet!
- Sometimes very effective, and the page comes quickly

6.6 Tearing Down the Connection

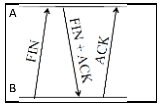
Normal termination, one at a time:

- A: FIN, B: ACK, B: FIN, A: ACK
- A has to wait 1-2 minutes after sending the last ACK to be sure that B received it (otherwise B would send FIN again)
  → A needs to stay connected for 1-2 min



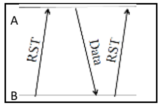
Normal termination, both together:

- B can send an FIN+ACK:
  → A: FIN, B: FIN+ACK, A: ACK

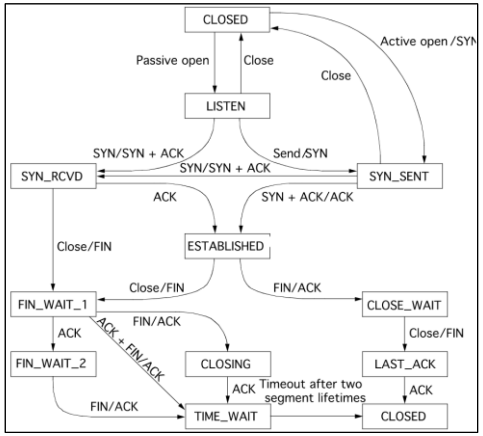


Abrupt termination:

- A sends RESET (RST) to B (e.g. because application process crashed)
- B does not ACK the RST (RST not delivered reliably)
- Any data in flight is lost
- If B sends again some data, A sends RST again



6.7 TCP state transitions



6.8 Reliability: TCP retransmissions

Timeouts and Retransmissions:

Reliability requires retransmitting lost data. Involves setting timer and retransmitting on timeout. TCP resets timer whenever new data is ACKed

- Retransmission of packet containing "next byte" when timer goes off

Example:

- Arriving ACK expects 100
- Sender sends packets 100, 200, 300, 400, 500
  - Timer set for 100
- Arriving ACK expects 300
  - Timer set for 300
- Timer goes off
  - Packet 300 is resent
- Arriving ACK expects 600
  - Packet 600 sent
  - Timer set for 600

Setting the Timeout value:

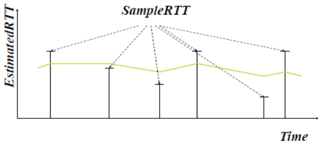
- Timeout value (called RTO (recovery time objective)):
  - RTO too long → inefficient
  - RTO too short → almost all packets get duplicated

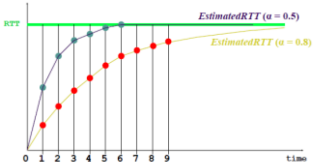
→ Use an estimation of RTT to set RTO

6.8.1 RTT estimation

Use exponential averaging of RTT samples $(0 < \alpha \leq 1)$

$$\boxed{\begin{gathered} SampleRTT = AckRcvdTime - SendPacketTime \\ EstimatedRTT = \alpha * EstimatedRTT + (1 - \alpha) * SampleRTT \end{gathered}}$$





Problem: Ambiguous measurements → How do we differentiate between the real ACK, and ACK of the retransmitted packet?

Solution: Karn/Partridge Algorithm

6.8.2 Karn/Partridge Algorithm

Measure SampleRTT only for original transmissions

- Once a segment has been retransmitted, do not use it for any further measurements
- Computes EstimatedRTT using α = 0.875

Timeout value (RTO) = 2 × EstimatedRTT

Use exponential backoff for repeated retransmissions

- Every time RTO timer expires, set RTO ← 2*RTO
  (Up to maximum ≥ 60 sec)
- Every time new measurement comes in (=successful original transmission), collapse RTO back to 2 × EstimatedRTT

6.8.3 Loss with cumulative ACKs

Rely on duplicate ACKs (for cumulative ACKs):

Example:

- Sender: 100, 200, 300, 400, 500, 600, 700, 800, …
- Received ACKs: 200, 300, 400, 500, 500, 500, 500, …
  - ➔ Lack of ACK progress means 500 hasn't been delivered
  - ➔ Stream of ACKs means packets are still delivered
- Duplicate ACKs are sign of an isolated loss
- Resend upon receiving k duplicate ACKs (TCP uses k = 3)
  - ➔ Less expensive than waiting for a timeout with an exponential backoff algorithm

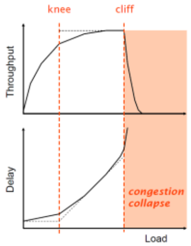
6.9 Congestion

In 1986 the internet collapsed because there was only flow control and no congestion control:

- Problem: Increase in network load results in decrease of useful work done
- What happened:
  → Higher RTTs due to increase in network load
  → RTOs increase until they reach their limit, then all hosts start to retransmit their packets (several packets at a time)
  → Congestion collapse

Characterization of congestion collapse:

- Knee: Point after which
  - throughput increases slowly
  - delay increases quickly
- Cliff: Point after which:
  - throughput decreases quickly
  - delay tends to infinity



6.10 TCP Congestion Control

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

Congestion control differs from flow control but both are provided by TCP

		TCP solution
Flow control	prevents one fast sender from overloading a slow receiver	solved using a receiving window
Congestion control	prevents a set of senders from overloading the network	solved using a "congestion" window

The sender adapts its **sending rate** based on these two windows:

Receiving window RWND	How many bytes can be sent without overflowing the receiver buffer? based on the receiver input
congestion window CWND	How many bytes can be sent without overflowing the routers? based on network conditions
Sender Window	minimum(CWND, RWND)

2 key mechanisms of congestion control:

1) Detecting congestion
2) Reacting to congestion

6.10.1 3 possibilities to detect congestion

- Network could tell the source with a message → This message could be lost as well
- Measure packet delay → Noisy signals, delay often varies (not reliable)
- Measure packet loss → Fail-safe, used in TCP

Packet dropping is the best solution delay- and signaling based methods are hard & risky.

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

duplicated ACKs	mild congestion signal packets are still making it
timeout	severe congestion signal multiple consequent losses

6.10.2 Bandwidth estimation

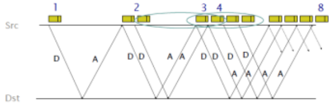
TCP approach is to gently increase when not congested and to rapidly decrease when congested. The goal here is to quickly get a first-order estimate of the available bandwidth:

- Start slow but rapidly increase until a packet drop occurs
- Increase policy:

$$Initially: CWND = 1$$

$$Upon\ receipt\ of\ an\ ACK: CWND += 1$$

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)



The problem with slow start is that it can result in a full window of packet losses

example	- Assume that CWND is just enough to “fill the pipe” - After one RTT, CWND has doubled - All the excess packets are now dropped
solution	We need a more gentle adjustment algorithm once we have a rough estimate of the bandwidth

6.10.3 Bandwidth adaption

The goal here is to track the available bandwidth, and oscillate around its current value. Two possible variations:

1) Multiplicative Increase or Decrease (MI/MD)
   cwnd = a * cwnd
2) Additive Increase or Decrease (AI/AD)
   cwnd = b + cwnd

	Increase behavior	Decrease behavior	fairness
AIAD	gentle	gentle	not fair
AIMD	gentle	aggressive	fair
MIAD	aggressive	gentle	not fair
MIMD	aggressive	aggressive	not fair

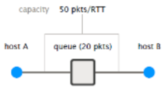
Fairness:

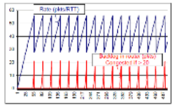
TCP notion of fairness: 2 identical flows should end up with the same bandwidth.

