Disclaimer

This summary is part of the lecture “ETH Communication Networks” by Prof. Dr. Laurent Vanbever (FS19). It is based on the lecture.

Please report errors to doberm@student.ethz.ch such that others can benefit as well.

The upstream repository can be found at https://github.com/noah95/formulasheets

Communication Networks

Marco Dober

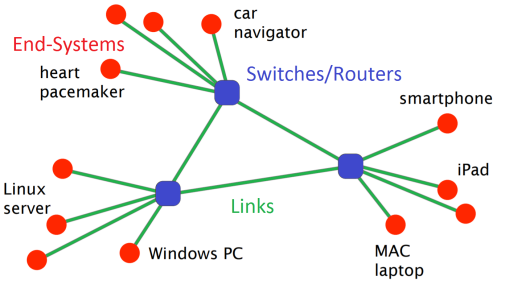
19th February 2021

1 Overview

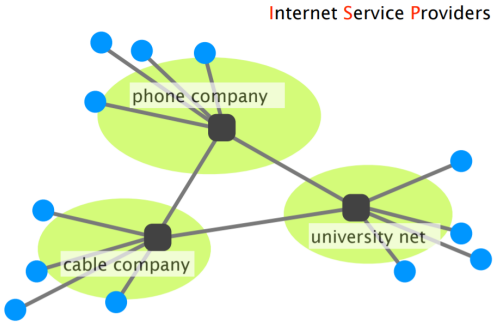
1.1 What is a network made of?

Networks are composed of three basic components:

- **End-systems** | send & receive data | PC, Server, Smartphone, car navigation
- **Switches/Routers** | forward data to destination | vary in size and usage (home to data center)
- **Links** | connect end-systems to switches and switches to each other | copper, wireless, optical-fiber



The internet is a network of networks. The Internet Service Providers (ISP) provide internet to their customers.



There exists a huge amount of **access technologies:**

- **Ethernet** | most common, symmetric (Up- and Down-stream same bandwidth)
- **DSL** | phone lines, asymmetric (Up- and Down-stream NOT same bandwidth)
- **CATV** | via cable TV, shared
- **Cellular** | Smart phones
- **Satellite** | remote areas
- **FTTH** | fiber to the home
- **Fibers** | Internet backbone
- **Infiniband** | High performance computing

1.2 How is it shared?

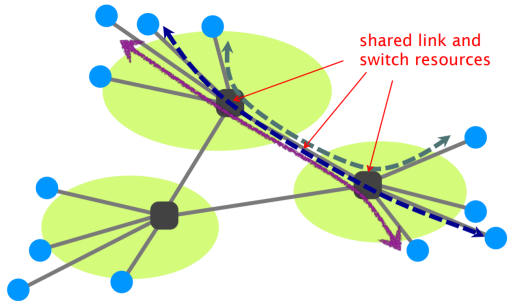
So far we discussed the "last mile" of the Internet. 3 must-have **requirements** of a good network topology:

- **Tolerate failure** | several path between src and dst.
- **Sharing to be feasible (praktikabel) & cost effective** | not too much links
- **Adequate per-node capacity** | not to few links

The Design of the Internet is a mix of full-mesh, chain and bus which is an optimization of the above requirements. This topology is called a **switched network**.

- Advantages:
  - Sharing and per-node capacity can be adapted to fit the network needs.
- Disadvantages:
  - Require smart devices to perform; forwarding, routing, resource allocation (Zuweisung)

In a switched network links and switches are shared between flows.

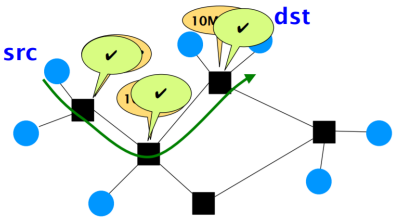


There exist two approaches of sharing, both are examples of statistical multiplexing:

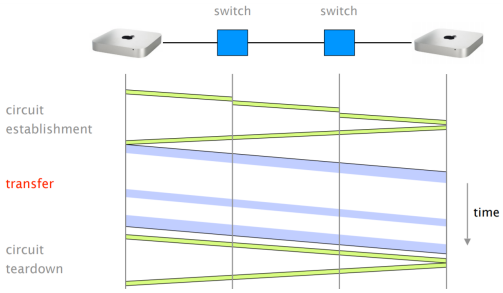
- **Reservation**
  principle: reserve needed bandwidth in advance
  multiplexing: at the flow-level
  implementation: **circuit-switching**
- **On-demand**
  principle: send data when you need
  multiplexing: at the packet-level
  implementation: **packet-switching**

Circuit-Switching:

- Relies on the Resource Reservation Protocol.
- The efficiency depends on how utilized the circuit is once established. The circuit can be mostly idle or just be used for a small amount of time (bad).
- It doesn't route around trouble



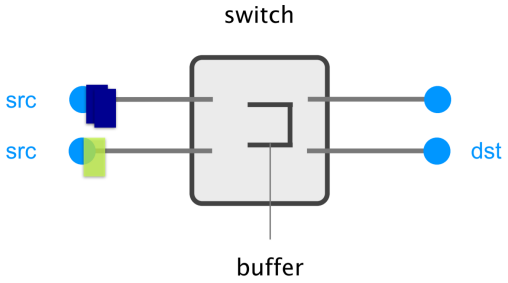
(1) src sends a reservation request for 10Mbps to dst
(2) switches "establish a circuit"
(3) src starts sending data
(4) src sends a "teardown circuit" message



- Advantages:
  - Predictable performance
  - Simple & fast switching (once circuit established)
- Disadvantages:
  - Inefficient if traffic is bursty or short
  - Complex circuit setup/teardown (adds delay to transfer)
  - Requires new circuit upon failure

Packet-Switching:

- Data transfer is done using independent packets
- Since packets are not coordinated, they can clash with each other
- To absorb transient overload, packet switching relies on buffers
- It routes around trouble on the fly



- Advantages:
  - Efficient use of resources
  - Simpler to implement
  - Route around trouble
- Disadvantages:
  - unpredictable performance
  - Requires buffer management and congestion (Stau) Control

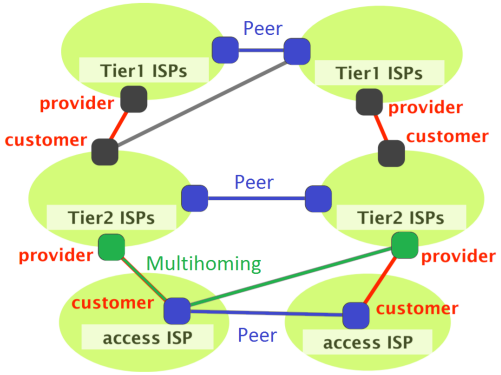
Packet-switching beats circuit-switching with respect to **resiliency** (robustness) and **efficiency**.

Internet ♥ packets

1.3 How is it organized?

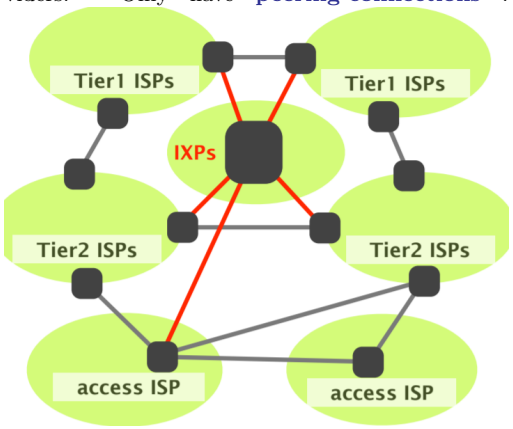
The Internet has a hierarchical structure and consists of about 60'000 networks:

- **Tier-1** (international)
  - have no provider
  - ≈12 networks
- **Tier-2** (national)
  - provide transit to Tier-3s
  - have at least one provider
  - ≈1'000s networks
- **Tier-3** (local)
  - do not provide any transit
  - have at least one provider
  - 85-90%



Some networks have an incentive (Anreiz) to connect directly, to reduce their bill with their own provider (direct traffic flow between them). This is known as **peering**

IXPs (Internet Exchange Points): provide Internet connection for Tier2 and other providers. Only have **peering-connections** .



1.4 How does communication happen?

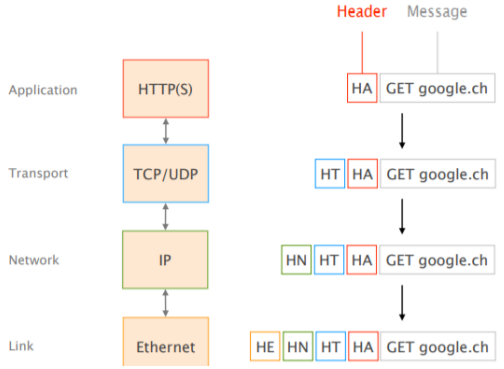
Use **protocols** to enable communication between processes in different networks. Protocols are like a conversation convention. There are thousands of different protocols. Subdivide in different **layers** to keep stuff simple (Modularity).

5 Layer Model

Layer	service provided	role	protocol
L5 Application	network access	exchanges **messages** btw. proc.	HTTP, SMTP, FTP, SIP, ...
L4 Transport	end-to-end delivery	transport **segments** btw. end-sys.	TCP, UDP, SCTP
L3 Network	global best-effort delivery	move **packets** around the network	IP
L2 Link	local best effort delivery	move **frames** across a link	Ethernet, Wifi, DSL, LTE,...
L1 Physical	physical transfer bits	move **bits** across medium	copper, fiber, coax, ...

Each layer provides a service to the layer above by using the layer below. Physical is foundation and everything is then built on top. Each layer has a **unit of data** and is implemented with different protocols and technologies (HW/SW). We can see shift to more HW because of speed.

Each layer takes message from above and encapsulates with its own **header** and/or **trailer**.



- **Switches** act as a **L2 gateway**
- **Routers** act as a **L3 gateway**

1.5 How do we characterize the network?

We characterize the network with:

- **Delay**
- **Loss**
- **Throughput**

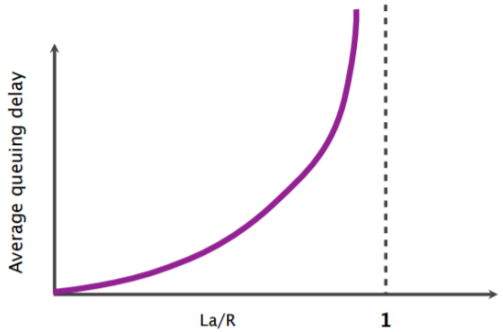
Delay
→ transmission | link property
→ propagation | link property
→ processing | traffic | mostly tiny
→ queuing | traffic | hardest to evaluate
  – arrival rate at the queue
  – transmission rate of outgoing link
  – traffic burstiness

$$\text{traffic intensity} = \frac{L \cdot a}{R}$$

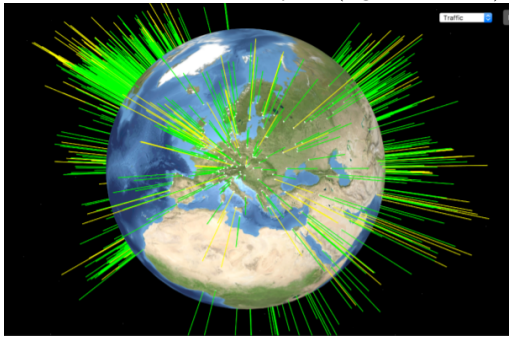
a = average packet arrival rate [packet/sec]
R = transmission rate of outgoing link [bit/sec]
L = fixed packet length [bit]

Loss
If the buffer of a queue is full, it drops packets and hence the packets are lost.

Throughput
To compute throughput one has to consider the bottleneck link



As technology improves , throughput increases & delays are getting lower, except for propagation → content delivery networks move content closer to you (e.g. akamai).



2 Concepts

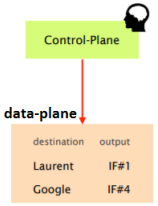
2.1 Routing

How do you guide **IP packet**s from a source to a destination?
Like an envelope, packets have a header and a payload.

Routers forward IP packets **hop-by-hop**. Routing is mostly not symmetrical (to/back not the same). Routers locally look up their **forwarding table** to know where to send the packet. Forwarding decisions necessarily depend on the **destination**, but also can depend on others (source, input port).
In addition to data-plane routers also have a control plane consisting of:

- Routing
- Configuration
- Statistics
- ...

Routing is the control-plane process that **computes** and **populates** the forwarding tables.



