Disclaimer

This should be a summary of the Communication Networks course. The goal is to update it weekly with the currently taught material.

Communication Networks

Marco Dober

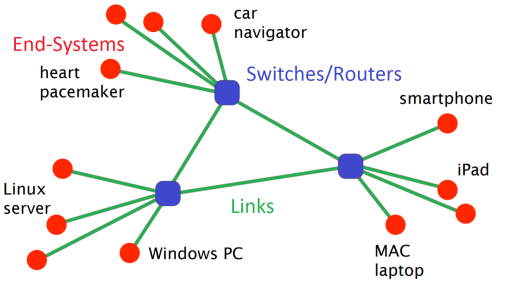
1st April 2019

1 Overview

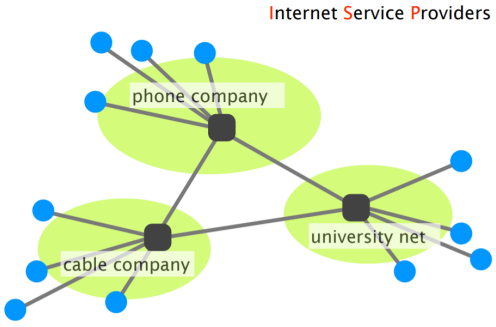
1.1 What is a network made of?

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?

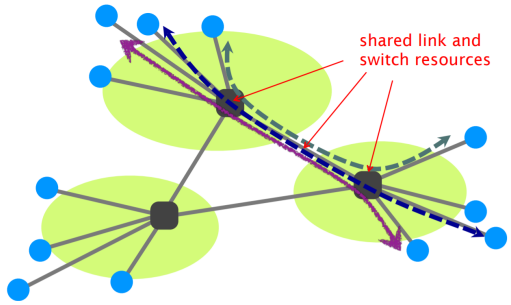
So far we discussed the "last mile" of the Internet. 3 must-have **requirements** of a good network topology:

- **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.

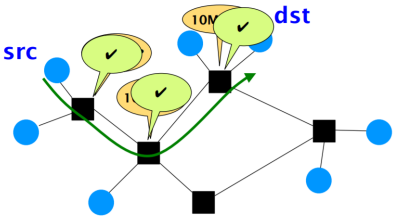


There exist two approaches of sharing, both are examples of statistical multiplexing:

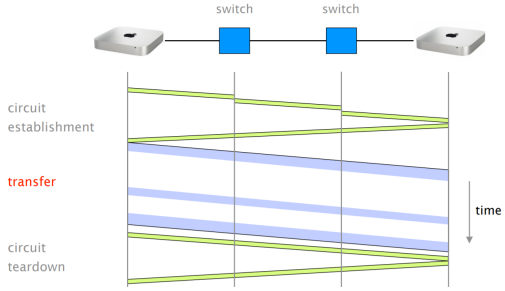
- **Reservation**
  principle: reserve needed bandwidth in advance
  multiplexing: at the flow-level
  implementation: **circuit-switching**
- **On-demand**
  principle: send data when you need
  multiplexing: at the packet-level
  implementation: **packet-switching**

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



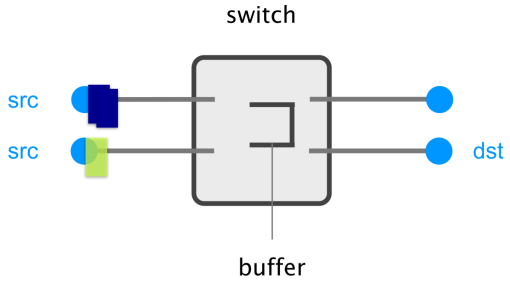
(1) src sends a reservation request for 10Mbps to dst
(2) switches "establish a circuit"
(3) src starts sending data
(4) src sends a "teardown circuit" message



- 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

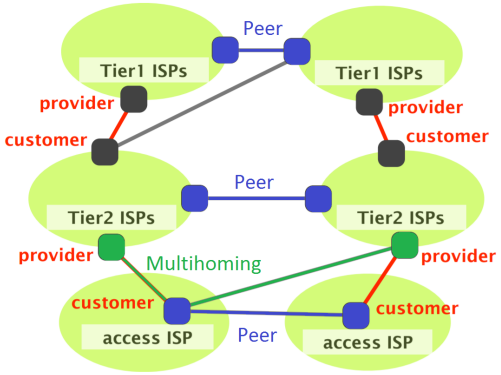
Packet-switching beats circuit-switching with respect to **resiliency** (robustness) and **efficiency**.