Consider first a single flow between A and B and AIMD:

- without congestion:
  - CWND increases by one packet every ACK
- upon congestion:
  - CWND decreases by a factor 2

This is what you will observe in the router: (throughput over time)

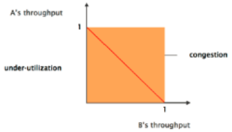




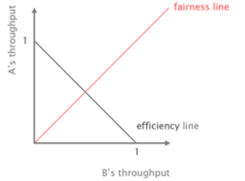
We can analyze the system behavior using a **system trajectory plot:**

Used to analyze the fairness of a given algorithm:

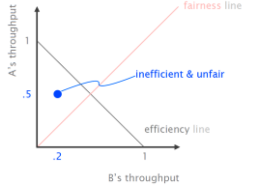
- Plot contains the throughput of host A and host B, as well as an efficiency line and a fairness line (see SysTP 1)
- Below the efficiency line = inefficient | On the efficiency line = efficient
  Above the efficiency line = congested
- Not on fairness line = unfair | On fairness line = fair
  → Examples in SysTP 2 and SysTP 3
- Convergence to fair state desired (only achieved by AIMD) → see SysTP 4



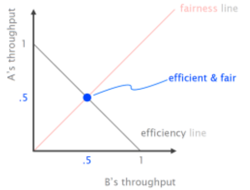
SysTP 1



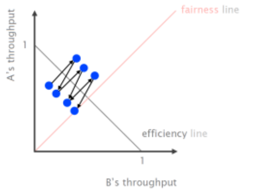
SysTP 2



SysTP 3

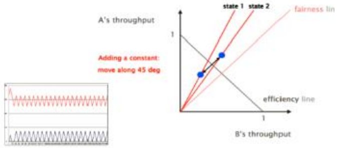


SysTP 4

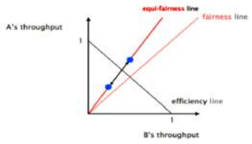


Behavior of different designs:

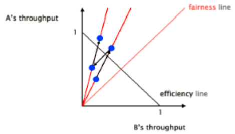
AIAD does not converge to fairness, nor efficiency: the system fluctuates between two fairness states



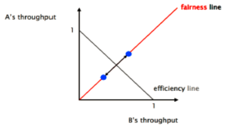
MIMD does not converge to fairness, nor efficiency: the system fluctuates along a equi-fairness line



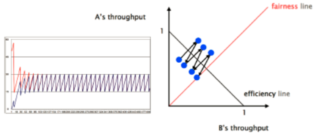
MIAD converges to a totally unfair allocation, favouring the flow with a greater rate at the beginning



If flows start along the fairness line, MIAD fluctuates along it, yet deviating from it at the slightest change



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



Intuition:

- During increase, both flows gain bandwidth at the same rate
- During decrease, the faster flow releases more

In practice, TCP implements AIMD

Implementation	After each ACK, increment cwnd by 1/cwnd (lin. Increase of max. 1 per RTT
Question	When does a sender leave slow-start & start AIMD? -> Introduce a slowstart threshold, adapt it in function of congestion: on timeout: ssthresh = CWND/2

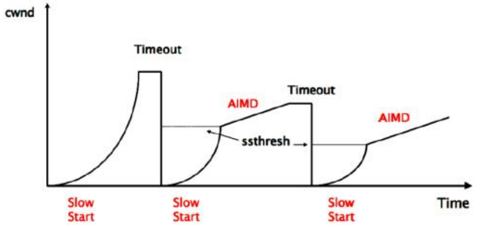
6.10.4 TCP congestion control

```
Initially:
    cwnd = 1
    ssthresh = infinite
New ACK received:        // Additive Increase
    if (cwnd < ssthresh):
        cwnd = cwnd + 1   // Slow Start
    else:
        cwnd = cwnd + 1/cwnd// Congestion Avoidance
    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
```

Two adaptations exist to improve the throughput of the algorithm:

- Fast retransmit: After receiving 3 or more duplicate ACKS for a packet, retransmit the packet
- Fast recovery: Assume mild congestion after receiving 3 duplicate ACKs. Don’t restart from cwnd = 1 with slow start, instead use AIMD and set ssthresh = cwnd = cwnd/2.

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



7 The Application Layer

7.1 DNS (Domain Name System)

DNS: google.ch ↔ 172.217.16.131
Web: http://www.google.ch

The internet has on global system for:

- Addressing hosts: IP by design
- Naming hosts: DNS by accident/ an afterthought

Using internet services can be divided into four logical steps:

Step 1	A person has a name entity she wants to access	www.ethz.ch
Step 2	She invokes an application to perform the task	Chrome
Step 3	The application involves DNS to resolve the name into an IP address	129.132.19.216
Step 4	The application invokes transport protocol to establish an app-to-app connection	

DNS = distributed database which enables to resolve a name into an IP addr.:

- The internet protocol is used for the addressing of the hosts. However, IP addresses are not easy to remember for humans, therefore DNS is used for the naming of the host.
- Names can be mapped on multiple IP addresses (e.g. Google directs traffic destined to www.google.ch to different IPs)
- IP addresses can be mapped on multiple Names (e.g. accessing "RTL Now" via www.rtl-now.de and www.rtl-nau.de)

7.1.1 How does on resolve a name into an IP?

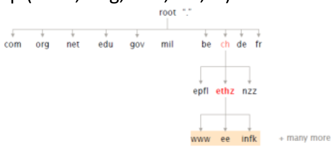
Initially	all host to address mappings were in a file called hosts.txt in /etc/hosts
Problem	scalability in terms of query load & speed - management, - consistency, - availability

To **scale**, DNS adopt three intertwined hierarchies:

1) naming structure: addresses are hierarchical https://www.ee.ethz.ch/de/departement/
2) management: hierarchy of authority over names
3) infrastructure: hierarchy of DNS servers

Naming structures:

- Top Level Domain (TLDs) sit at the top (.com, .org, .net, .ch, …)
- **Domains** are subtrees
- A name, e.g. ee.ethz.ch, represents a leaf-to-root path in the hierarchy:



Management:

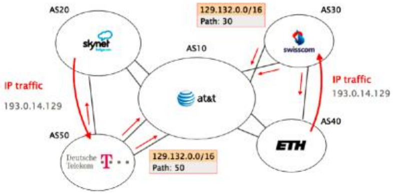
- Root servers are managed by IANA
- TDLs are managed by private and non-profit organizations → (.ch is managed by SWITCH)
- Domains are managed by ISPs or locally (company/institution) → (.ethz.ch is managed by ETH Zurich)

Infrastructure:

To scale root servers, operators rely on **BGP anycast.** Intuition:

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

Instances of the k-root server are hosted in more than 40 locations worldwide.
→ k.root-servers.org
All locations announce 193.0.14.0/23 in BGP, 193.0.14.129 being the IP of the server.