Forwarding vs. Routing

	forwarding	routing
goal	directing packet to an outgoing link	computing the paths packets will follow
scope	local	network-wide
implem.	hardware usually	software always
timescale	nanoseconds	10s of ms hopefully

A global forwarding state is valid if and only if:

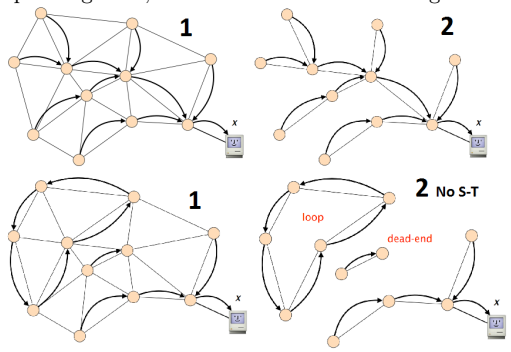
- No dead ends
- No loops

2.1.1 Verifying that a forwarding state is valid

It's easy to verify that a routing state is valid.
Simple algorithm:

1. Mark all outgoing ports with an arrow
2. Eliminate all link with no arrow
3. Sate is valid iff the remaining graph is a **spanning-tree**

See the following pictures for an example with a resulting spanning tree and one with no spanning tree, hence no valid forwarding state.



2.1.2 How to compute forwarding states

Producing valid routing state is harder → prevent dead ends (easy) & loops (hard). Prevent loops is the hard part, this is where routing protocols differ. There are three ways to compute valid routing state:

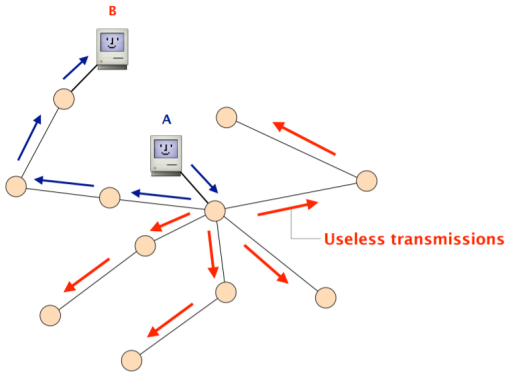
1. Use tree-like topologies | **Spanning-tree**
2. Rely on a global network view | **Link-state**
3. Rely on distributed computation | **Distance vector**

In the Internet we use 3., because it is not possible to make precise map of whole Internet.
In Networks we use 2.
Inside (part of) Networks we use 1.

1. Spanning Tree
The easiest way to avoid loops is to route traffic in a loop free topology (Sherlock). Simple algorithm:

1. Take an arbitrary topology
2. Build a spanning tree and ignore all other links
3. Done!

It works, because spanning trees only have one path between any two nodes. There are numerous tress for a topology and they vary in efficiency.
Once we have an spanning tree, forwarding is easy → just **flood** the packets everywhere (see picture below). This is very **inefficient**.



Solution: Nodes can **learn** how to reach nodes by remembering where packets came from. Ethernet works just like that. Learning is **topology-dependent!**
Routing by flooding on a spanning tree (in a nutshell):

- Flood first packet to node you're trying to reach
  → all switches learn where you are
- When destination answers, some switches learn where it is
  → some because packet to you is not flooded anymore
- The decision to flood or not is done on each switch
  → depending on who has communicated before

- Advantages:
  - Plug-and-Play (no config. needed)
  - Automatically adapts to moving host
- Disadvantages:
  - Mandate a spanning tree (eliminate many links form the topology)
  - Slow to react to failures / host movements

2. Link-state
If **each routers** knows the entire graph, it can **locally** compute paths to all other nodes. Once a node u knows the entire topology, it can compute shortest paths using **Dijkstra's algorithm**:

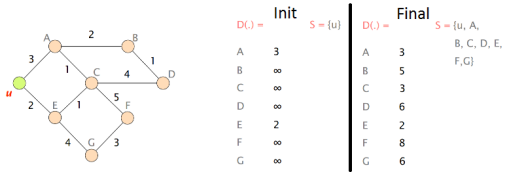
Initialization

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

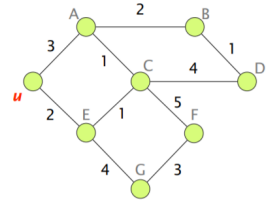
Loop

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

u is the node running the algorithm
$c(u, v)$ is the weight of the link connecting u and v.
$D(v)$ is the smallest distance currently known by u to reach v.



Normally the algorithm has $\mathcal{O}(n^2)$ complexity (n being the number of nodes), but with the help of a heap (data-structure...) the complexity can be brought down to $\mathcal{O}(n \log n)$, which is really **efficient**.
From the shortest paths, u can directly compute its forwarding table!



Forwarding table

destination	next-hop
A	A
B	A
C	E
D	A
E	E
F	E
G	E

How do we know cost?
Initally, routers only knwo their ID and their neighbors and the cost to reach them. They then build message known as Link-state (with neighbors and their cost/weight) and flood it in the network
→ At the end of the flooding process everyone should have the exact same view of the network.
I can configure wight of links **static** by hand (Dijkstra will converge), or **dynamic** (may be problematic).

3.Distance-vector
Paths can be computed in distributed computation.
Let $d_x(y)$ be the cost of the least-cost path known by x to reach y.
Each node bundles these distances into one message (vector) that it **repeatedly** sends to all its neighbors.
Each node updates its distances based on neighbors vectors:
$d_x(y) = \min\{c(x, v) + d_v(y)\}$, overall neighbors v.
This leads to a recursive computation of the shortest path, the result must be the same as Dijkstra algorithm! Example: Compute shortest path from u to D:
$d_u(D) = \min\{c(u, A) + d_A(D), c(u, E) + d_E(D)\}$
↓
$d_A(D) = \min\{c(A, B)+d_B(D), c(A, C)+d_C(D)\} = \min\{2 + 1, 1 + 4\} = 3$
↓
$d_E(D) = \min\{c(E, C) + d_C(D), c(E, G) + d_G(D), c(E, u) + d_u(D)\} = 5$
↓

$d_u(D) = \min\{3 + d_A(D), 2 + d_E(D)\} = 6$
As before u can directyl infer its forwarding table, by directing the traffic to its best neighbor (the one which advertises the smallest cost). Evaluating the complexity of DV is harder.

2.2 Reliable delivery

How do you ensure reliable transport on top of best-effort delivery?
Goals:

- Keep the network simple,dumb
  → make it easy to build/operate network
- Keep application as network unaware as possible
  → Developer should focus on app, not network

Design:

- Implement reliability in-between, in the networking stack
  → relive the burden from both the app and network

The Internet puts **reliability in L4**, just above the network layer.
What can the mean Internet do to our IP-packets:

- Lost or delayed
- Corruption
- Reordering
- Duplication

We have four goals of reliable transfer:

- **Correctness:** ensure data is delivered, in order, and untouched
- **Timeliness:** minimize time until data is transferred
- **Efficiency:** optimal use of bandwidth
- **Fairness:** play well with concurrent communications

Correctness:
A reliable transport design is correct iff:
A packet is always resent if the previous packet was lost or corrupted. A packet may be resent at other times.
Note: It is ok to give up after a while, but it must be announced to the application.

Designing a correct, timely, efficient transport mechanism knowing that packets can get lost (focus on mentioned aspects):

```
for word in list:
    send_packet(word);
    set_timer();

    upon timer going off:
        if no ACK received:
            send_packet(word);
            reset_timer();
    upon ACK:
        pass
```

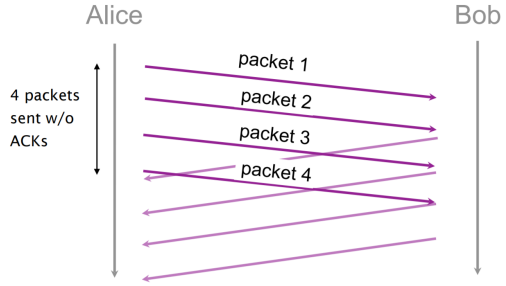
```
receive_packet(p);
if check(p.payload) == p.checksum:
    send_ack();

    if word not delivered:
        deliver_word(word);
else:
    pass;
```

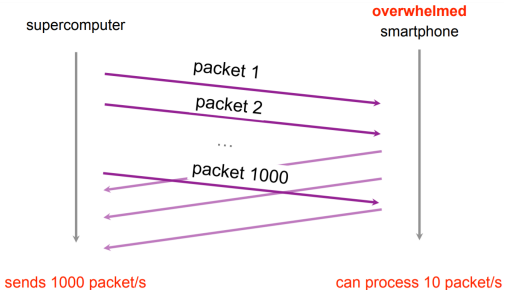
There is a clear tradeoff between timeliness and efficiency in the selection of the timeout value. Big challenge to choose optimal value.
Small timers: Risk of **unnecessary retransmissions**
Large timers: Risk of **slow transmission**

To improve timeliness just send multiple packets at the same time and not wait to ACK every packet.
Approach:

- Add sequence number to every packet
- Add buffers to sender and receiver:
  - Sender: store packets sent & not acknowledged
  - Receiver: store out-of-order packets received



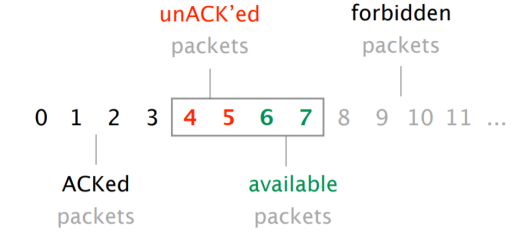
One problem that can occur is:



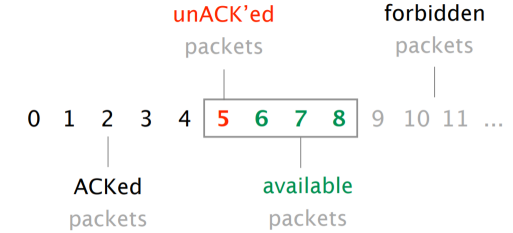
To solve this issue we need a mechanism for **flow control**. Using a **sliding window** is one way to do that:

- Sender keeps a list of the sequence # it can send.
  → known as the *sending window*
- Receiver keeps a list of acceptable sequence #
  → known as the *receiving window*
- Sender and receiver negotiate the window size
  → sending window ≤ receiving window

Example with a window-size of 4 packets:



Window after sender receives ACK 4



Timeliness of the window protocol depends on the size of the sending window.

ACKing individual packets
- Advantages:
  - Know fate of each packet
  - Simple window algorithm
  - Not sensitive to reordering
- Disadvantages:
  - Unnecessary retransmission upon losses

Cumulative ACKs
ACK the highest sequence number for which all the previous packets have been received.
- Advantages:
  - Recover from lost ACKs
- Disadvantages:
  - Causes unnecessary retransmissions
  - Confused by reordering
  - Incomplete information about which packets have arrived

Full information feedback
List all packets that have been received highest cumulative ACK, plus any additional packets
- Advantages:
  - Complete information
  - Resilient (belastbar) form of individual ACKs
- Disadvantages:
  - Overhead

With individual ACKs and full information detection of a missing packet is easy (implicit/explicit).
With cumulative ACKs missing packets are harder to know.

Fairness:
Fair is mostly not efficient. Defining what fair is, is not easy. What matters is to **avoid starvation**. Equal-per-flow is good enough for this. Simply dividing available bandwidth doesn't work.
We want to give users with small demands what they want and evenly distribute the rest. **max-min fair allocation** is such that: the lowest demand is maximized → the second lowest is maximized → the third lowest is maximized and so on...
max-min fair allocation:

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

Max-min fair allocation can be approximated by increasing window until a loss is detected.

Corruption:
Dealing with corruption is easy: Rely on a checksum and treat corrupted packets as lost ones.

Reordering:
Effect depends on ACKing mechanism which is used:
- Individual ACK: no problem
- Full feedback: no problem
- Cumm. ACKs: Create duplicate ACKs.

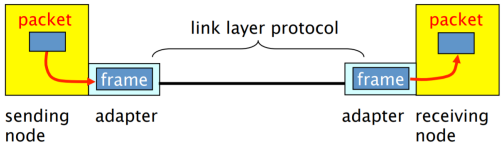
Duplicates:
Can lead to duplicated ACKs whose effect depends on the ACKing mechanism:
- Individual ACK: no problem
- Full feedback: no problem
- Cumm. ACKs: problematic

Delay:
Can create useless timeouts for all designs. It is hard to deal with the different delays which occur over the whole network. How do I set the right amount of timeout?

3 The Link Layer

3.1 What is a link?

Link = Medium + Adapter
Network adapters communicate together through the medium.
In the link layer we talk about **Frames** which are sent form one adapter to the other.



Sender
- Encapsulate packets in a frame
- add error checking bits, flow control,...

Receiver:
- Look for errors, flow control,...
- Extract packet and passes it to network layer

Link-Layer provides a best effort delivery service to the network layer, composed od **5** sub services.