Internet ♥ packets

1.3 How is it organized?

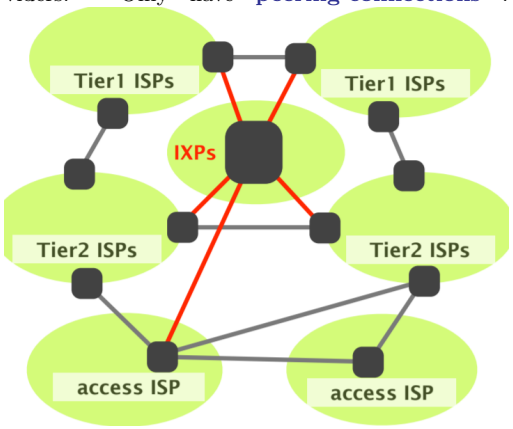
The Internet has a hierarchical structure and consists of about 60'000 networks:

- **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?

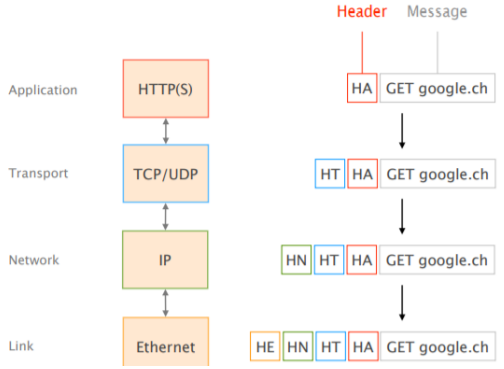
Use **protocols** to enable communication between processes in different networks. Protocols are like a conversation convention. There are thousands of different protocols. Subdivide in different **layers** to keep stuff simple (Modularity).

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**

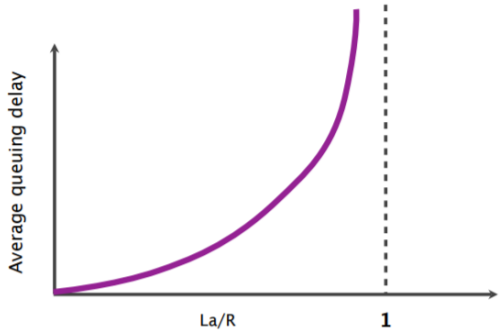
Delay
→ transmission | link property
→ propagation | link property
→ processing | traffic | mostly tiny
→ queuing | traffic | hardest to evaluate
  – arrival rate at the queue
  – transmission rate of outgoing link
  – traffic burstiness

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

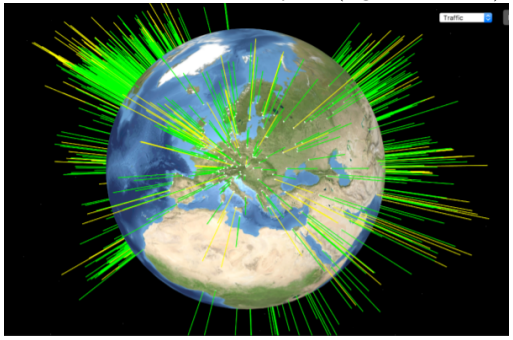
a = average packet arrival rate [packet/sec]
R = transmission rate of outgoing link [bit/sec]
L = fixed packet length [bit]

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

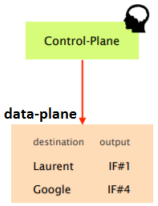
2.1 Routing

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:

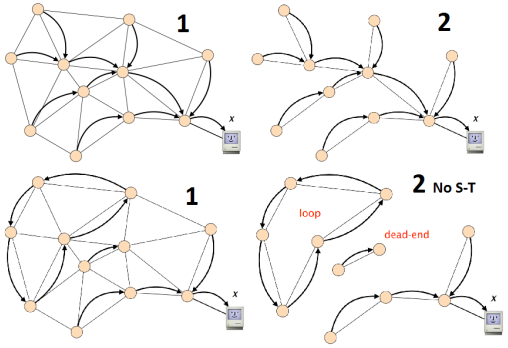
- No dead ends
- No loops

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:

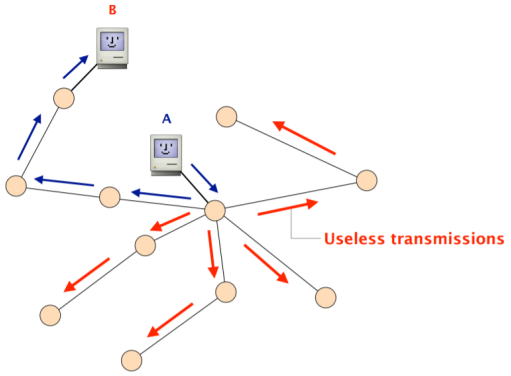
1. Use tree-like topologies | Spanning-tree
2. Rely on a global network view | Link-state
3. Rely on distributed computation | Distance vector

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. Tree-like topologies
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. Rely on a global network view
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**:

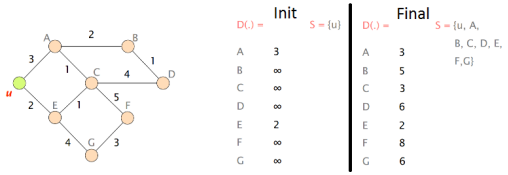
Initialization

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

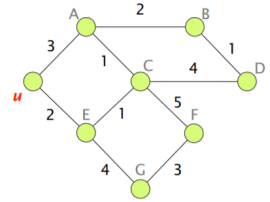
Loop

```
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. Rely on distributed computation
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
```