- Every server knows the addresses of the root servers (otherwise a server wouldn't be able to start a DNS request)
- Every root server knows the addresses of all TDLs
- From there on, each server knows the address of all children
- To ensure availability, each domain must have at least a primary and secondary DNS server (ensure service availability, balance DNS queries)
  - Ensure name service availability as long as one of the servers is up
  - DNS queries can be load-balanced across the replicas
  - On timeout, client use alternate servers exponential backoff when trying the same server

Overall, the DNS system is highly scalable, available, and extensible:

scalable	#names, #updates, #lookups, #users, but also administration
available	domains replicate independently of each other
extensible	any level (including the TLDs) can be modified independently

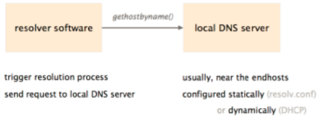
7.1.2 How do you insert into the DNS?

- You register next-startup.ch at a registrar X, e.g. Swisscom or GoDaddy
- Provide X with the name and IP of your DNS servers e.g., [ns1.next-startup.ch,129.132.19.253]
- You set-up a DNS server @129.132.19.253 define A records for www, MX records for next-startup.ch...

7.1.3 DNS queries

Using DNS relies on two components:

- Resolver software (sends request to local DNS server)
- Local DNS server (resolves request statically or dynamically)


resolver software
gethostbyname()
local DNS server
trigger resolution process
send request to local DNS server
usually, near the endhosts
configured statically (resolv.conf)
or dynamically (DHCP)

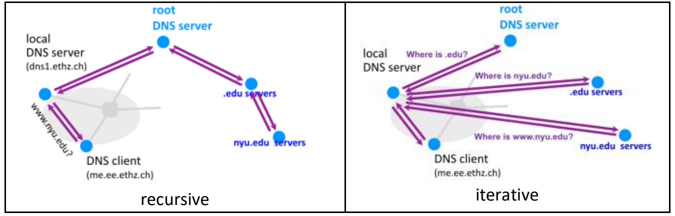
DNS query and reply uses UDP (port 53), reliability is implemented by repeating requests.

DNS server stores **resource records** (name, value, type, TTL)

Records	Name	Value
A	host name	IPv4 address
AAAA	host name	IPv6 address
NS	domain	DNS server name
MX	domain	Mail server name
CNAME	alias	canonical name
PTR	IP address	corresponding host name

7.1.3.1 DNS resolution

- **Recursive**: Clients offload task of resolving to server
- **Iterative**: Server only returns address of next server to query



➔ Today iterative queries are more common

Reducing resolution times relies on **caching** (servers, OS, browsers).
DNS records are cached for TTL seconds.

7.1.4 Caching

To reduce resolution times, DNS relies on caching

- DNS servers cache responses to former queries and your client and the applications (!)
- Authoritative servers associate a lifetime to each record Time-To-Live (TTL)
- DNS records can only be cached for TTL seconds after which they must be cleared

As top-level servers rarely change & popular website visited often, caching is very effective!

- Top 10% of names account for 70% of lookups	- Limit cache hit rate to 91%
- 9% of lookups are unique	- Practical cache hit rates ~75%

7.2 Web (HTTP, HTTPS)

The web was founded approx. in 1990 by Tim Berners-Lee. He developed the World Wide Web (WWW): A distributed database of "pages" linked together via **HTTP:**

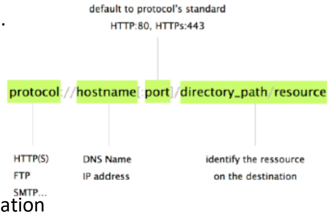
Key components of the web:

Infrastructure	Content	Implementation
- Clients/Browser - Servers - Proxies	Objects: *files*, *pictures*, *videos*, … Web sites are a collection of objects	URL: name content HTTP: transport content

7.2.1 URL

A **Uniform Resource Locator (URL)** refers to an Internet resource:

- Protocol: HTTP, HTTPS, FTP, SMTP, …
- Hostname: DNS name or IP address
- Port: Not necessary, if it is omitted the default port of the protocol standard is taken (HTTP: 80, HTTPS: 443)
- Directory_path/resource: Identifies the resource on the destination

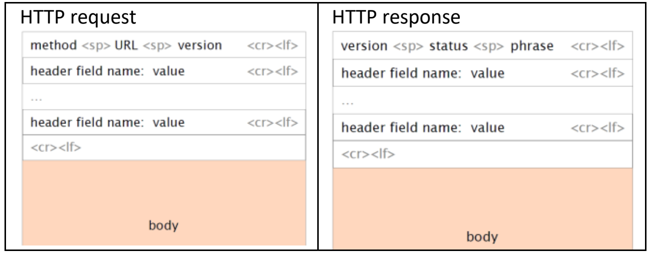


7.2.2 HTTP (Hypertext Transfer Protocol)

HTTP is a rather simple synchronous request/reply protocol:

- HTTP is layered over a bidirectional byte stream (almost always TCP)
- HTTP is text-based (ASCII) human readable, easy to reason about
- HTTP is stateless it maintains no info about past client requests (→ thats why cookies)

HTTP request/ HTTP response


HTTP request
method <sp> URL <sp> version <cr><lf>
header field name: value <cr><lf>
...
header field name: value <cr><lf>
<cr><lf>
body
HTTP response
version <sp> status <sp> phrase <cr><lf>
header field name: value <cr><lf>
...
header field name: value <cr><lf>
<cr><lf>
body

- **HTTP request**

method	GET return ressource HEAD return headers only POST send data to server (forms)
URL	Relative to server (e.g. /index.html)
version	HTTP version (1.0, 1.1 or 2.0)
headers	(variable length, contain different information): Authorization info, acceptable document types, from (user email), if-modified-since, referrer (cause of the request), user agent (client software) Header are human readable.

- **HTTP response**
  - Version: HTTP version (1.0, 1.1, 2.0)
  - Status: 3 digit response code

3 digit response code		Reason phrase	
1XX	informal		
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

  - Headers (variable length, contain different information): Location (for redirection), allow (list of sup-ported methods), content [encoding, length, type], expires (caching), last-modified (caching). Header are human readable.

HTTP is a stateless protocol, meaning each request is treated independently

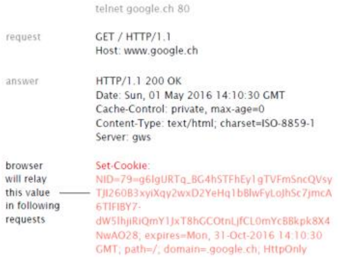
Advantages	Disadvantages
- server-side scalability - failure handling is trivial	- some applications need state! (shopping cart, user profiles, tracking)

Solution: Cookies

7.2.3 Cookies

HTTP makes the client maintain the state. This is what the cookies are for!