- **Encoding** | represents the 0's and 1's
- **Framing** | encapsulates packet into a frame (header/trailer)
- **Error detection** | detects errors with checksum
- **error correction** | optionally correct errors
- **Flow Control** | pace sending and receiving node

3.2 How do we identify link adapters?

Medium **A**ccess **C**ontrol address:
- **Identify the sender & receiver adapters** | used within a link
- **Are uniquely assigned** | hard coded into adapter
- **Use a flat space of 48 bits** | allocated hierarchically

The first 24 bits blocks are assigned to network adapter vendor by IEEE. (1 Vendor may have more than 1 block.)

34:36:3b:d2:8a:86

The second 24 bits block is assigned by the vendor to each network adapter.(My use the same in geographically different locations!)

34:36:3b:d2:8a:86

broadcast address has set all bits to 1: ff:ff:ff:ff:ff:ff, enables to send a frame to all adapters on the link.

By default, adapters only decapsulate frames addressed to the local MAC or broadcast address. The promiscuous mode enables to decapsulate everything.
Why don't we simply use IP-addresses?

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

You need to solve two problems when bootstrapping an adapter:

- Who am I? (How do I acquire an IP address | MAC-to-IP binding) | Dynamic Host Configuration Protocol DHCP
- Who are you? (Given an reachable IP-address on a link, how do I find out what MAC to use) | IP-to-MAC binding | Address Resolution Protocol ARP

Network adapters traditionally acquire an IP address using **DHC**:

1. **Discovery** | Client searching DHCP Server (via broadcast)
2. **Offer** | DHCP server sending offer to client
3. **Request** | Client request IP from DHCP server
4. **ACK** | DHCP server assigns IP and sends ACK

DHCP discovery		DHCP offer	
dstmac	ff:ff:ff:ff:ff:ff	dstmac	34:36:3b:d2:8a:10
payload	I want an IP	payload	use 192.168.1.9

The **ARP** enables to discover the MAC associated to an IP:

1. ARP **Request**: Who has <some Ip> tell <my IP> to broadcast MAC
2. ARP **Reply**: <some IP> is at <this MAC>
3. Requester puts entry in his **ARP-table**

ARP request		ARP reply	
dstmac	ff:ff:ff:ff:ff:ff	dstmac	34:36:3b:d2:8a:10
payload	Who has 192.168.1.10? Tell 192.168.1.9	payload	192.168.1.10 is at 34:36:3b:d2:8a:86

ARP table

192.168.1.10	34:36:3b:d2:8a:86
...	...

3.3 How do we share a network medium?

Some medium are **multi-access**: > 1 host can communicate at same time.
Problem: Collision lead to garbled (verstümmelt) data.
Solution: Distributed algorithm for sharing the channel.
Essentially there are three techniques to deal with Multi Access Control (MAC):

- **Divide the channel into pieces** | either in time or frequency
- **Take turns** | pass a token for the right to transmit
- **Random access** | allow collisions, detect them and then recover

3.4 What is Ethernet?

- Was invented as a broadcast technology
- Is the dominant wired LAN technology
- Has managed to keep up with the speed race

Ethernet offers an **unreliable** and **connectionless** service.
Unreliable:

– Receiving adapter does not acknowledge anything
– Packets passed to the networks layer can have gaps

Connectionless:

– No handshake between sender and receiver

Traditional Ethernet relies on CSMA/CD (carries-sense multiple access with collision detection). All hosts were on a big bus-cable connected. You needed to sense the cable in order to know if someone is speaking. multiple hosts had access to the medium, while you were speaking you still were listening to detect collisions. CSMA/CD imposes **limits** to the network length.
Network length = $\frac{min\ frame\ size \cdot speed\ of\ light}{2 \cdot bandwidth}$
For this reason Ethernet imposes a minimum packet size of 512 bits.
Modern Ethernet links interconnect exactly two hosts, in full duplex, rendering collisions impossible.

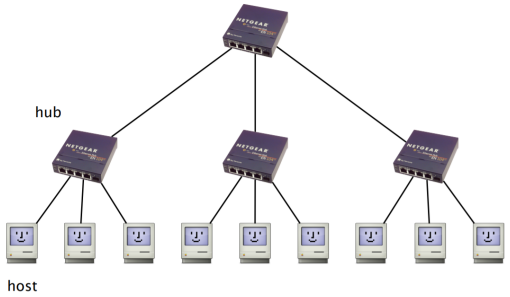
- CSMA/CD is only needed for half-duplex communication
- This means the 64 Byte restriction is not strictly needed (but still kept)
- Multiple Access Protocols are still important for wireless communication

The Ethernet header is simple, composed of 6 fields:

Ethernet efficiency is ≈ 97.5% (paylod/framesize).

3.5 How do we interconnect segments at the link layer?

Historically, people connected Ethernet segments together at the physical level with ethernet **hubs**. Hubs work by repeating bits from one port to all the others (flooding everything).

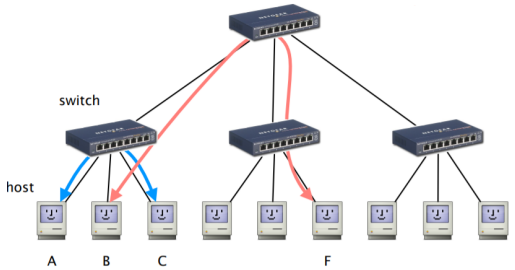


- Advantages:
  – Cheap, simple
- Disadvantages:
  – Inefficient
  – Limited ot one LAN technology
  – Limited number of nodes/distances

LANs are now almost exclusively composed of Ethernet **switches**. Switches connect two or more LANs together on the Link layer, actings as L2 gateways. Switches are **store and forward** devices, they:

- Extract the DST MAC | from the frame
- Look up the MAC in a table | using exact match
- Forward the frame | on the appropriate port

Similiar to IP routers, except they are one layer below. Switch enables each LAN segment to carry its own traffic (no flooding of network).



Switches are Plug-and-Play they build their forwarding table on their own.
The advantages of switches are numerous:

- Advantages:

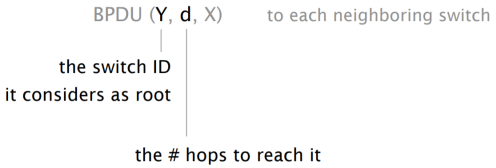
- Only forward frames where needed | avoids unnecessary load on segments
- Join segments using different technologies
- Improved privacy | hosts can only snoop traffic traversing their segment
- Wider geographic span | separates segments allow longer distance

When Frames arrive:
– Inspect the SRC 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 flood the network (when in doubt, shout).
When a frame arrives with an unknown DST → forward the frame out all ports, except where the frame arrived from.
Flooding enables **automatic discovery** of hosts, but with loops the load increases exponentially!
Solution: Reduce the network to one logical **spanning tree**.
In practice, switches run distributed Spanning-Tree-Protocol (**STP**).
Construction of a spanning tree in a nutshell: Switches...

- Elect a root | the one with the smallest identifier
- determine if each interface is on the shortest path from the root | if not → disable it.

For this switches exchange Bridge Protocol Data Unit (BPDU) messages
Each Switch X iteratively sends:



Initially:
- Each switch proposes itself as root | sends (X,0,X) on all its interfaces
- Upon receiving (Y,d,X), checks if Y is a better root | if so, consider Y as new root, flood updated message
- Switch compute their distance to the root, for each port | simply add 1 to the distance received, if shorter, flood
- Switches disable interfaces not on shortest-path

tie-breaking:
- Upon receiving ≠ BPDUs from ≠ switches with = cost → pick the BPDU with the lower siwtch sender ID
- Upon receiving ≠ BPDUs from a neighboring switch → Pick the BPDU with the lowest port ID.

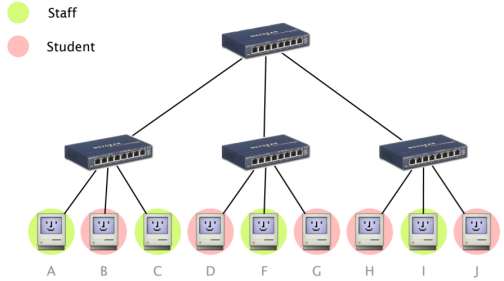
To be robust, STP must react to failures:
- Any switch, link or port can fail | including the root switch
- Root switch continuously send messages | announcing itself as the root (1,0,1), others forward it
- Failures are detected through timeout (soft state) | if no word from root in X, times out and claims to be root

3.6 Virtual Local Aerea Networks VLANs

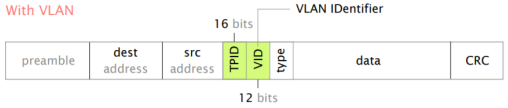
The Local Area Networks we have considered so far define single broadcast domains (if one broadcasts, everyone receives it).
As the networks scales, operators like to **segment** thir LANs.
Why?

- Improves **security** | smaller attack surface (visibility & injection)
- Improves **performance** | limits the overhead fo broadcast traffic
- Improves **logistics** | separate traffic by role (staff, student,...)

You do not want to separate your LAN physically (huge pain), but reader do it in software → **Virtual** Networks.
Definition:
A VLAN identifies a set of ports attached to one or more Ethernet Switches, forming one broadcast domain.

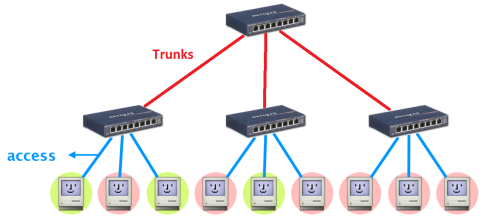


Switches need **configuration** tables telling them which VLANs are accessible via which interface.
To identify VLAN, switches add **new header** when forwarding traffic to another switch.



With VLANs, Ethernet links are divided in two sets: **access** and **trunks** (inter switch) links.
Trunks carry traffic for more than one VLAN (access not), there the new header is needed! On the access links not! Communication between VLANs goes over a router! Each switch runs one MAC learning algorithm for each VLAN.

- When a switch receives a frame wit an unknown or broadcast DST.
  → it forwards it on all the ports that belong to the same VLAN
- When a switch learns a SRC address on a port
  → it associates it to the VLAN of this port and only uses it when forwarding frames to this VLAN

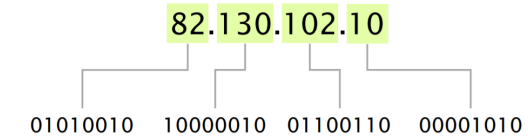


Switches can also compute per VLAN spanning trees → distinct SPT for each VLAN! This enables better use of the network.

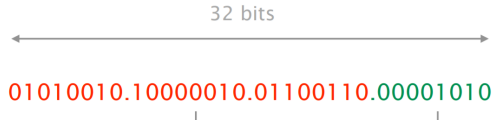
4 The Network Layer

4.1 IPv4 addresses

IPv4 addresses are unique 32-bits number associated to a network interface (on a host, router). IP addresses are usually written using dotted-quad notation.



Routers forward packets based on their DST IP address. If IP addresses were assigned arbitrarily routers would require forwarding table entries for all of them!
IP addresses are hierarchical, composed of aprefix (network address) and a suffix (host address).



Each prefix has a given length, usually written using the slash notation:
IP prefix: 82.130.102.0/24→ prefix length in bit
The suffix part can then be used to address the hosts of the network.
•The **first address** of the suffix (all zeros) is used to identify the **network itself**.
• The **last address** of the suffix (all ones) is used ot determine the **broadcast** address.
# of hosts = $2^{32-\#\text{suffix}} - 2$
Prefixes are also sometimes specified using an address and a **mask**. ANDing the address and the mask gives you the prefix.

Mask: set the number of suffix bits to one, the rest zeros (from left to right).

Address	82.130.102.0
	01010010.10000010.01100110. 00000000
	11111111.11111111.11111111. 00000000
Mask	255.255.255.0

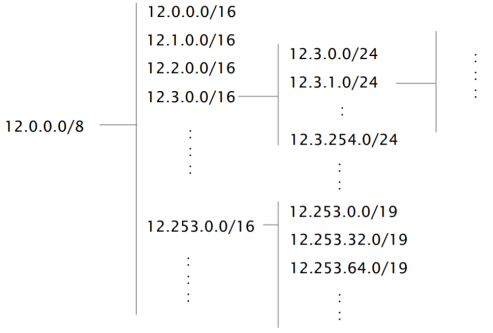
Routers forward IP packets based on the network part, not the host part. This enables a scaling of the forwarding table.
Originally there were only 5 fixed allocation sizes (classes) - known as classful networking. This is wasteful and leaded to IP address exhaustion.

	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

Problem: Class C was too small, so everybody requested B
Solution: Classless Inter-Domain Routing (CIDR) (1993)
CIDR enables flexible division between network and host addresses.

- CIDR must specify both, the address and the mask | classful was communicating this in the first address bits
- Masks are carried by the routing algorithms | it is not implicitly carried in the address.