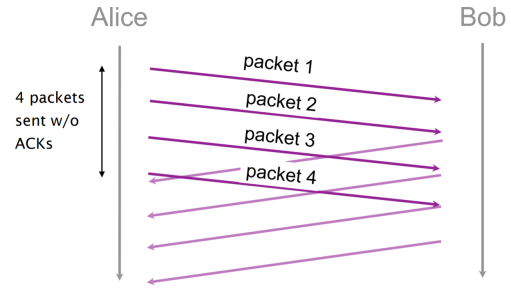
```
receive_packet(p);
if check(p.payload) == p.checksum:
    send_ack();

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

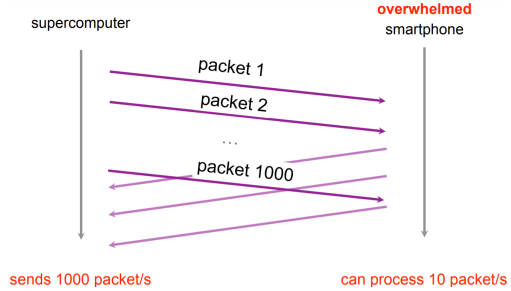
There is a clear tradeoff between timeliness and efficiency in the selection of the timeout value. Big challenge to choose optimal value.
Small timers: Risk of **unnecessary retransmissions**
Large timers: Risk of **slow transmission**

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



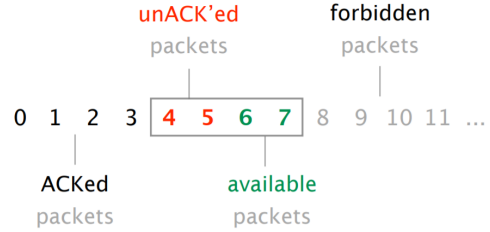
One problem that can occur is:



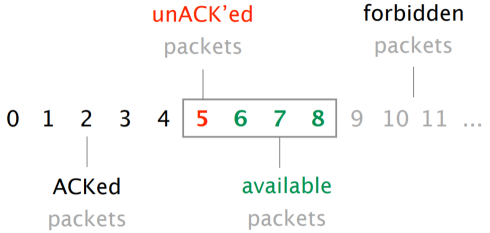
To solve this issue we need a mechanism for **flow control**. Using a **sliding window** is one way to do that:

- 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.

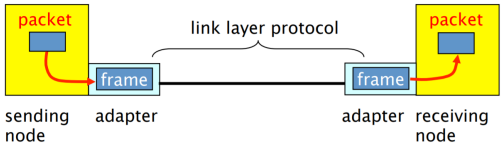
Duplicates:
Can lead to duplicated ACKs whose effect depends on the ACKing mechanism:
- Individual ACK: no problem
- Full feedback: no problem
- Cumm. ACKs: problematic

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

dstmac	ff:ff:ff:ff:ff:ff
payload	I want an IP

DHCP offer

dstmac	34:36:3b:d2:8a:10
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

dstmac	ff:ff:ff:ff:ff:ff
payload	Who has 192.168.1.10? Tell 192.168.1.9

ARP reply

dstmac	34:36:3b:d2:8a:10
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 (carriers-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.

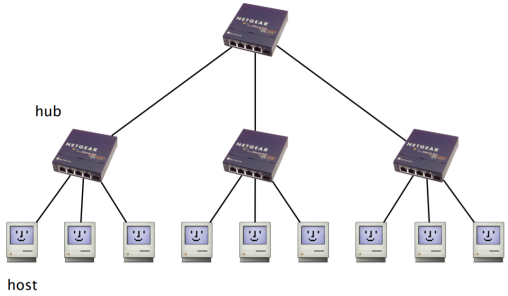
- CSMA/CD is only needed for half-duplex communication
- This means the 64 Byte restriction is not strictly needed (but still kept)
- Multiple Access Protocols are still important for wireless communication

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).

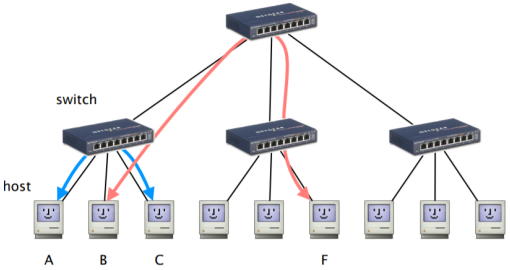


- Advantages:
  - Cheap, simple
- Disadvantages:
  - Inefficient
  - Limited ot one LAN technology
  - Limited number of nodes/distances

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

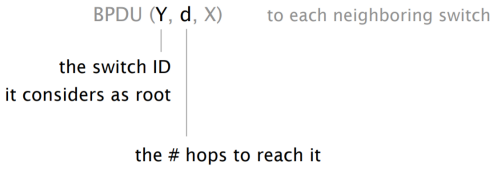
- Improved privacy | hosts can only snoop traffic traversing their segment