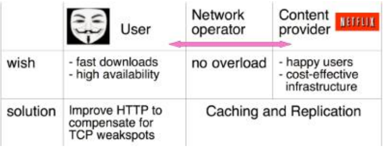
- client stores small state on behalf of the server X
- client sends state in all future requests to X
- can provide authentication



```
telnet google.ch 80

request      GET / HTTP/1.1
             Host: www.google.ch

answer       HTTP/1.1 200 OK
             Date: Sun, 01 May 2016 14:10:30 GMT
             Cache-Control: private, max-age=0
             Content-Type: text/html; charset=ISO-8859-1
             Server: gws

browser      Set-Cookie:
will relay   NID=79=g6IgURTq_BG4hSTFhEy1gTVFmSncQVsy
this value — TJI260B3xyiXqy2wxD2YeHq1bBlwFyLoJhSc7jmcA
in following 6TIFIBY7-
requests     dW5IhjiRiQmY1JxT8hGCOtnLjfCL0mYcBBkpk8X4
             NwAO28; expires=Mon, 31-Oct-2016 14:10:30
             GMT; path=/; domain=.google.ch; HttpOnly
```

7.2.4 HTTP performance

Performance goals vary depending on who you ask:

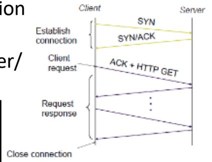


	User	Network operator	Content provider
wish	- fast downloads - high availability	no overload	- happy users - cost-effective infrastructure
solution	Improve HTTP to compensate for TCP weakspots	Caching and Replication	

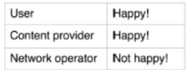
7.2.4.1 Improve HTTP to compensate for TCP weakpoints

Relying on TCP forces a HTTP client to open a connection before exchanging anything.
Most web pages have multiple objects, a naive browser/ naive HTTP opens one TCP connection for each…

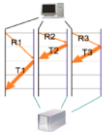
Fetching n objects requires ~2n RTTs
2n → TCP establishment HTTP request/ response



Solution #1: use multiple TCP connections in **parallel**



User	Happy!
Content provider	Happy!
Network operator	Not happy!

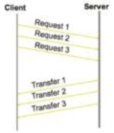


Solution #2: use persistent connections across multiple requests, default in HTTP/1.1

- Avoid overhead of connection set-up and teardown clients or servers can tear down the connection
- Allow TCP to learn more accurate RTT estimate and with it, more precise timeout value
- Allow TCP congestion window to increase and therefore to leverage higher bandwidth

Solution #3: pipeline requests & replies asynchronously, on one connection

- batch requests and responses to reduce the number of packets
- multiple requests can be packed into one TCP



Time to retrieve *n* small objects

	#RTTs
One-at-a-time	$\sim 2n$
M concurrent	$\sim 2n/M$
Persistent	$\sim n+1$
Pipelined	2

Time to retrieve *n* big objects, there is no clear winner as bandwidth matters more:

$$\#RTTs = \frac{\sim n * avg.\ file\ size}{bandwidth}$$

Average webpage size is 2.3Mb. Top websites have decreased in size though because they care about TCP performance.

7.2.5 Caching

- Highly popular content largely overlaps
- Caching reduces time to load a webpage and decreases the network and server load (just think how many times the Facebook logo is requested)
- Nonetheless many content is uncachable:

Dynamic data	Stock prices, scores, …
Scripts	Results based on parameters
Cookies	Results may be based on passed data
SSL	Cannot cache encrypted data
Advertisements	Wants to measure # of hits ($$$)

To limit staleness of cached objects, HTTP enables a client to validate cached objects:

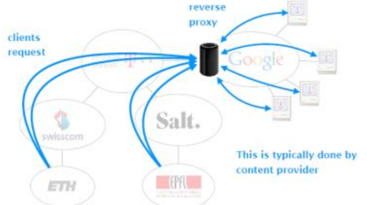
- Server hints when an object expires (kind of TTL) as well as the last modified date of an object
- Client conditionally requests a resource using the “ifmodified-since” header in the HTTP request
- Server compares this against “last modified” time of the resource and returns:
  - Not Modified if the resource has not changed
  - OK with the latest version

Caching can and is performed at different locations:

Client	**Browser cache**
Close to the client	Forward proxy Content Distribution Network (CDN)
Close to the destination	Reverse proxy

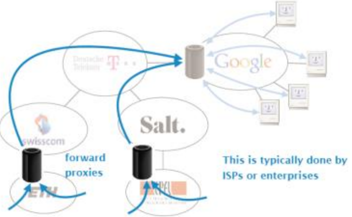
Many clients request the same information. This increases servers and network’s load, while clients experience unnecessary delays.

- **Reverse proxies** cache documents close to servers, decreasing their load



This is typically done by content provider

- **Forward proxies** cache documents close to clients, decreasing network traffic, server load and latencies



This is typically done by ISPs or enterprises

7.2.6 Replication

The idea behind replication is to duplicate popular content all around the globe:

- Spreads load on server e.g., across multiple data-centers
- Places content closer to clients only way to beat the "speed-of-light"
- Helps speeding up uncachable content still have to pull it, but from closer

The problem of CDNs is to direct and serve your requests from a close, non-overloaded replica.

Approaches:

DNS-based	BGP Anycast
Returns ≠ IP addresses based on:	Advertise the same IP prefix from different locations
- client geo-localization - server load	- avoided in practice

7.2.7 Akamai

Akamai is one of the largest CDNs in the world, boasting servers in more than 20,000 locations. Akamai uses a combination of

- pull caching *direct result of clients requests*
- push replication *when expecting high access rate*

together with some dynamic processing dynamic Web pages, transcoding,...

"Akamaizing" content is easily done by modifying content to reference the Akamai's domains

- Akamai creates domain names for each client
  - Ex: a128.g.akamai.net for cnn.com
- Client modifies its URL to refer to Akamai's domain
  - http://www.cnn.com/image-of-the-day.gif becomes http://a128.g.akamai.net/image-of-the-day.gif
- Requests are now sent to the CDN infrastructure

8 Video streaming & Email

8.1 Video streaming

Video streaming accounts for a huge share of today's internet traffic. How do we deliver high quality videos without constant delays/ buffering?

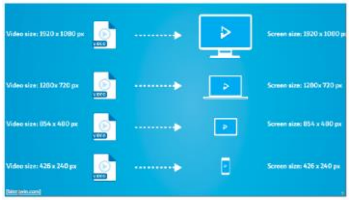
Naive approach: one-size-fits-all

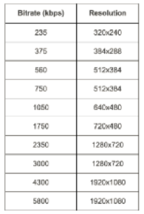
→ encode video into one progressive video file (e.g. resolution 1280 x 720), then up- and downscale the resolution, depending on the screen. But: we can't send the whole file at once. Also, if we have a 4k TV, we don't want an upscaled 1280x720 video.

In practice: three step solution

- **Encoding:** Encode video in multiple bitrates (client picks the bitrate since the client has the best view of the current network situation)
- **Replication:** Replicate using a content delivery network
- **Adaption:** Video player picks bitrate adaptively
  - Estimate connection's available bandwidth
  - Pick a bitrate ≤ available bandwidth

8.1.1 Encoding


Video size: 1920 x 1080 px
Screen size: 1920 x 1080 px
Video size: 1280x 720 px
Screen size: 1280x 720 px
Video size: 854 x 480 px
Screen size: 854 x 480 px
Video size: 426 x 240 px
Screen size: 426 x 240 px



Bitrate (kbps)	Resolution
235	320x240
375	384x288
560	512x384
750	512x384
1050	640x480
1750	720x480
2350	1280x720
3000	1280x720
4300	1920x1080
5800	1920x1080

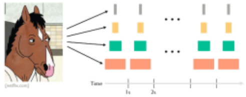
Simple solution for encoding: use a "bit ladder"

→ for what bitrate (network bandwidth) can I use what resolution

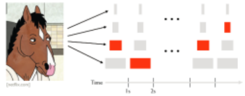
Problem of this static ladder: some movies need less bandwidth for high resolutions (e.g. cartoons or documentaries). Hollywood action movies need lots of bandwidth since there's lots of movement/ change/ effects.

Better solution: dynamically adapt the resolution

- Your player downloads "chunks" of video at different bitrates



- Depending on your network connectivity, your player fetches chunks of different quality



8.1.2 Replication

To get faster streams, video streaming companies replicate their content all over the world such that all users get high quality streams.