With CIDR the maximal waste is bounded to 50%. Today, addresses are allocated in contiguous chunks:



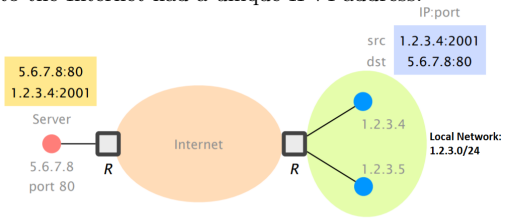
The allocation process of prefixes is also hierarchical:

4.1.1 Network Address Translation (NAT)

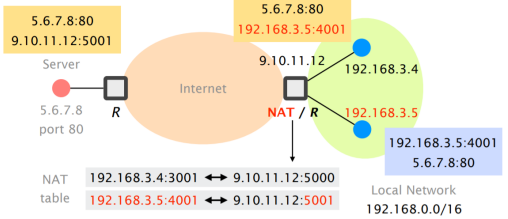
In order to be able to use more hosts than available IPv4 addresses NAT was introduced:

- Sharing a single public address between hosts
  - Port numbers are used to distinguish
- One of the main reasons we still can use IPv4
  - Saved us from address depletion
- Violates the general end-to-end principle of the Internet.
  - A NAT box adds a layer of indirection

The Internet before NAT: Every machine connected to the Internet had a unique IPv4 address:



The Internet with NAT: Hosts behind NAT get a private address. The router works as a NAT as well and executes the translation between public and private addresses (NAT table). The port numbers are used to multiplex single addresses.

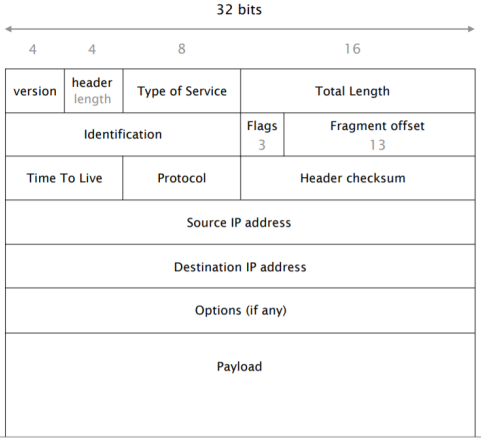


NAT also provides other (dis-)advantages:

- **Better privacy anonymization**
  - All hosts in one network get the same public IP
  - But cookies, browser version,... still identifies hosts
- **Better security**
  - From the outside you can not directly reach the hosts
  - Problematic e.g., for online gaming
- **Limited scalability**
  - Example: Wi-fi access problems in public places (often due to full NAT table).

4.1.2 IPv4 packet

This is what an IPv4 Packet looks like:



Version | "4" or "6", tells us what other fields to expect
Header length | Denotes the number of 32-bits word in the header (typically 5 (20 Bytes))
ToS | Allows different pakctes to be treated differently (low delay (VoIP), high bandwidth (Video))
Total length | Denotes the number of Bytes in the entire packet (max. 65 635, mostly 1500 because ether.)
Identification | Uniquely identifies the fragments of a particular packet.
Flag | Tells you if more packets are coming or not.
Fragment Offset | Used if need to reorder packets in right order.
Time To Live | Used to identify packets in a loop and drop them.
Protocol | Identifies the higher level protocol carried in the packet ("6" for TCP, "17" for UDP).
Header Checksum | Sum of all 16 bits words in the header (does not protect the payload).
Source IP | Uniquely identifies the Source host.
Destination IP | Uniquely identifies the Destination host.
Options | Used to provide flexibility, but mostly deactivated because security. (record route, strict source route, loose source route, timestamp, traceroute, router alert)

4.2 IPv6 addresses

IPv6 addresses are encoded in **128 bits**.

- Notation
  - 8 groups of 16 bits each separated by a colon (:)
  - Each group is written as 4 hexadecimal digits
- Simplification
  - Leading zeros in any group are removed
  - **One** section is replaced by a double colon (::). Normally the longest section.

Examples:

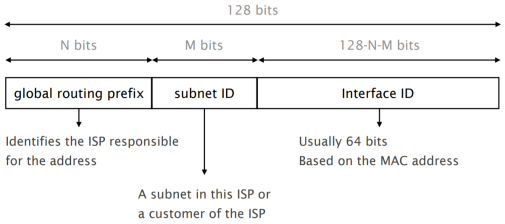
1080:0:0:0:8:8000:200C:417A → 1080::8:8000:200C:417A
FF01:0:0:0:0:0:0:0101 → FF01::101
0:0:0:0:0:0:0:1 → ::1

There are three types of IPv6 addresses:

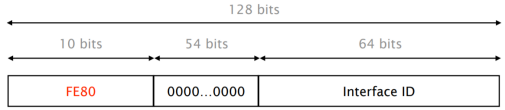
- **Unicast**
  - Identifies a **single** interface
  - Packets are delivered to this specific interface
- **Anycast**
  - Identifies a **set** of interfaces
  - Packets are delivered to the **nearest** interface
- **Multicast**
  - Identifies a **set** of interfaces
  - Packets are delivered to **all** interfaces

4.2.1 Unicast

Similar to IPv4 **global unicast** addresses are hierarchically allocated. Currently only **2000::/3** is used for global unicast → all addresses are in the range of 2000 to 3FFF.



Link-Local (**FE80**) addresses are unique to a single link (subnet) (same as private IPv4 addresses). Each host/router must generate a link-local address for **each** of its interfaces. An interface can therefore have **multiple** IPv6 addresses!



In addition to global and link-local addresses, some IPv6 uincast addresses have a special meaning.

- Unspecified address
  - 0:0:0:0:0:0:0:0
  - Used as src address if no IPv6 address is available
- **Loopback address**
  - ::1
  - 127.0.0.1 for IPv4
- IPv4 embedded
  - The lowest 32 bit contain an IPv4 address.
- **Important**
  - There no IPv6 broadcast addresses.

4.2.2 Anycast (not in exam)

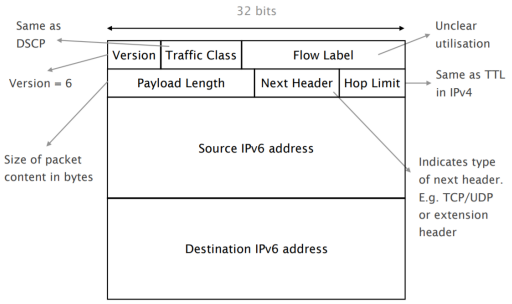
- Multiple interfaces with the same address
  - Packets are sent to the nearest interface
- Anycast uses the global unicast address range
  - E.g. for DNS and HTTP services
- IPv6 anycast is rarely used

4.2.3 Multicast (not in exam)

Multicast addresses identify a group of receivers/interfaces. Some Multicast addresses are well-known and used for bootstrapping, auto-discovery, etc.

- FF02::1
  - All IPv6 end-systems
  - E.g. hosts, servers, routers, mobile devices, ...
- FF02::2
  - All IPv6 routers
  - All routers automatically belong to this group.

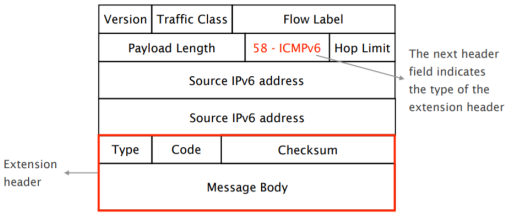
4.2.4 IPv6 packet header



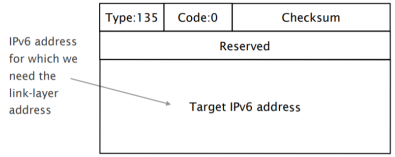
Compared to IPv4, IPv6 does...

- **Not** include checksums in the packet header
  - link, application or transport application layer provides checksums
- **Not** support fragmentation
  - End host is required to send small enough packets
- Provide more flexibility
  - flow labels and **extension headers**

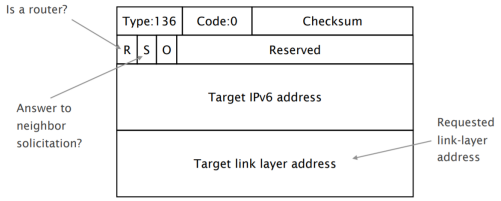
Extension header example ICMPv6:



ICMPv6 can be used for **neighbor discovery** (replacement for IPv4 ARP).
First step: **neighbor solicitation**



Second step: **neighbor advertisement**



4.2.5 Obtain IPv6 address

How can a node obtain its IPv6 address(es)?

- Manual configuration
  - As in the project, e.g. with ifconfig
- Form a sevrer using DHCPv6
- **Automatically**
  - Using its link-local address and neighbor discovery

IPv6 configuration to find link-local address:
Consider an end-to-end system which has just started, it needs an IPv6 address to be able to send ICMPv6 messages.

- MAC: 0800:200C:417A
- Link-local: FE80::M64(800:200C:417A)
  - M64: 64 bit representation of MAC address
- Neighbor solicitation for FE80::M64(800:200C:417A)
  - If **no** answer, the created link-local address is valid

IPv6 autoconfiguration to obtain the IPv6 prefix of subnet:

- Routers periodically advertise the prefix
  - Sent to all systems: FF02::1
- The advertisement can contain
  - IPv6 prefix and length
  - Network MTU to use
  - Maximum hop limit to use
  - Lifetime of the default router
  - How long generated addresses are preferred

IPv6 autoconfiguration to build global unicast address:

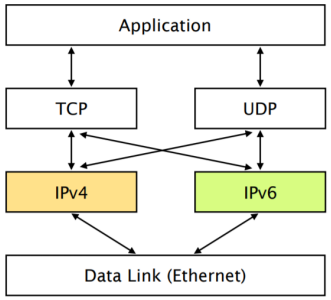
MAC	0800:200C:417A
Prefix	2001:6a8:3080:1::/64
Global Unicast	2001:6a8:3080:1:M64(800:200C:417A)

4.2.6 From v4 to v6

To port your IPv4-based application to IPv6, you need to...

- Change the used socket functions
- adjust all logging functions
- Adapt all data structures to support IPv6
- Adjust GUIs to display IPv6

Thats a huge pain and a big reason why it took us 20 years to make the transition from v4 to v6. Today, a lot applications and OSes use **dual stack** approach:

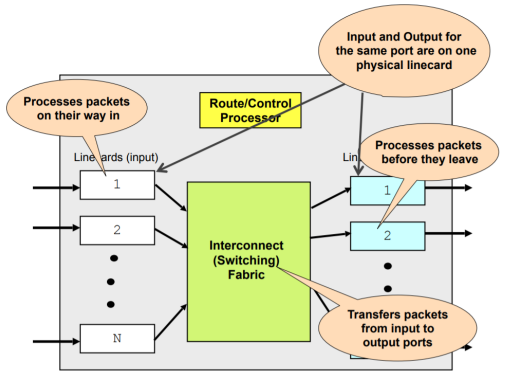


Over the years, a lot of transition mechanisms were developed (e.g. **6in4**: Tunnel IPv6 packets over static IPv4 link).

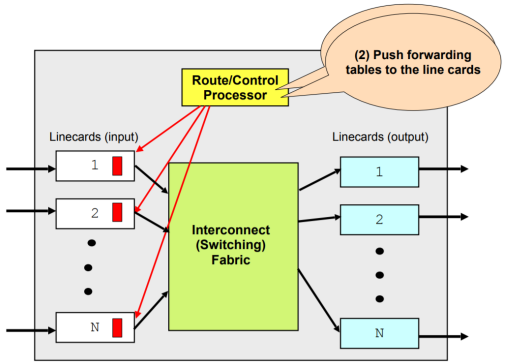
4.3 IP forwarding

Whats inside an IP router?:

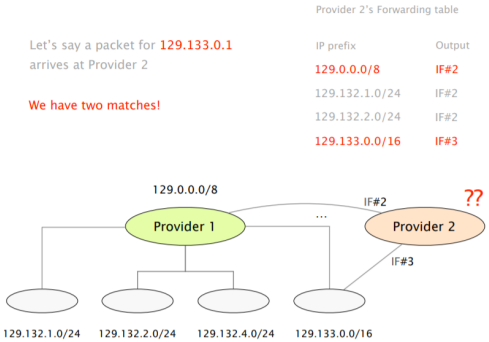
- **Data Plane:**



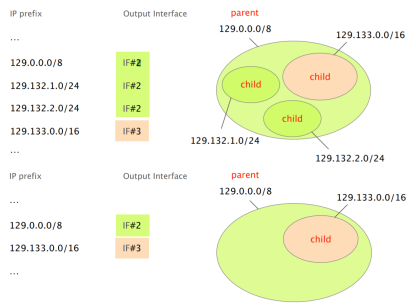
- **Control Plane:**



Routers maintain **forwarding entries** for each Internet prefix.
When a router receives an IP packet, it performs an IP **look up** to find the matching prefix.
CIDR makes forwarding harder, as one packet can match many IP prefixes.