- Wider geographic span | separates segments allow longer distance

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.

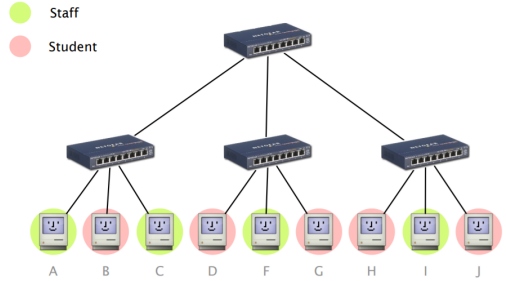
To be robust, STP must react to failures:
- Any switch, link or port can fail | including the root switch
- Root switch continuously send messages | announcing itself as the root (1,0,1), others forward it
- Failures are detected through timeout (soft state) | if no word from root in X, times out and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

3.6 Virtual Local Aerea Networks VLANs

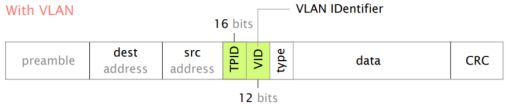
The Local Area Networks we have considered so far define single broadcast domains (if one broadcasts, everyone receives it).
As the networks scales, operators like to **segment** thir LANs.
Why?

- 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.

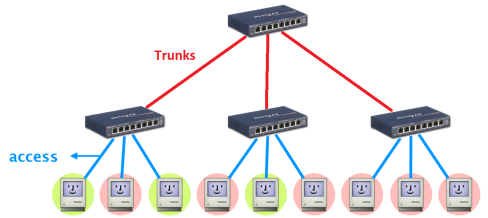


Switches need **configuration** tables telling them which VLANs are accessible via which interface.
To identify VLAN, switches add **new header** when forwarding traffic to another switch.



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

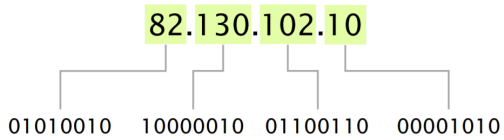


Switches can also compute per VLAN spanning trees → distinct SPT for each VLAN! This enables better use of the network.

4 The Network Layer

4.1 IP 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

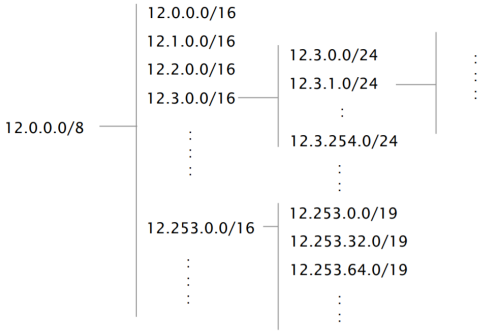
Routers forward IP packets based on the network part, not the host part. This enables a scaling of the forwarding table.
Originally there were only 5 fixed allocation sizes (classes) - known as classful networking. This is wasteful and leaded to IP address exhaustion.

	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:

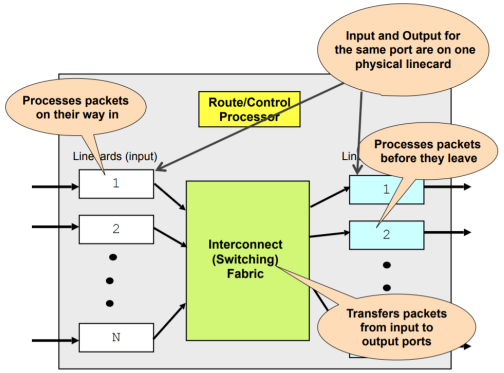


The allocation process of prefixes is also hierarchical:

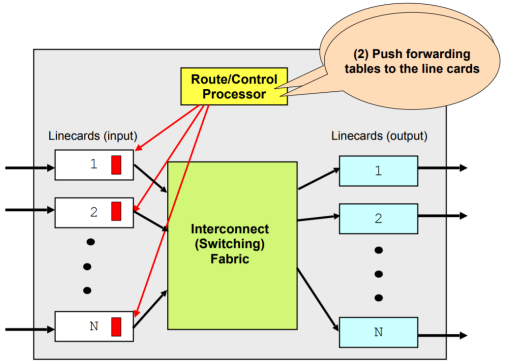
4.2 IP forwarding

Whats inside an IP router?:

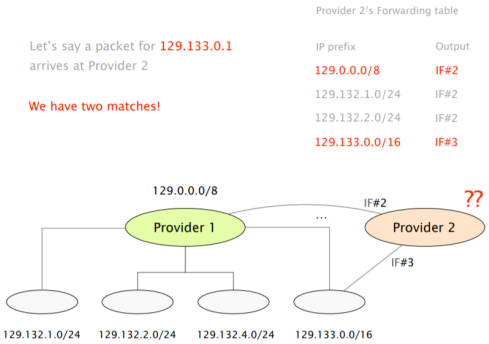
- Data Plane:



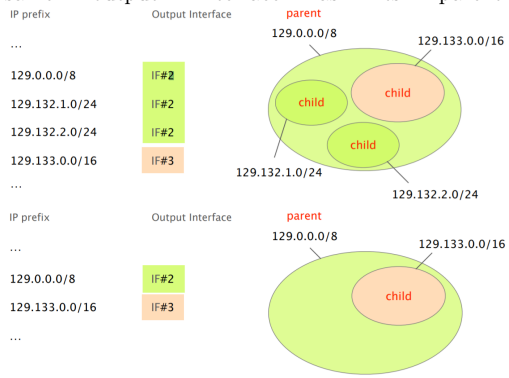
- Control Plane:



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!

4.3 IP header