→ push content close to users

Map:

Orange: deployment in IXP

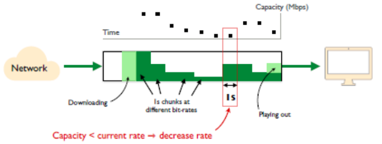
Green: deployment in ISP

8.1.3 Adaption

- The client has a buffer → this allows to be robust against network inconsistencies

Capacity (Mbps)
Time
Network

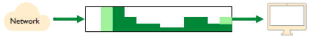
- **Adaption:** Video player picks bitrate adaptively
  - Estimate connection's available bandwidth
  - Pick a bitrate ≤ available bandwidth


Capacity (Mbps)
Time
Network
Downloading
1s chunks at different bit-rates
1s
Playing out
Capacity < current rate → decrease rate

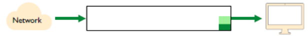
→ e.g. we have 4Mbps out (watching) and 1Mbps in (download) → too high out rate → decrease rate

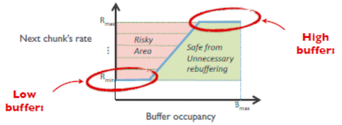
Estimating available capacity:

- Buffer-based adaption:
  - Nearly full buffer → large rate (we can afford to watch high rate, buffer takes long to empty)



  - Nearly empty buffer → small rate (buffer empties fast)





- New approaches… Machine Learning based capacity estimation … → Active research topic

Problem: startup phase? Pick rate based on immediate past throughput.

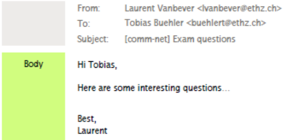
8.2 Email

Email was the first application on the internet and is still widely used. There are 3 perspectives to email: Content, Infrastructure/ Transmission and Retrieval

8.2.1 Content

An email is composed of two parts:

- A **header**, in 7-bit U.S. ASCII text
- A **body**, also in 7-bit U.S. ASCII text



From: Laurent Vanbever <lvanbever@ethz.ch>
To: Tobias Buehler <buehlert@ethz.ch>
Subject: [comm-net] Exam questions

Body

Hi Tobias,

Here are some interesting questions…

Best,
Laurent

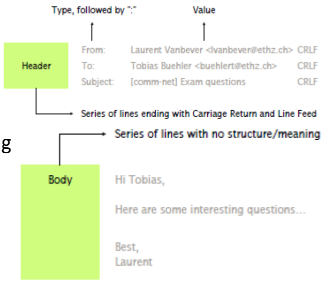
Header:

A series of lines ending with Carriage Return and Line Feed (CRLF)

Body:

Series of lines with no structure/ meaning

A blank line separates the header from The body.


Type, followed by ":"
Value
Header
Series of lines ending with Carriage Return and Line Feed
Series of lines with no structure/meaning
Body

From: Laurent Vanbever <lvanbever@ethz.ch> CRLF
To: Tobias Buehler <buehlert@ethz.ch> CRLF
Subject: [comm-net] Exam questions CRLF

Hi Tobias,

Here are some interesting questions…

Best,
Laurent

8.2.2 Multipurpose Internet Mail Extensions (MIME)

Email relies on 7-bit U.S. ASCII → how do you send non-English text? Binary files? Solution: MIME

MIME defines:

- additional headers for the email body
  - MIME-Version: the version of MIME being used
  - Content-Type: the type of data contained in the message
  - Content-Transfer-Encoding: how the data are encoded
- a set of content types and subtypes
  - image with subtypes gif or jpeg
  - text with subtypes plain, html, and rich text
  - application with subtypes postscript or msword
  - multipart with subtypes mixed or alternative: These are the most common types/subtypes for MIME:

Content type	Indicates that the msg contains
Multipart/mixed	multiple independent parts e.g. plain text and a binary file
Multipart/alternative	multiple representation of the same content e.g. plain text and HTML

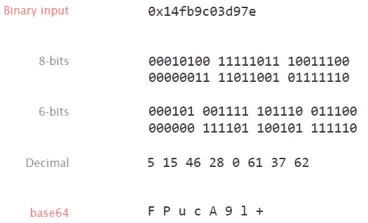
- base64 to encode binary data in ASCII: MIME relies on Base64 as binary-to-text encoding scheme

8.2.2.1 base64

Relies on 64 characters out of the 128 ASCII characters:

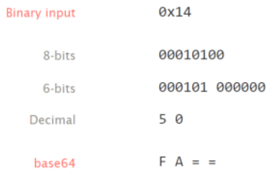
- the most common and printable ones, i.e. A-Z, a-z, 0-9, +, /

Divides the bytes to be encoded into sequences of 3 bytes each group of 3 bytes is then encoded using 4 characters

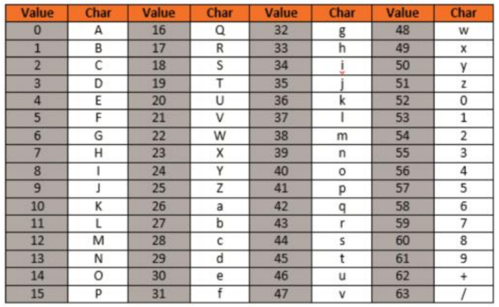


```
Binary input   0x14fb9c03d97e

8-bits         00010100 11111011 10011100
               00000011 11011001 01111110

6-bits         000101 001111 101110 011100
               000000 111101 100101 111110

Decimal        5 15 46 28 0 61 37 62

base64         F P u c A 9 1 +
```

If the length of the input is not a multiple of three, Base64 uses "=" as padding character



```
Binary input   0x14

8-bits         00010100

6-bits         000101 000000

Decimal        5 0

base64         F A = =
```



Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

We need to know where a binary file starts an ends → MIME uses boundary i.e. in the header, it specifies:

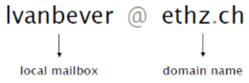
```
    Content-Type: multipart/mixed; boundary="123boundary"
```

Then, the binary file is surrounded by:

```
    --123boundary
    …
    --123boundary
```

8.2.3 Infrastructure/ Transmission

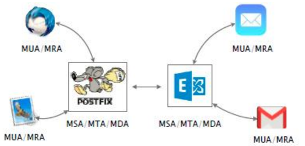
An **email address** is composed of two parts, identifying the local mailbox and the domain: The actual mail server is identified using a DNS query for a MX record.



We can divide the email infrastructure into five functions:

MUA	Mail User Agent	Use to read/write emails (mail client)
MSA	Mail Submission Agent	Process email and forward to local MTA
MTA	Mail Transmission Agent	Queues, receives, sends mail to other MTAs
MDA	Mail Delivery Agent	Deliver email to user mailbox
MRA	Mail Retrieval Agent	Fetches email from user mailbox

MSA/MTA/MDA and MRA/MUA are often packaged together leading to simpler workflows



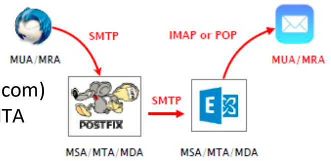
8.2.3.1 Simple Mail Transfer Protocol (SMTP)

SMTP is the current standard for transmitting e-mails.