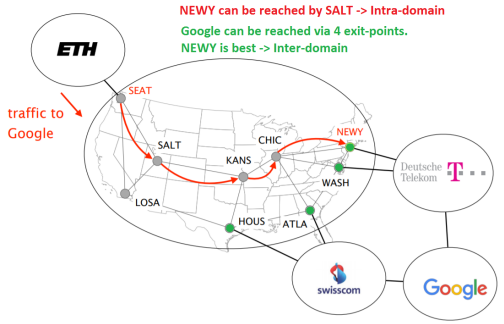
To resolve ambiguity, forwarding is done along the **most specific** prefix (IF#3)
A child prefix can be **filtered** from the table whenever it shares the same output interface as its parent:



Exactly the same forwarding!

4.4 Internet routing

Internet routing comes in two flavors: **intra**- and **inter**-domain routing.



Intra vs. **Inter** domain routing

4.4.1 Intra-domain routing

Find paths **within** a network.
Intra-domain routing enables routers to compute **good forwarding paths** to any internal subnet.
What is **good**?
Definition: A good path is a path that minimizes some network wide metric (cost, delay load, loss,...).
Approach: Assign to each link wa weight (usually static), compute the *shortest path* to each destination.

Link-state protocols
In link-state routing, routers build a precise map of the network by flooding local views to everyone.

- Each router keeps track of its incident links and cost | as well as weather it is up or down
- Each router broadcast its own links state | to give every router a complete view of the graph
- Routers run Dijkstra on the corresponding graph | to compute their shortest path and forwarding table.

Flooding is performed as in the L2 learning, except that it is **reliable**. All nodes are **ensured** to receive the latest version of all link-states.

- Challanges:
  - packet loss
  - out of order arrival
- Solution:
  - ACK & Retransmission
  - Sequence number
  - Time-To-Live for each link-state

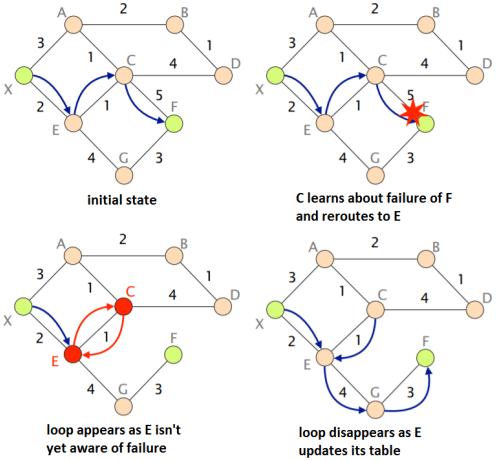
A link-state node initates flooding in 3 conditions:

- **Topology change** | link or node failure/recovery
- **Configuration change** | link cost change
- **Periodically** | Refresh (account for possible data corruption)

Once a node knows the entire topology, it can run Dijkstra to compute shortest-path.
By default, link-state protocols detect changes using software based beaconing:
Routers periodically exchange "Hello" in both directions and trigger a failure after few missed "Hellos". Tradeoff between:

- Detection speed
- Bandwidth and CPU overhead
- false positive/negative

During network changes, the link-state database of each node might differ. Inconsistencies lead to transient disruptions in the form of black holes or loops.
Blackholes appear due to detection delay, as nodes do not immediately detect failure → depends on the timeout for detecting lost "Hellos".
Transient loops appear due to inconsistent link-state database.



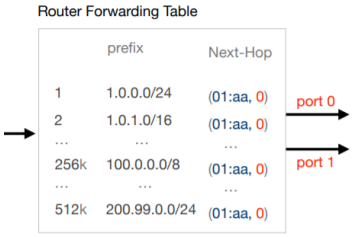
Convergence is the process during which routers seek to actively regain a consistent view of the network.
Network convergence time depends on 4 main factors:

- **Detection**
  - realizing that a link/neighbor is down
  - few ms
  - smaller timers
- **Flooding**
  - flooding news to entire network
  - few ms
  - high-priority flooding
- **Computation**
  - recomputing Dijkstra
  - few ms
  - incremental algorithms
- **Table update**
  - updating forward table
  - potentially minutes!
  - better table design

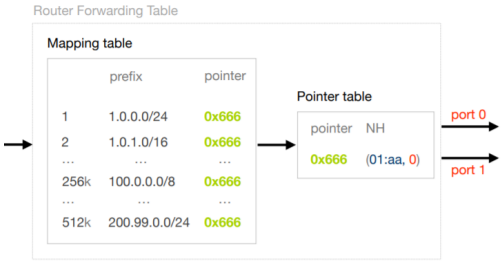
The problem with updating the forwarding table is that they are flat, means:

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

The solution is to add a layer of **indirection** (Dereferenzierung):
Replay this:



With that:



Now we only need to update 1 entry, the one in the pointer table! Hierarchical tables are able to converge within 150ms *independently* on the number of prefixes!
Today, two link-state protocols are widely used:

- **OSPF**
  - used in many enterprise & ISPs
  - work on top of IP
  - only route IPv4 by default
- **IS-IS**
  - used mostly in large ISPs
  - work on top of link layer
  - network protocol agnostic

Distance-vector protocols
Distance vector protocols are based on Bellman-Ford algorithm (see 2.1.2 (3.))
The same reasons as with the link-state cause the nodes to send new DVs.
What happens when a link changes it cost?

- Decrease cost
  - The algorithms terminates fast
  - → Good news travel fast
- Increase cost
  - The algorithm takes a while to terminate
  - → Bad news travel slow

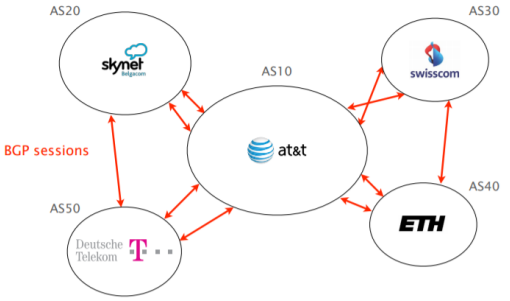
This problem is known as count-to-infinity, a type of routing loop.
Solution: Whenever a router uses another one, it will announce it an infinite cost. This technique is known as **poisoned reverse**. This method does not solve loops involving 3 or more nodes.
Actual distance-vector protocols mitigate this issue by using small "infinity" (e.g "16").
(Please see slides and exercises for examples of this problem and the solution method).

Link-state vs. Distance-vector routing:

	Message complexity	Convergence speed	Robustness
Link-State	O(nE) message sent n: #nodes E: #links	relatively fast	node can advertise incorrect link cost nodes compute their own table
Distance-Vector	between neighbors only	slow	node can advertise incorrect path cost errors propagate

4.4.2 Inter-domain routing

Find paths **between** networks.
The Internet is a network of neteorks, referred to as Autonomous Systems (AS). Each AS has a number which identifies it.
BGP is the routing protocol "gluing" the entire Internet together.



Using BGP, ASes exchange information about the IP prefixes they can reach, directly or indirectly.
BGP needs to solve three key challenges:

- **Scalability**
  - There is a huge # number of networks and prefixes
  - 700k prefixes, >50k networks, millions of routers
- **Privacy**
  - Networks do not want to divulge internal topologies
  - Or their business relationships
- **Policy enforcement**
  - Networks need to control where to send and receive traffic
  - without an internet-wide notion of a link cost metric

Link-state and Distance-vector do both not solve these problems, but **BGP** (Border Gateway Protocol) does:

- BGP relies on **path-vector routing** to support flexible routing policies and avoid count-to-infinity.
- BGP announcements carry **complete path** information instead of distances.
- Each AS appends itself to the path when it propagates announcements

Complete path information enables ASes to easily detect a loop (if they see themselves in the path).
Each AS can apply local routing policies, each AS is free to:

- Select and use any path
  - prefarably the cheapest one
- Decide which path to export (if any) to which neighbor
  - preferably none to minimize carried traffic

BGP
1. **BGP Policies** | Follow the money
2. **Protocol** | How does it work
3. **Problems** | Security, performance

1. Policies

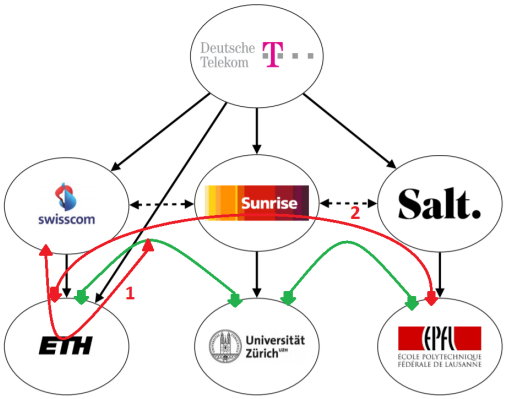
BGP is a "follow the money" protocol. Two ASes connect only if they have a business relationship. There are two main business relationships today:

(i) Customer/Provider
(ii) Peer/Peer

(i) Customer/Provider
Customers pays providers to get full Internet connectivity (monthly bill).
The amount paid is based on peak usage, usually according to the 95% rule.
Most ISPs discount unit price of data, if you pre-commit to a certain volume (Mengenrabatt).

(ii) Peer/Peer
Peers **don't** pay each other for connectivity, they do it out of **common interest**. → If they exchange tons of traffic, they save money by connecting directly to each other.

Follow the money:



Allowed: Swisscom/Salt/Sunrise make money, without paying someone to provide them. → Providers transit traffic for their customer.
Forbidden 1: Swisscom and Deutsche get money by sending over ETH, the ETH network will collapse, because all the traffic will be sent over them. → Customer do not transit traffic between their provider.
Forbidden 2: Sunrise won't receive upon, but has to provide content over their network. → Peers do not transit traffic between each other.

These policies are defined by constraining which BGP routes are **selected** and **exported**.

Selection

Which path to use? Control outbound traffic
Selection rule For a destination p, prefer routes coming from:

- Customers over
- Peers over
- Providers

There will always be a provider route, but not necessarily a customer/peer route.

Export

Which path to advertise? Control inbound traffic
Route exportation:

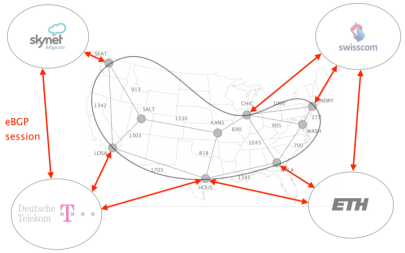
		send to		
		customer	peer	provider
from	costumer	✓	✓	✓
	peer	✓	-	-
	provider	✓	-	-

- Routes coming from customers are propagated to everyone else.
- Routes coming from peers and providers are only propagated to

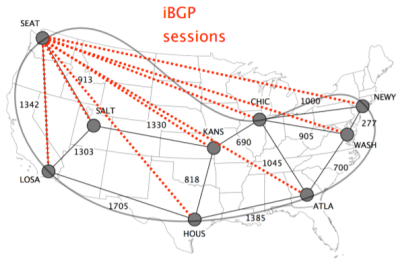
2. Protocol

BGP sessions come in two flavors:

- externeal BGP (**eBGP**)
  - Connect border routers in different ASes
  - Used to learn routes to external destinations



- internal BGP (**iBGP**)
  - Connect the routers in the same AS
  - Used to to disseminate (verbreiten) externally-learned routes internally
  - Routes disseminated internally are then announced externally again using eBGP



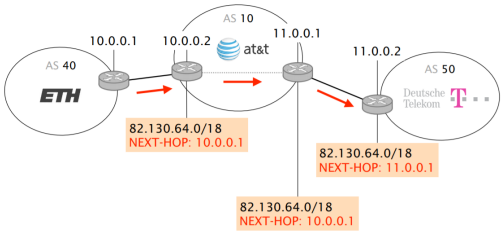
On the wire BGP is simple and only composed of four messages:

- Open
  - establish TCP-based BGP sessions
- Notification
  - report unusual conditions
- **Update**
  - inform neighbor of a new best route
  - change in best route
  - removal of best route
- Keepalive
  - inform neighbor that connection is alive

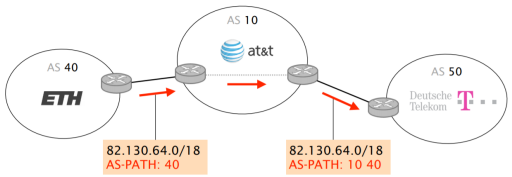
Update is the most important one. They carry an IP **prefix** together with a set of **attributes**. Attributes:

- **Next-Hop**
  - egress point identification
- **AS-Path**
  - loop avoidance
  - outbound traffic control
  - inbound traffic control
- **Local-Pref**
  - outbound traffic control
- **Med**
  - inbound traffic control

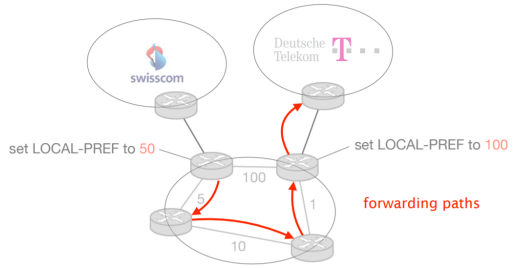
The **Next-Hop** is a global attribute which indicates where to send the traffic next. The Next-Hop is sent when the route enters an AS, it does not change within the AS. It only changes when going form one AS to an other!
Solution:
Force overwrite of Next-Hop to a virtual IP-address (**loopback** address) given to the router (unique identifier in internal network) and advertise this via OSPF in the internal network. Method is called **Next-Hop-Self**.