- SMTP is a text-based, client-server protocol
  → client sends the e-mail, server receives it
- SMTP uses reliable data transfer, built on top of TCP (port 25 and 465 for SSL/TLS)
- SMTP is a push-like protocol
  → sender pushes the file to the receiving server (no pull)

	SMTP 3 digit response code			comment
Status	2XX	success	220	Service ready
			250	Requested mail action completed
	3XX	Input needed	354	Start mail input
	4XX	Transient error	421	Service not available
			450	Mailbox unavailable
			452	Insufficient space
	5XX	Permanent error	500	Syntax error
			502	Unknown command
			503	Bad sequence

Steps for a mail transmission:



1) The sender MUA uses SMTP to transmit the e-mail first to a local MTA (e.g. mail.ethz.ch, gmail.com)
2) The local MTA then looks up the MTA of the recipient domain (DNS MX) and transmits the e-mail further
3) Once the e-mail is stored at the recipient domain, IMAP or POP is used to retrieve it by the recipient MUA

- Emails typically go through at least 2 SMTP servers, but often way more:
  - Separate SMTP servers for separate functions (SPAM filtering, virus scanning, data leak prevention, etc.)
  - Separate SMTP servers that redirect messages e.g. from lvanbever@tik.ee.ethz.ch to lvanbever@ethz.ch
  - Separate SMTP servers to handle mailing-list mail is delivered to the list server and then expanded
- Each SMTP server/MTA hop adds its identity to the e-mail header by prepending a "Received" entry

8.2.3.2 Email security

As with most of the key Internet protocols, security is an afterthought

SMTP Headers:

MAIL FROM:	no checks are done to verify that the sending MTA is authorized to send e-mails on behalf of that address

Email content (DATA):

From:	no checks are done to verify that the sending system is authorized to send e-mail on behalf of that address
Reply-to:	ditto

In short, none of the addresses in an email are typically reliable

Countermeasures have been proposed with various level of success:

Example: Sender Policy Framework (SPF)

Enables a domain to explicitly authorize a set of hosts that are allowed to send emails using their domain names in "MAIL FROM".

How? using a DNS TXT resource record

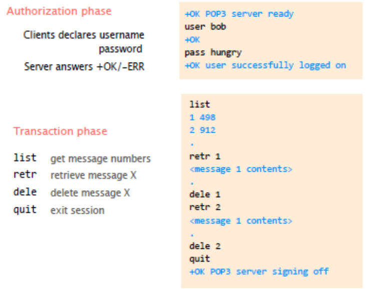
→ look for "v=spf1" in the results of "dig TXT google.com"

8.2.4 Retrieval

8.2.4.1 Post Office Protocol (POP)

POP is a simple protocol which was designed to support users with intermittent network connectivity. POP enables e-mail users to:

- retrieve e-mails locally
  → when connected
- view/manipulate e-mails
  →vwhen disconnected



Authorization phase

Clients declares username
password

Server answers +OK/–ERR

```
+OK POP3 server ready
user bob
+OK
pass hungry
+OK user successfully logged on
```

Transaction phase

list	get message numbers
retr	retrieve message X
dele	delete message X
quit	exit session

```
list
1 498
2 912
.
retr 1
<message 1 contents>
.
dele 1
retr 2
<message 1 contents>
.
dele 2
quit
+OK POP3 server signing off
```

and that's pretty much it…

POP is heavily limited. Among others, it does not go well with multiple clients or always-on connectivity:

- Cannot deal with multiple mailboxes
  → designed to put incoming emails in one folder
- Not designed to keep messages on the server
  → designed to download messages to the client
- Poor handling of multiple-client access
  → while many (most?) users have now multiple devices

8.2.4.2 Internet Message Access Protocol (IMAP)

Unlike POP, Internet Message Access Protocol (IMAP) was designed with multiple clients in mind:

- Support multiple mailboxes and searches on the server
  → client can create, rename, move mailboxes & search on server
- Access to individual MIME parts and partial fetch
  → client can download only the text content of an e-mail
- Support multiple clients connected to one mailbox
  → server keep state about each message (e.g. read, replied to)

9 Other protocols: ICMP & NAT

What should network tell host about?

- No route to destination? Host can't detect or fix routing failure.
- TTL expires? Host can't detect or fix routing loop.
- Packet too big (with DF set)? Host can adjust packet size, but can't tell difference between congestion drops and MTU drops
- Buffer overflowing? Transport congestion control can detect/deal w/ this
- Header corruption or ill-formed packets? Host can't fix corruption, but can fix formatting errors

Router response to problems?

- Router doesn't really need to respond
  - Best effort means never having to say you're sorry
  - So, IP could conceivably just silently drop packets
- Network is already trying its best
  - Routing is already trying to avoid loops/dead-ends
  - Network can't reduce packet size (in DF packets)
  - Network can't reduce load, nor fix format problems

• What more can/should it do? Error reporting!

Error reporting helps diagnosis!

- Silent failures are really hard to diagnose
- IP includes feedback mechanism for network problems, so they don't go undetected
- ICMP: The Internet "print" statement
- Runs on IP, but viewed as integral part of IP

9.1 ICMP (Internet Control Message Protocol)

ICMP can be used for diagnosis and discovery of a network. It runs on IP but is viewed as integral part of IP (IP doesn't need to report any problems, since it only tries best-effort delivery). *ICMP is the "print" statement of the internet.*

Possible problems a router might see

- Dead-end: no route to dest	- Congestion: Buffer overflow
- Sign of a loop: TTL expires	- Header corruption/ ill-formatted packet
Can't physically forward: Packets too big (Routers shouldn't fragment packets for security reasons)	

9.1.1 Operating principle of ICMP

- Triggered when IP packet encounters a problem
  - (TTL expired, destination unreachable)
- ICMP packet sent back to source IP address:
  - Includes the error information (e.g. type and code)
  - IP header plus 8+ byte excerpt from the original packet
- Source host receives the ICMP packet:
  - Inspects excerpt (e.g. protocol/ports) to identify socket
- Exceptions: No ICMP packets are sent:
  - if the problem packet itself was an ICMP packet
  - for all fragments of a fragmented packet except fragment 0

9.1.2 Types of ICMP messages

- Needs fragmentation (if IP packet too large for link layer)
- TTL expired (decremented at each hop; generated if TTL=0)
- Unreachable (Subtypes: network / host / port)
- Source Quench (Asks the sender to slow down)
- Redirect (Tells the source to use a different local router)

9.1.3 Discovering network path properties with ICMP

9.1.3.1 Ping: Echo and Reply

ICMP includes simple "echo" functionality

- Sending node sends an ICMP Echo Request message
- Receiving node sends an ICMP Echo Reply

Ping tool

- Tests connectivity with a remote host
- ... by sending regularly spaced Echo Request
- ... and measuring delay until receiving replies

9.1.3.2 Path MTU Discovery

MTU = Maximum Transmission Unit (Largest IP packet that a link supports)
PMTU = Path MTU (minimal end-to-end MTU (largest possible packet))
→ must keep datagrams no larger to avoid fragmentation
How to find the PMTU of a connection:

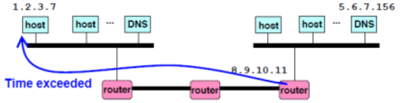
- Try a desired value (4352 byte (FDDI), 1500 byte (Ethernet), 1480 byte (IP-in-IP over Ethernet), …)
- Set DF flag (Don't Fragment) to prevent fragmentation
- When "Need Fragmentation" ICMP reduce packet size and try again