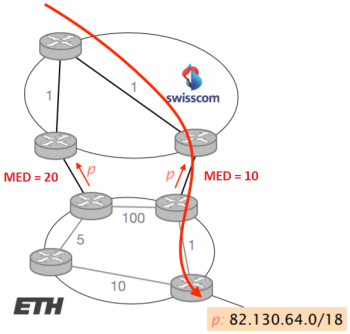
The **AS-Path** is a global attribute that lists all the ASes a route has traversed (in reverse order):



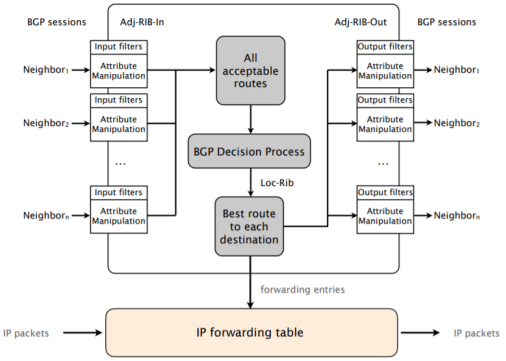
The **Local-Pref** is a *local* attribute set at the border, it represents how *preferred* a route is. Where do I want to send my traffic. Here all the routers will prefer DT over Swisscom:



The **MED** is a global attribute which encodes the relative proximity of a prefix wrt to the announcer. You can **influence** your neighbor and try to tell them where to send your receiving traffic (Only works with multiple connections to the **same AS**). In the end though, the sender decides where he sends the traffic → I can influence not control:



Each BGP router processes updates according to a precise **pipeline**:



How to choose best route - prefer routes with:

1. higher Local-Pref
2. shortest AS-Path length
3. lower MED
4. learned via eBGP instead of iGBP
5. lower iGBP metric to next-hop
6. smaller egress IP-address

3. Problems
BGP suffers from many problems!

(i) **Reachability**
(ii) **Security**
(iii) **Convergence**
(iv) **Performance**
(v) **Anomalies**
(vi) **Relevance**

i) **Reachability**
Policy routing does not guarantee reachability, even if the network is physically connected.

ii) **Security**

- **IP Hijacking**
  - Steal IP-prefix and receive all the traffic that is not intended for you → Black-hole/Snooping/Impersonation
- **Bogus AS-paths**
  - Remove ASes form the path
  - Add Ases to the path
  - Add AS hop(s) at the end of the path
- **Invalid paths**
  - AS exports a route it shouldn't
- **Missing/Inconsistent routes**

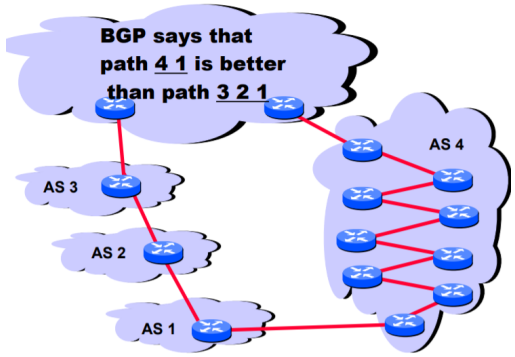
iii) **Convergence**
With arbitrary policies, BGP may have multiple stable states and never converges. It is possible that networks oscillate forever! Policy oscillations are a direct consequence of policy autonomy.

- Checking BGP correctness/convergence is not possible, even when you know all the policies!

But

- If all ASes follow the provider/peer/customer rules, BGP is **guaranteed** to converge, otherwise the network wouldn't be economical.

iv) **Performance**
BGP path selection is mostly economical, not based on accurate performance criteria



v) **Anomalies**
BGP configuration is hard to get right. Human factors are responsible for 50% to s80% of network outages.

- BGP is both bloated and underspecified
  - Lots of knobs and sometimes conflicting interpretations
- BGP is often manually configured
  - Humans make often mistakes
- BGP abstraction is fundamentally flawed
  - disjoint, router-based config. to effect AS-wide policy

vi) **Relevance**
The world of BGP policies is rapidly changing:

- ISPs are now eyeballs talking to content networks
  - e.g., Swisscom and Netflix/Youtube/Spotify
- Transit becomes less important and less profitable
  - Traffic moves more and more to interconnection points
- No system practices, yet
  - details of peering arrangements are private anyway

5 The Transport Layer

5.1 General stuff and context

What problems should be solved on transport layer?

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

What is needed to address these?

- Demultiplexing
  - **Identifier for applicaiton process**
  - Going from host-to-host (IP) to process-to-process
- Translating between bytestream and packets
  - **Do segmentation and reassembly**
- Reliability
  - **ACKS and all that stuff**
- Corruption
  - **Checksum**
- Not overloading receivers
  - **Flow Control**
  - Limit data in receiver's buffer
- Not overloading network
  - **Congestion Control**

What transport protocols do **NOT** provide:

- **Delay** and/or **bandwidth** guarantees
- Sessions that survive change of IP address

Sockets
A socket is a software abstraction by which an application process exchanges network messages with the operating system. (**OS abstraction**).
Two important types of sockets:

- UDP sockets
- TCP sockets

Ports
Is a **network abstraction**. Think of it as a **logical** interface on the host.
Port is used as a transport layer identifier (16 bits) to tell which app (socket) gets which packets. OS stores mapping between sockets and ports.
There exist some well known ports (0-1023):

ssh	22
http	80
https	443
...	...

Ports randomly given to clients lay in the range 1024-65535.

5.2 UDP: User Datagram Protocol

UDP provides a **connectionless/unreliable** transport service.
UDP provides only two services to the App. layer:

- Multiplexing/Demultiplexing among processes
- Discarding corrupted packets (optional)

Therefore it is a very lightweight communication between processes.
Advantages:

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

Popular applications that use UDP:

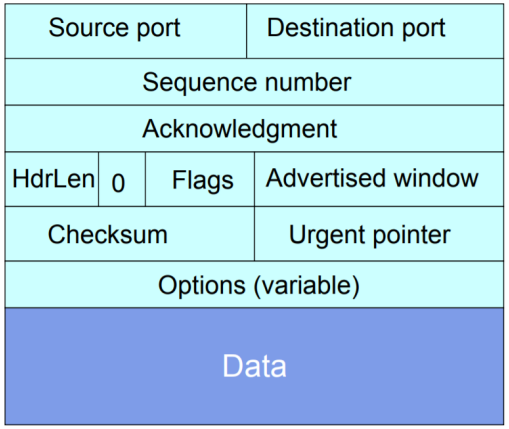
- VoIP
- Video conferencing
- online gaming
- streaming
- DNS

5.3 TCP: Transmission Control Protocol

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

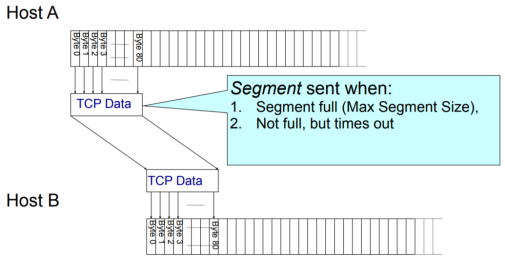
- **Reliable, in-order delivery** (see also 2.2)
  - Ensure byte stream arrives intact
    - ACKs
    - Checksums
    - Timeouts and retransmissions
- **Connection oriented**
  - Set-up and tear-down of TCP session
- **Full duplex stream of bytes service**
  - Sends ans receives stream of bytes, not messages
- **Flow control** (see also 2.2)
  - Ensure taht sender doesn't overwhelm receiver
    - sliding window
    - Cumulative ACKs
- **Congestion Control**
  - Dynamic adaption to network path's capacity

The TCP header looks as follows:

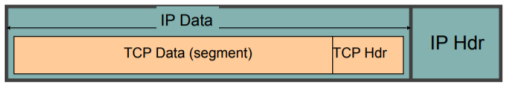


5.3.1 Segments and Sequence Numbers

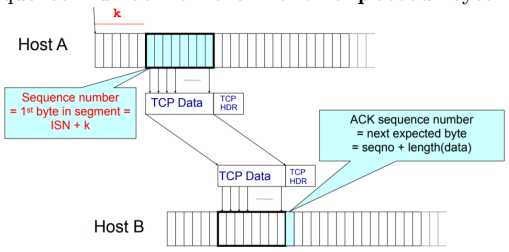
TCP stream-of-bytes service is provided by using segments:



Segments are sent either if they reached their maximal size (**MSS** (usually 1460)) or a time out is reached. The segment is embedded in the IP packet:

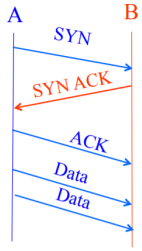


The MSS depends on the MTU and the IP/TCP-header:
MSS = MTU − (IP header) − (TCP header)
The **ISN** (Initial Sequence Number) identifies the first byte of the byte-stream. Because security reasons this number is picked randomly.
The receiver will send an ACK with the sequence number of the **next expected** byte!:



5.3.2 Connection Establishment/Teardown

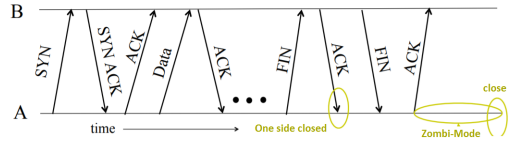
To establish connection hosts exchange ISNs with the **SYN**:



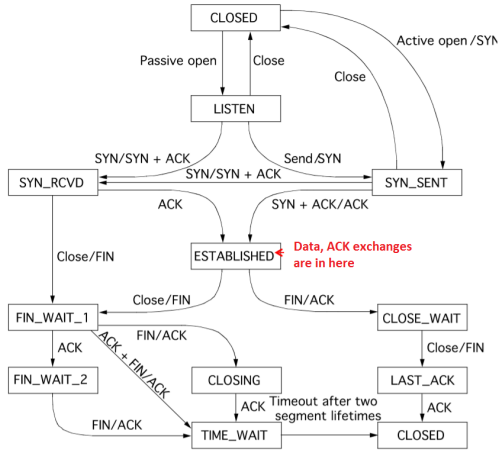
If SYN gets lost, sender waits some time and transmits it again (hard to set timer reasonable).
Tearing down connection can be made in three different ways:

- One side at a time (normal)
- Both together | send FIN+ACK (normal)
- Abrupt | RST (not normal)

In the normal cases the sender of the last ACK stays some time in Zombi-Mode in case its ACK got lost.



TCP state transitions

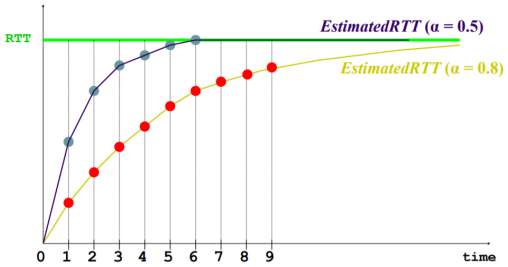


5.3.3 Timeouts and Retransmissions

Setting the timeout value is very difficult, in order to do it measurements of the **RTT** (Round Trip Time) are used. To define the RTT an exponential averaging of the measurements is used:

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

Assume RTT is constant → $SampleRTT$ = RTT



Once a segment is retransmitted, it is not used for the measurement.
A timeout is very expensive, therefore we rely mostly on duplicate ACKs:
Duplicate ACKs are a sign of a isolated loss. One could trigger a resend upon recieving k duplicate ACKs (TCP uses $k = 3$)

5.3.4 Congestion Control

Because of traffic burstiness and lack of BW reservation, congestion is inevitable and very harmful.
Van Jacobsen saved us with Congestion Control.
Congestion control aims at solving **three problems**:

- **BW estimation**
  - How to adjust the BW of a single flow to the bottleneck BW?
- **BW adaption**
  - How to adjust the BW of a single flow to variation of the bottleneck BW?
- **Fairness**
  - How to share BW fairly among flows, without overloading the network.