Issues with Path MTU discovery

- What if the PMTU changes? (how could it?)
  - Sender will immediately see reductions in PMTU
  - Sender can periodically try larger values
- What if Needs Fragmentation ICMP is lost?
  - Retransmission will elicit another one
- if routers don't send ICMP messages this approach doesn't work ("PMTU Black Holes")

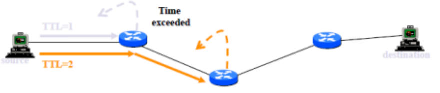
9.1.3.3 Discovering routing via time exceeded

- Host sends an IP packet
  - Each router decrements the time-to-live field
- If TTL reaches 0
  - Router sends Time Exceeded ICMP back to the source
  - Message identifies router sending it
    - Since ICMP is sent using IP, it's just the IP source address
    - And can use PTR record to find name of router



Traceroute: Exploiting Time Exceeded

- Time-To-Live field in IP packet header
  - Source sends a packet with TTL ranging from 1 to n
  - Each router along the path decrements the TTL
  - Nodes send "TTL expired" ICMP when TTL reaches 0
  - Sender receives the ICMPs of the different nodes and can reconstruct the path
- Traceroute tool exploits this TTL behavior



Send packets with TTL=1, 2, …
and record source of *Time Exceeded* message

9.2 Network Address Translation (NAT)

9.2.1 Sharing Single Address Across Hosts

Network Address Translation (NAT) enables many hosts to share a single address

- Uses port numbers (fields in transport layer)

Was thought to be an architectural abomination when first proposed, but it:

- Probably saved us from address exhaustion
- And reflects a modern design paradigm (indirection)

Before NAT ...every machine connected to Internet had a unique IP address!

9.2.2 Special-purpose address blocks of NAT

- Private addresses:
  - By agreement, not routed to the public internet
  - Blocks: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16
- Limited broadcast:
  - Sent to every host attached to the local network
  - Block: 255.255.255.255/32
- Loopback:
  - Address blocks that refer to the local machine
  - Block: 127.0.0.0/8 (usually only 127.0.0.1/32 is used)
- Link-local:
  - Not forwarded by any router
  - Used for single link communication → autoconfiguration
  - Block: 169.254.0.0/16

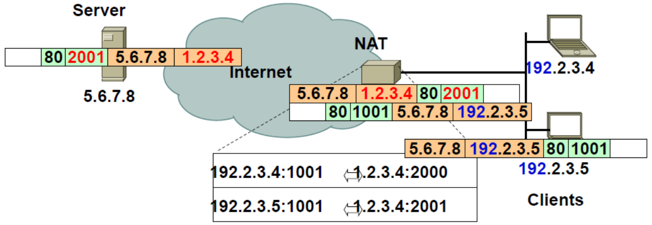
9.2.3 Operating principles of NAT

- Assign addresses to machines behind same NAT
  - Can be any private address range (e.g. 192.168.0.0/16)
- Use port numbers to multiplex single address

Example: Content of packet headers (Network with IP address 1.2.3.4):

Device	dst addr	src addr	dst port	src port
User 1	5.1.1.1	192.2.3.4	80	1001
→ NAT	5.1.1.1	1.2.3.4	80	2000
User 2	6.1.1.1	192.2.3.5	80	1001
→ NAT	6.1.1.1	1.2.3.4	80	2001

Afterwards, the destination servers respond with a packet with the src address and src port. Then, the NAT forwards these packets to the corresponding users using the src ports (200X).



NAT: Early Example of "Middlebox"

- Boxes stuck into network to delivery functionality
  - NATs, Firewalls,....
- Don't fit into architecture, violate the end-to-end principle
- But a very handy way to inject functionality that:
  - Does not require end host changes or cooperation
  - Is under operator control (e.g., security)
- An interesting architectural challenge:
  - How to incorporate middleboxes into architecture

9.2.4 Problems with NAT

A host in a NAT can't receive request traffic (i.e. an external client wanting to connect to a server *in* a NAT) since the external client doesn't know the port of the server.
Solution: keep track of a state (Port forwarding!)

10 Network Security

Basic network security properties:

- Confidentiality (= Geheimhaltung): keep data secret
- Authenticity: identification and assurance of origin
- Integrity: preventing unauthorized changes
- Availability
- Non-repudiation (= Nachweisbarkeit): prove that somebody sent or received a message
- Access control: who is allowed to access what resources

Possible threats in the internet:

- Eavesdropper: Listening to conversations (confidentiality)
- Man-in-the-middle: Modifying content (integrity)
- Impersonation: Fake website (authentication, confidentiality)

10.1 Public Key Cryptography (asymmetric encryption)

Person A has a public and a private key to encrypt/decrypt data. Everything encrypted with the public key can only be decrypted with the private key and vice versa. The public key gets published everywhere whereas only person A knows the private key.

- Confidentiality: Anyone can encrypt data with the public key and only person A decrypt it again.
- Authenticity: Person A can encrypt data with his private key and publish this data somewhere. Anyone can decrypt the data with the public key. If the decryption is successful, this shows that indeed person A has sent the data (since only person A knows the private key).
- Integrity: If both Person A and B have a public & private key pair, they can communicate securely. Person A encrypts the data twice, first with private key A then with public key B. Hence, only person B can decrypt the data and can be sure that indeed Person A sent it. Additionally, this procedure ensures that nobody modified the data since this would require the private key of person B.

10.2 HTTP Security

- HTTPS = Hypertext Transfer Protocol Secure
- HTTP sitting on top of secure channel (TLS)
- Uses TCP Port 443 (normal HTTP uses TCP Port 80)
- All HTTP bytes are encrypted and authenticated (no change in the basic functionality of HTTP). User encrypts with public key and server encrypts data with private key (see above "Public Key Cryptography").
- Hierarchical public key infrastructure: Ensures the binding between identities (e.g. domain names) and public keys.

10.3 Transport Layer Security (TLS)

- Based on the earlier Secure Socket Layer (SSL)
- **TLS Handshake** (asymmetric encryption – slow)
  - Client requests TLS connection (TLS Hello)
  - Server response with TLS certificate (including public key)
  - Client validates the certificate
  - Client generates symmetric session key, encrypts it with public key and sends it to server
- **TLS Records** (symmetric encryption – fast)
  - Messages get fragmented, integrity-protected (HMAC) and encrypted using the session key
  - Encrypted messages are passed to transport layer (TCP)
  - To prevent replays/reordering, HMAC includes seqnos

10.4 IP Security (IPsec)

- General IP security framework (specification quite complex)
  → Mandatory support in IPv6, optional in IPv4
- Provides security below transport layer. However, IPsec is less used than TLS
- Can provide: Access control, integrity, authentication, originality and confidentiality
- Applicable to narrow streams (specific TCP connections) or wide streams (all packets between two gateways)
- Resistant to bypass (e.g. included in router/firewall)
- Transparent to applications and sometimes to end users
- Different IPsec protocols:
  - Authentication Header (AH):
    - Origin authentication (MAC over most header fields) (MAC = Message Authentication Code)
    - Anti-replay protection
    - Incompatible with NAT (since header encrypted)
  - Encapsulating Security Protocol (ESP):
    - Transport mode: Data encrypted, but not header (headers are needed for routing)
    - Tunnel mode: Encrypts entire IP packet (add new header for next hop). Used in VPNs.