Congestion control differs from flow control, TCP solves both using two distinct windows:

- **Flow Control**
  - prevents one fast sender from overloading a slow receiver
  - → Solved using a receiving window (**RWND**)
- **Congestion Control**
  - prevents a set of senders from overloading the network
  - → Solved using a congestion window (**CWND**)

Sender window = min(CWND, RWND)

Detecting congestion
There are essentially three ways to detect congestion:

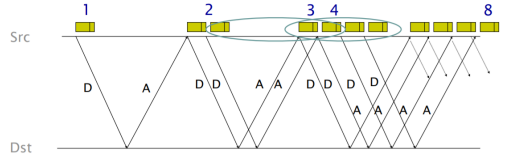
- Network could tell the source
  - Signal could be lost
- Measure packet delay
  - Signal is noisy
- Measure packet loss
  - Fail-safe signal that TCP already has to detect

→ Packet dropping is the best solution! Detecting losses can be done using **ACKs** or **timeouts**, the two signal differ in their degree of severity:

- Duplicated ACKs
  - **mild** congestion signal
  - → packets are still making it
- Timeout
  - **Severe** congestion signal
  - → multiple consequent losses

Reacting to congestion
TCP approach is to **gently increase** when not congested and to **rapidly decrease** when congested.

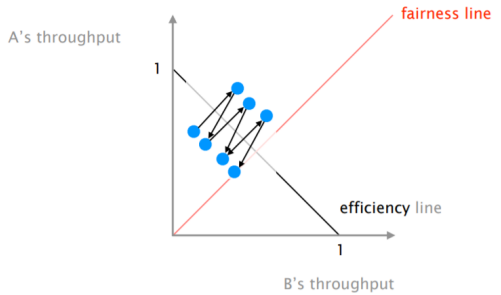
1) Get a first order estimate of the available BW.
→ start slow but rapidly increase until a packet drop occurs. This increase phase, known as **slow start**, corresponds to an exponential increase of the CWND. The problem with slow start is that it can result in full window of packet loss → we need a more gentle adjustment once we have a rough estimate of the BW.



2)Track the available BW and oscillate around its current value
→ Tow possible variations:

- Multiple Decrease/Increase (MD/MI)
- Additive Decrease/Increase (AD/AI)

This leads to four alternative designs: AIAD, AIMD, MIAD, MIMD. To select one scheme, we need to consider **fairness** → two identical flows should end up with the same BW. In the following graph we can plot the system trajectories and analyze the systems behavior. The blue dots are the result of the AIMD scheme.

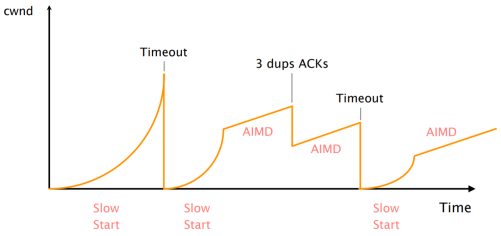


In practice TCP implements **AIMD** (gentle increase / aggressive decrease), because it converges to fairness and efficiency and then oscillates around the optimum in a stable way.
TCP congestion control almost complete:

Initially:
cwnd = 1
ssthresh = infinite
New ACK received:
if (cwnd < ssthresh):
/* Slow Start*/
cwnd = cwnd + 1
else:
/* Congestion Avoidance */
cwnd = cwnd + 1/cwnd
dup_ack = 0
Timeout:
/* Multiplicative decrease */
ssthresh = cwnd/2
cwnd = 1

Duplicate ACKs received:
dup_ack ++;
if (dup_ack >= 3):
/* Fast Recovery */
ssthresh = cwnd/2
cwnd = ssthresh

Congestion control makes TCP throughput look like a sawtooth:



6 The Application Layer

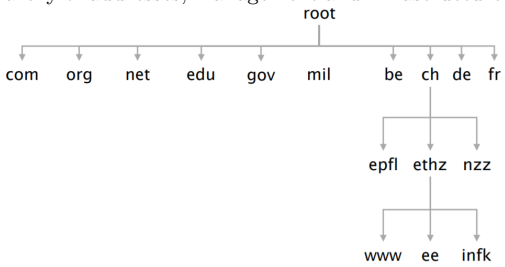
6.1 DNS

Internet has one global system for naming hosts → DNS. The DNS system is a distributed database which enables to resolve a name into an IP address.

In practice, names can be mapped to more than one IP and IPs can be mapped by more than one name.
To scale, DNS adopt three intertwined hierarchies:

- Naming structure
  - Hierarchy of addresses
- Management
  - Hierarchy of authority over names
- Infrastructure
  - Hierarchy of DNS servers

The following tree structure shows the hierarchy of addresses, management and infrastructure.

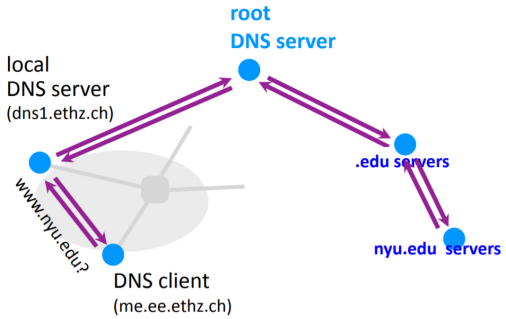


There are 13 root-servers (managed professionally). Top Level Domain (TLD) servers are also managed professionally by private or non-profit organizations.
The bottom (and bulk) of the hierarchy is managed by Internet Service Provider or locally.
Every server knows the address of the root servers. Each root server knows the address of all TLD servers. From there on, each server knows the address of all its children (but not children of children).
DNS query and reply uses UDP port 53, reliability is implemented by repeating requests.
A DNS server stores Resource Records composed of a name, value, type, TTL.

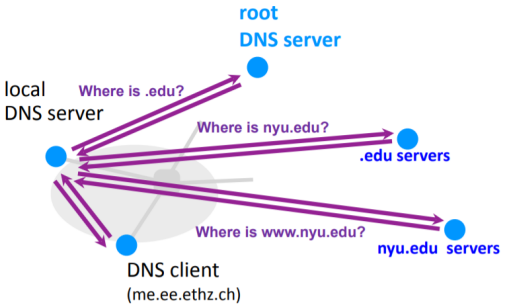
Records	Name	Value
A	hostname	IP address
NS	domain	DNS sevrer name
MX	domain	Mail server name
CNAME	alias	Canonical name
PTR	IP address	corresponding hostname

DNS resolution can either be **recursive** or **iterative**:

Recursive
When performing a recursive query, the client offloads the task of resolving to the server. Because of security and scalability reasons, this is never used.



Iterative
When performing an iterative query, the server only returns the address of the next server to query.



To reduce resolution time, DNS relies on caching:

- DNS servers cache responses to former queries
  - Also your client and the application caches.
- Authoritative servers associate a life time (TTL) to each record
  - Short TTL: flexible in changing IP → load balance, reacting to shut downs ...
  - Long TTL: fewer queries → less junk traffic
  - Setting TTL is a trade off and differs for each destination, e.g. google ≈ 240s, ETH ≈ 6000s

As top-level servers rarely change & popular website visited often, caching is very effective, practical hit rate ≈ 75%.

6.2 Web

Founded in ∼1990 by Tim Berners-Lee the goal of the World Wide Web was to provide distributed access to data. The World Wide Web is a distributed database of "pages" linked together via the Hypertext Transport Protocol (**HTTP**). The Web was and still is so successful as it enables everyone to self-publish content very easily.
The WWW is made of three key components:

- Infrastructure
  - Clients/Browsers
  - Servers
  - Proxies
- Content
  - Objects (files, pictures, videos,...)
  - Web sites (collection if Objects)
- Implementation
  - URL: name content
  - HTTP: transport content

6.2.1 Implementation

URL: name content:
A Uniform Resource Locater refers to an Internet resource:

protocol://hostname[:port]/directory_path/resource

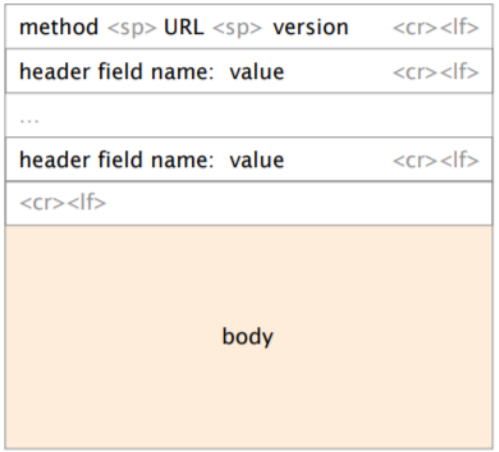
- protocol: HTTP(s), FTP, SMTP, ...
- hostname: DNS name, IP address
- port: HTTP:80, HTTPs:443, default to protocol
- directory_path/resource: identify the resource on the destination

HTTP: transport content
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
- HTTP is stateless
  - It maintains no info about past client requests

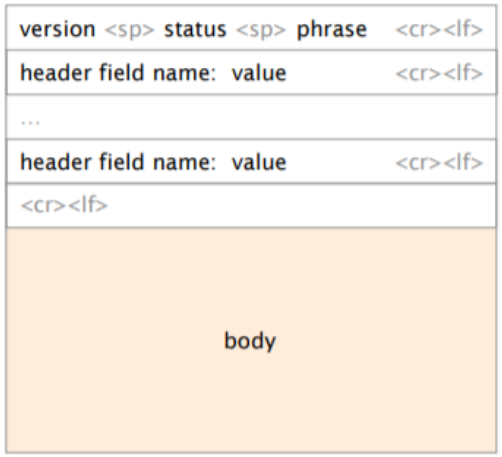
1. Protocol

HTTP clients make request to the server. Request headers are of variable lengths, but still, human readable.



HTTP servers answers to clients request. Like request headers, response headers are of variable

length and human readable.



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

- Advantages:
  - Server side scalability
  - Failure handling is trivial
- Disadvantages:
  - Some Applications need state
    shopping carts, user profiles, tracking

HTTP makes the client maintain the state, through the so called **cookies**.

- Client stores small state
  - On behalf of server X
- Client sends state in all future reqeust to X
- Can provide authentication

2. Performance Performance goals vary, depending on who you ask:

- Users
  - Fast downloads
  - High availability
- Network operators
  - No overload
- Content provider
  - Happy users
  - Cost-effective infrastructure

Solution:

(i) Improve HTTP to compensate for TCP weakspots
(ii) Caching
(iii) Replication

i)Improve HTTP
Relying on TCP forces a HTTP client to open a connection before exchanging anything. Most Web pages have multiple objects, naive HTTP opens a TCP connection for each object.
Solutions:

- **Multiple** TCP connections in parallel
  - Network operator is not happy
- Use **persistent** connections across multiple requests
  - Avoid overhead
  - Allow TCP to learn more accurate RTT
  - Allow TCP congestion window to increase
- **Pipeline** requests & replies asynchronously, on one connection
  - Send all requests, then send all replies → only 2 RTTs!

Considering the time to retrieve n small objects, pipeline wins.
Considering the time to retrieve n big objects, there is no clear winner, as bandwidth matters most.

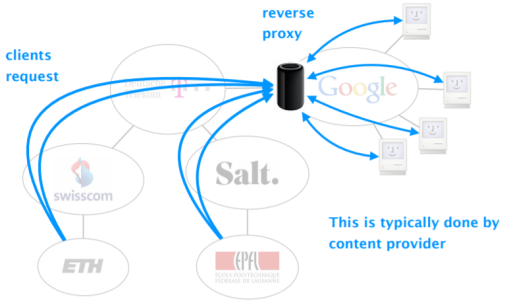
ii) Caching
Leverages the fact that highly popular **content largely overlaps**. Caching it saves time for your browser and decreases network and server load. Yet, a significant portion of the HTTP objects are **uncachable**:

- Dynamic data
  - Stock prices, score
- Scripts
  - Results based on parameters
- Cookies
  - Results may be based on passed data
- SSL
  - Cannot cache encrypted data
- Advertising
  - Wants to measure # of hits

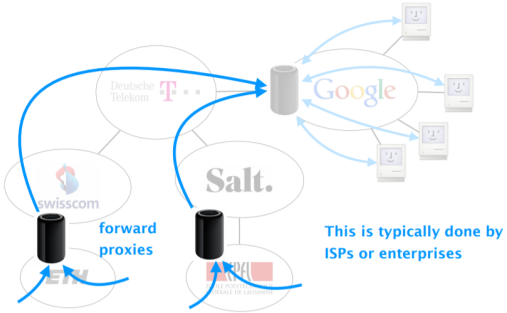
To limit staleness of cached objects, HTTP enables client to validate cached objects → *TTL*, *if-modified*.
Caching is performed at different locations:

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

Reverse proxies cache documents close to the servers, decreasing their load:



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



iii) Replication
The idea of replication is to duplicate popular content all around the globe.