10.5 DNS Security (DNSSEC)

- Attacks on DNS:
  - **Denial-of-Service (DoS)** attacks using DNS as amplifier:
    Certain DNS servers reply to short requests (60 bytes) with long responses (3000 bytes). Therefore, the data rate is amplified by a factor of 50. An attacker uses this, by sending many requests and changing the source IP to the DoS target (IP spoofing). Hence, already a 100 Mbps stream from the attacker yields to a 5 Gbps traffic at the target and thus overloading his network.
    Countermeasures: Prevent IP spoofing or disable open amplifiers.
  - **Cache poisoning**: Client wants to resolve a name (e.g. www.evil.com) into an IP address. The name server of www.evil.com returns the requested IP and additionally returns a fake IP address for any other site (e.g. www.ubs.com is at IP 66.66.66.66).
    Countermeasures: Client remembers the requested domain name and adds 16-bit request ID (to demux UDP response)
  - **DNS hijacking**: With 16 bits there are only 65536 possible IDs. Already with a decent data rates (approx. 32 Mbps) these values can quickly be listed.
    Countermeasure: Randomize DNS source port
- DNSSEC:
  - Protects against data spoofing and corruption
    - Provides mechanisms to authenticate servers and requests
    - Provides mechanisms to establish authenticity and integrity

- PK-DNSSEC (Public Key):
  - DNS servers sign IP addresses with their private key. Therefore, public keys can be used to verify the source.
  - Authenticity of public key ensured by hierarchy:
    Example: client request for www.cnn.com:

Request	Response (Enc = private key encrypted)
1. .(root)	Enc(IP of .com server & its public key)
2. .com	Enc(IP of cnn.com server & its public key)
3. cnn.com	Enc(IP of www.cnn.com host)

→ The public keys of the root servers are needed to decrypt the initial response. One has to trust the root servers.

11 BGP Security

Problems with BGP:

- Prefix hijacking: an AS originates a wrong prefix
  (BGP does not verify that an AS is authorized)
- Registries of prefix ownership are inaccurate

Prefix Hijacking:

- The hijacking AS (hAS) advertises the same IP as its chosen victim AS (vAS). This attracts only part of the traffic since BGP policies will choose the shortest path to the vAS.
  - Blackhole: data traffic is discarded at hAS (easy to detect…)
  - Snooping: data traffic is inspected, then redirected to vAS
    → Only longer delays, difficult to detect. Run traceroutes…
  - Impersonation: traffic is sent to bogus destinations

  May be hard to detect since vAS may not see any problem. Additionally, these attacks may not cause loss of connectivity.
- Sub-Prefix Hijacking:
  The hijacking AS advertises a more specific prefix than the victim AS. Hence, every AS picks the bogus route (traffic always follows the most specific IP). This is however easy to detect.
- Requirements for hijacking:
  - Hijacking AS needs a router with BGP sessions which is configured to originate prefixes.
  - The bad guy needs access to the router (either internal operator or outsider breaks into the AS). Also configuration mistakes are possible.
  - Other ASes need to believe bogus routes (no filter installed)
- Example: Spam emails
  - From TCP connection to mail server
  - Possible source IP addresses a spammer might use:
  - His own IP address → Easy to trace back
  - Hijack someone's IP → return traffic might not be received
  - Hijack unused (unallocated) IP address blocks and temporarily use them to send spam → best solution

Bogus BGP AS paths

- Remove ASes from the AS path
  - Example: AS 701 turns "701 3715 88" into "701 88"
  - Motivations:
    - Attract sources that try to avoid AS 3715
    - Small AS 88 seems to be closer to center
  - Who can detect this AS path modification?
  - AS 88 (and AS 701), only they know their network structure
- Add ASes to the AS path
  - Example: AS 701 turns "701 88" into "701 3715 88"
  - Motivations: ▪ Trigger loop detection in AS 3715
  - ▪ AS looks as if it had a richer connectivity
  - Who can detect this AS path modification?
  - AS 3715 (if it "sees" the route), AS 88 (does it care?), AS 701
- Add AS hop(s) at the end of the AS path
  - Example: AS 701 turns "701 88" into "701 88 3" and advertises the IP of AS 3
  - Motivations: ▪ Evade detection of wrong IP advertisement
    (since AS 3 lies in the path…)
  - Who can detect this AS path modification?
    Hard to detect, filter based on prefix ownership don't work
- Invalid AS paths
  - Example: Violated BGP policies, e.g. customer that exports routes from provider to provider
  - Consequences: Provider directs all traffic through customer
  - Main defense: Filter routes based on prefixes and AS path
- Missing/inconsistent routes
  - Policy: Prefixes are advertised to all peers, all advertisements have the same AS path length
  - Reasons for violation: advertise longer paths to neighbors to make them keep the traffic inside their own network
  - Main defense: Analyzing BGP updates/traffic

BGP Security Today

- Securing the session (authentication, encryption)
- Filtering routes by prefix and AS path
- Filtering packets to block unexpected control traffic

➔ Security depends on proper application of these rules

➔ The fundamental problems are not addressed
(Who owns an IP? AS path valid? Do packets follow a route?)

Possible enhancements to BGP security

- Secure BGP (S-BGP or BGPsec)
  - Public & private key signatures: AS1 signs its advertisements with private key 1 such that the other ASes can verify with the public key 1 that it indeed originated from AS1. If AS2 further propagates the advertisements of AS1, it signs it a second time with private key 2. This way a chain of signa-tures is generate and the entire path can be reconstructed.
    → AS paths can be validated and can't be modified
  - Requirements for S-BGP:
    - Complete, accurate registries of prefix owners (ASes)
    - Public key infrastructure (hierarchical)
    - Routers which can do cryptographic operations quickly
      → Very expensive and difficult to deploy incrementally
- Detecting and avoiding suspicious routes
  - Monitoring BGP update messages
  - Use out-of-band detection mechanisms to generate reports and alerts (some websites provide such an analysis of BGP)
  - Soft response to suspicious routes (prefer routes that agree with the past, delay unfamiliar routes when possible)
  - → This way some attacks go away on their own and
  - network operators have time to investigate.
  - If the largest 40 ASes applied this technique, most other ASes would be protected as well!
- Solution needs to be incrementally deployable
  This means, that there can't be a "flag day" where all ASes switch at once to a secure BGP variant. Therefore:
  - Backwards compatibility needed
  - Incentive (Anreiz) needed for early adopters
    (else nobody will start to use the secure BGP…)

Data plane attacks

Even if everything is fine on the control plane (no changes to AS paths, correct advertisements) there can still be security flaws on the data plane (just forward a part of a traffic out of another physical interface of the router…)

- Drop packets in the data plane: if only some packets are dropped this is really hard to detect (could be congestion)
- Send packets in a different direction:
  - Impersonate the legitimate destination
  - Snoop on traffic and forward it to real destination
    → End-to-end check needed (e.g. encryption) to protect

However, data plane attacks are hard to perform. The bad guy needs access to a router along the path. But there is also quite some motivation, e.g. if the evil internet telephone company (EITC) slows down all Skype traffic users will probably start using EITC instead of Skype because it seems to be faster.

Why are these security problems not a bigger deal?

Most outages are configuration errors (there are not so many bad guys…). Additionally, the bad guys want the internet to stay up. Hence, they only make minor changes that don't affect the whole internet.

Why do these problems still exist?

The internet is extremely complex, hence all ASes would need to cooperate. It's hard to agree on the right solution and even if there was an agreement, every AS would need to apply this solution.
BGP is still very vulnerable!