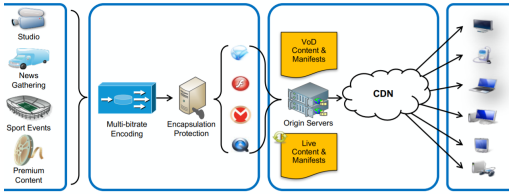
- Spreads load on server
  - e.g. across multiple data-centers
- Places content closer to client
  - only way to beat the "speed of light"
- helps speeding up uncachable content
  - Still have to pull it, but from closer

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

- **DNS-based**
  - Returns not the same IP address based on:
    - Client geo-localization
    - Server laod
- **BGP Anycast**
  - Advertise the same IP prefix form different locations
    - Avoided in practice, because BGP mostly does not follow shortest path.

6.3 Video streaming

We want the highest video quality without annoying buffering. In practice, things are complex:

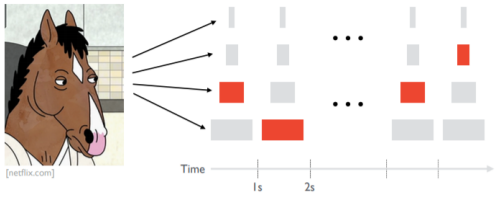


The three steps behind most contemporary solutions:

- **Encode** video in multiple bitrates
  - To enable adaptive quality.
- **Replicate** using a content delivery network
  - Bring the content physically closer to the customer
- Video player picks bitrate **adaptively**
  - Estimate available BW
  - Pick bitrate ≤ available BW

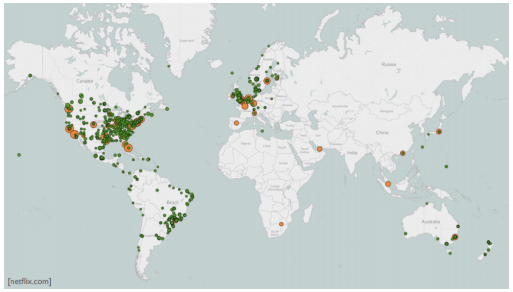
6.3.1 Encoding

Simple solution for encoding → use a "**bitrate ladder**". This table maps the available BW to the resolution which will be sent.
Problem: This doesn't take into account the **variability** in the video content. A Comic will need less BW to be shown in good quality than a sports video.
Solution: Every content will get its own table.
Your player download **chunks** of video at different bitrates. Depending on your network connectivity, your player fetches chunk of different quality. Your player gets the information about the chunks via "**Manifest**" which is downloaded at the beginning.



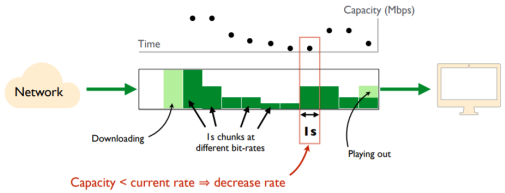
6.3.2 Replication

Put your Hardware with all your content around the globe to increase availability. See below an example of Netflix:



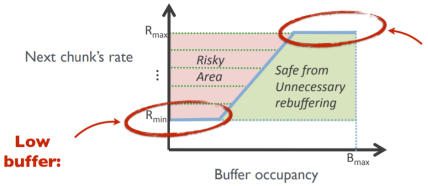
6.3.3 Adaption

The available BW varies extremely and therefore it is difficult to set the bitrate according to the capacity estimation:



In case of Netflix this method resulted in ≈ 30% unnecessary rebuffers.
A better way to adapt the bitrate is according to the buffer itself → **buffer-based adaption**:

Nearly empty buffer ⇒ small rate
Nearly full buffer ⇒ large rate



6.4 E-Mail

We'll study e-mail form three different perspectives:

- **Content**
  - Format: Header/Content
  - Encoding: MIME
- **Infrastructure/Transmission**
  - SMTP: Simple Mail Transfer Protocol
  - Infrastructure: mail-servers
- **Retrieval**
  - POP: Post Office Protocol
  - IMAP: Internet Access Message Protocol

6.4.1 Content

"What is inside the envelope"

Header	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	

E-mail relies on 7-bit U.S ASCII, if you want to send non-English text or even binary files you need the Multipurpose Internet Mail Extensions (**MIME**). MIME defines:

- Additional headers for the e-mail body:
  - MIME-Version
  - Content-Type
  - Content transfer encoding
- A set of content-types and subtypes:
  - Image with subtypes gif, jpeg
  - Text with subtypes plain, html and rich text
  - Application with subtypes postscript or ms-word
  - Mulitpart with subtypes mixed or alternative
- Base64 to encode binary data to ASCII
  - Binary files are encoded in ASCII with base64 and then the ASCII file is sent!

6.4.2 Infrastructure / Transmission

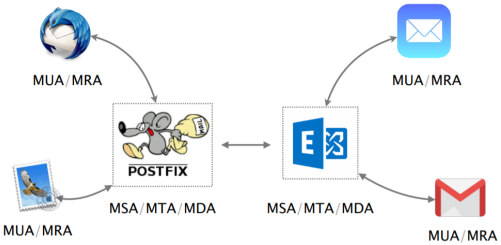
An e-mail address is composed of two parts identifying the **local mailbox** and the **domain**.

lvanbever @ ethz.ch
local mailbox domain name

We can divide the e-mail infrastructure into five functions (M=Mail, A=Agent):

User	Use to read/write e-mails (mail client)
Submission	Process email and forward to local MTA
Transmission	Queues, receives, sends mail to MTAs
Delivery	Deliver email to user mailbox
Retrieval	Fetches email from user mailbox

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



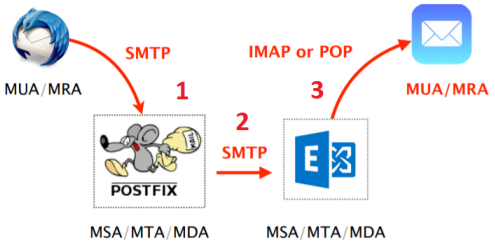
Simple Mail Transfer Protocol (**SMTP**) is the current standard for transmitting e-mails.

- SMTP is text based, client-server protocol
  - client sends the email, server receives it
- SMTP uses reliable data transfer
  - Built on top of TCP
  - Port 25 and 465 (25 should not be used anymore → SPAM)
- SMTP is a push-like protocol
  - Sender pushes the file to the receiving server (no pull)

1. The sender MUA uses SMTP to transmit the e-mail first to the local MTA (e.g. mail.ethz.ch, gmail.com, hotmail.com)

2. The local MTA then looks up the MTA of the recipient domain (DNS MX..) and transmits the email 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 two SMTP servers, but often way more. Separate SMTP servers for separate functions → SPAM-filtering, virus scanning, data leak prevention, ...

6.4.3 Retrieval

POP (Post Office Protocol):
POP is a simple protocol which which was designed to support user with intermittent Internet connectivity.
POP enables e-mail users to:

- Retrieve e-mails locally | when connected
- View/Manipulate e-mails | when disconnected

POP is heavily limited → it does not do well with multiple clients or always-on connectivity.

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

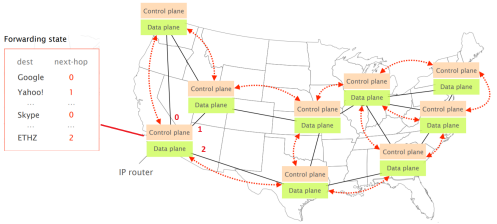
- Support multiple mailboxes and searches on the server
  - Client can create, rename, move mailboxes & search on server
- Access to individual MIME parts and partial fetch
  - Clients can download only the text part of an email.
- Support multiple clients connected to one mailbox
  - Server keeps state about each message (e.g. read, replied to, ...)

7 Programmable Networks

Is meant to be a short introduction at this point. For the exam not that relevant (hopefully), but basic concepts should be known.

7.1 The network management crisis

Networks are large **distributed systems** running a set of **distributed algorithms**. These algorithms produce the forwarding state which drives IP traffic to its destination. Operators adapt their network forwarding behavior by configuring **each network device individually**. Configuring each element is often done manually, using low-level, vendor-specific "languages". A single mistyped line is enough to bring the whole network down. The correct configuration and management of the network can be a huge pain, as we experienced in our project. Solving this problem is hard, because network devices are completely locked down (closed HW, closed SW (thanks Cisco)).

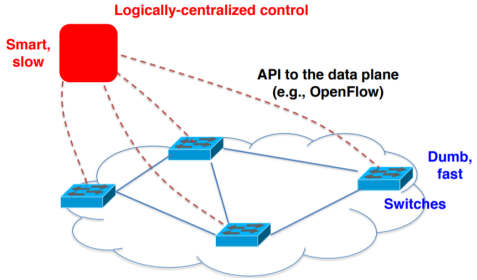


7.2 Software Defined Networking (SDN)

SDN is predicated around two simple concepts:

- **Separate** the control- from the data-plane
- Provide open **API** ot directly access data-plane

Instead that every node has a separate control-plane, a logically-centralized control is introduced.



SDN Advantages:

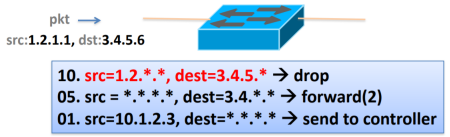
- Simpler management
  - No need to invert control-plane operations
- Faster pace of innovation
  - Less dependence of vendors and standards
- Easier interoperability
  - Compatibility only in "wire" protocols
- Simpler cheaper equipment
  - Minimal Software

7.2.1 OpenFlow Networks

OpenFlow is an API to a switch flow table. In the flow table simple packet-handling rules can be defined:

- **Pattern**
  - Match packet header bits, i.e. flowspace
- **Actions**
  - Drop, forward, modify, send to controller
- **Priority**
  - Disambiguate overlapping patterns
- **Counters**
  - #Bytes and #packets

Example of a switch flow table and an incoming packet with longest prefix match (**Exam style question!**):



OpenFlow switches can emulate different kind of boxes!:

- **Router**
  - Match: longest DST IP Prefix
  - Action: forward out a link
- **Switch**
  - Match: DST Mac address
  - Action: forward or flood
- **Firewall**
  - Match: IP address and TCP/UDP port numbers
  - Action: permit or deny
- **NAT**
  - Match: IP address and port
  - Action: rewrite address and port

Example OpenFlow applications:

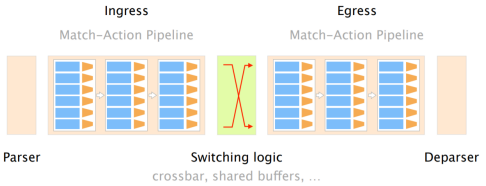
- **Dynamic access control**
  - Inspect first packet of a connection
  - Consult the access control policy
  - Install rules ot block or access policy
- **Seamless mobility/migration**
  - See host send traffic at new location
  - Modify rules to reroute the traffic
- **Server load balancing**
  - Pre-install load-balancing policy
  - Split traffic based on source IP
- Network virtualization
- Using multiple wireless access points
- Energy-efficient networking
- Adaptive traffic monitoring
- Denial-of-Service attack detection

7.2.2 Challanges

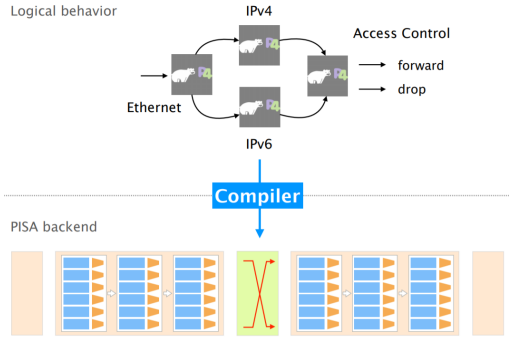
- **Heterogeneous Switches**
  - Number of packet-handling rules
  - Range of matches and actions
  - Multi-stage pipeline of packet processing
  - Offload some control-plane functionality
- **Controller delay and overhead**
  - Controller is much slower than the switch
  - Processing packets leads to delay and overhead
  - Need to keep most packets in the fast path
- **Distributed Controller**
  - Needed for scalability
  - Needed for security
- **Testing and debugging**
  - Still plenty of room for bugs
- **Programming abstractions**
  - An API makes network programming possible, not easy!

7.3 Deep Network Programmability

As we see OpenFlow is not all roses, therefore Protocol Independent Switch Architecture (**PISA**) and **P4** entered the game. PISA is used for high-speed programmable packet forwarding. Architecture of PISA:



By default PISA doesn't do anything, it's just an "architecture". P4 is a domain specific language which describes how a PISA architecture should process packets:



If you think this is the shit, consider taking Advanced Topics in Communication Networks.