Computer Networks - Lecture 1 & 2

- Author: Ruben Schenk
- Date: 01.06.2021
- Contact: ruben.schenk@inf.ethz.ch

Course Structure

Module 1

- Part 1: Overview & Principles
- Part 2: Applications
- Part 3: Transport
- Part 4: Algorithms

1. Overview & Principles

1.1 What is a network made of?

A `network` consists of three basic components:

- `End systems`: Send an receive data, such as computers, smartphones, etc.
- `Switches / Routers`: Forward data to the desired destination.
- `Links`: Connect the routers and switches and end systems.

A `Digital Subscriber Line (DSL)` brings high bandwidth to households over *phone lines*. Another common access type is using `Cable Access Technologies (CATV)`, where a cable provider is connected via a fiber cable to some distribution center and from there via copper cable to different households. The fiber cable is shared between many different households.

1.2 How is a network shared?

An `Internet Service Provider (ISP)` such as Swisscom benefits from sharing by `statistical multiplexing`. They might have a 100 mbps line on which they make subscriptions of 5 mbps. Knowing that it is very unlikely that every user is active at the same time, they send out more contracts that theoretically supported (that is, more than 20 contracts in our example). An `oversubscription` of x100 is generally accepted.

1.2.1 Resource Sharing

We differ between two types of resource sharing:

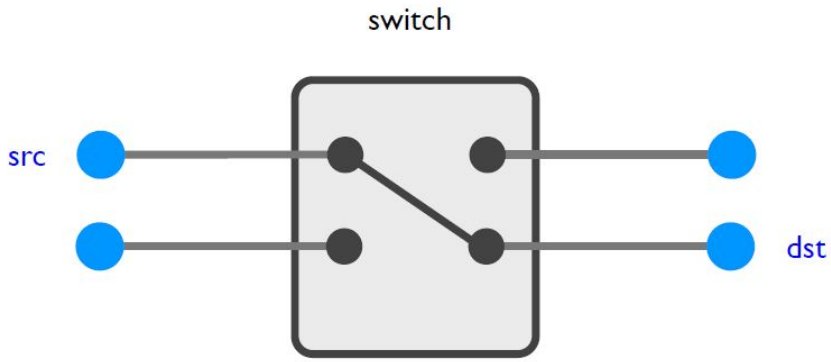
- `Reservation`: Reserve the bandwidth you need in advance, happens at the flow-level. Implemented by `circuit-switching`.
- `On-demand`: Send data whenever you need, happens at the packet-level. Implemented by `packet-switching`.

For each network flow we define P has the peak flow rate and A as the average flow rate. The problem with *reservation* is that we must reserve a flow of P, however the level of utilization is only $\frac{A}{P}$. We therefore put the general rule:

- *Reservation makes sense* if $\frac{P}{A}$ is small.
- *Reservation wastes capacity* if $\frac{P}{A}$ is big.

12.2.1 Circuit Switching

`Circuit switching` relies on the `Resource Reservation Protocol`. The source end system "asks" along the circuit to the desired destination each switch whether or not it can provide the desired speed. If each switch agrees, the circuit is established. Inside each switch, the circuit is "reserved" according to the following figure:

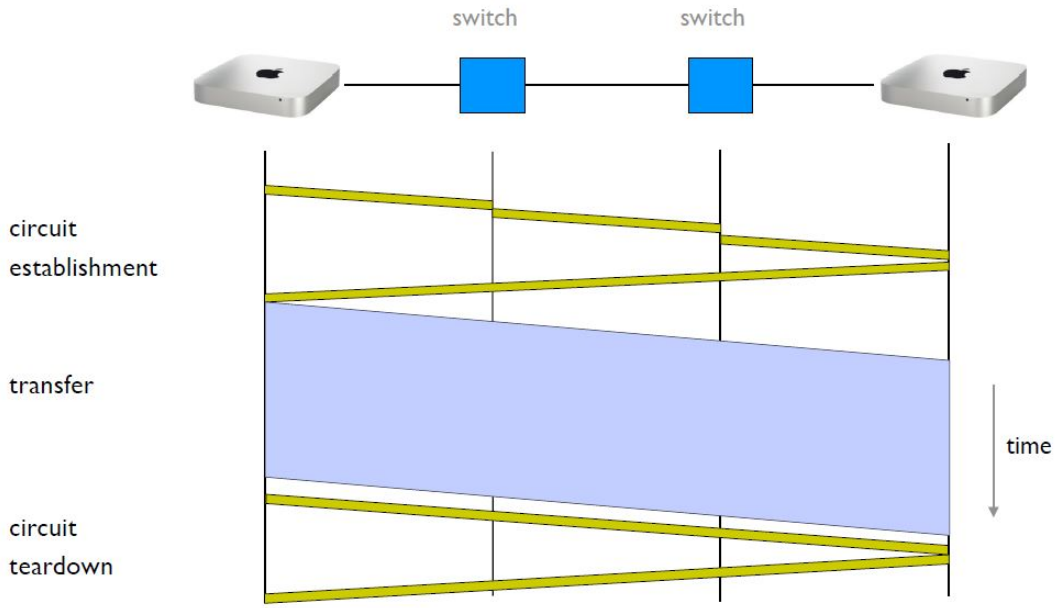


There exist many kinds of circuits, we introduce the following two types:

- `Time-based multiplexing`: We divide the whole frequency of a link into different time slots and allocate one slot per circuit.
- `Frequency-based multiplexing`: We divide the frequency of our link into different frequency bands and allocate one or more bands per circuit.

We might combine the two types mentioned above together.

Example: The figure below shows an example of data transfer using circuit switching:



Problems with circuit switching:

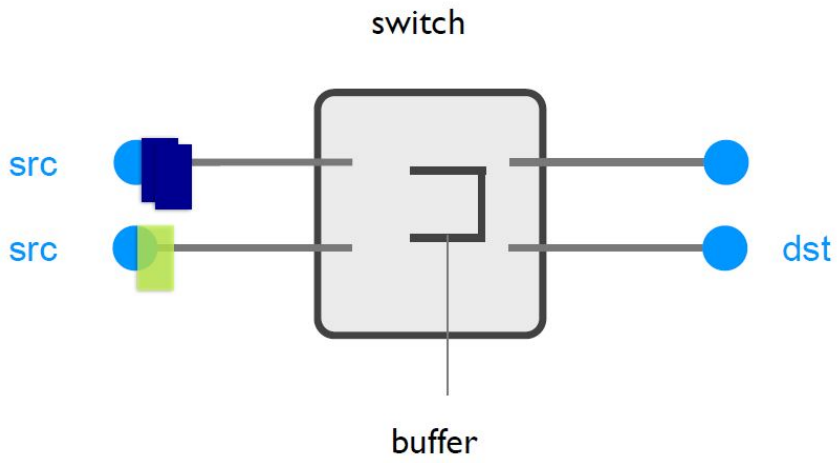
- The circuit is `mostly idle` when we're only sending traffic bursts.
- If we shorten the transfer time, a lot of time is wasted during circuit establishment and teardown.
- We require a new circuit setup upon a failure of a switch in the circuit.

Pros with circuit switching:

- We have predictable performance.
- Once established, we have a simple and fast mechanism to send data.

12.2.2 Packet Switching

Data transfer is done using independent `packets`. Each packet contains a destination. At each switch, the packet is forwarded according ot its destination, as seen in the following figure.



To absorb transient overload, each switch needs a buffer.

Pros of packet switching:

- Packet switching `routes around trouble`.
- Efficiently uses resources.

Problems with packet switching:

- Unpredictable performance.
- Requires buffer management and congestion control.

> The internet uses **packet switching** and we will focus on this for the rest of the course.

1.3 How does communication in a network happen?

Assume there are two people, Alice and Bob, which wish to exchange some data over the internet. To exchange data, the use a set of `network protocols`. A protocol is like a *conversational convention*.

1.3.1 Network Layer Model

To provide structure to the design of network protocols, internet communication can be decomposed into 5 independent `layer`:

- L5 - Application layer: Provides network access, exchanges `messages` between processes. Protocol: HTTP, SMTP, FTP, etc.
- L4 - Transport layer: Provides end-to-end delivery, transports `segments` between end systems. Protocol: TCP, UDP, etc.
- L3 - Network layer: Provides a global best-effort delivery, moves `packets` around the network. Protocol: IP.
- L2 - Link layer: Provides local best-effort delivery, moves `frames` across a link. Protocol: Ethernet, Wifi, DSL, LTE,

etc.
- L1 - Physical layer: Provides physical transfer of bits, moves `bits` around a physical medium. Protocol: TWister pair, fiber, coaxial cable, etc.

> Each layer provides *a service to the layer above* by using the *services of the layer directly below it*.

Each layer takes messages from the layer above, and `encapsulates` them with its own header and/or trailer.

Computer Networks - Lecture 3&4

- Author: Ruben Schenk
- Date: 01.06.2021
- Contact: ruben.schenk@inf.ethz.ch

1.3.2 The End-End Principle

If no reliable transport is provided, every application that needs reliability has to engineer it from scratch, which is a wasteful effort. However it also wouldn't make sense for the network layer to provide reliable delivery, since it is a burden for applications that rely on speed and not on reliability.

The solution to this problem is the `end-end principle`. We allow unreliable steps (e.g., the network layer is best effort), but the destination end system checks the received data and tells the source end system to retry on failure.

1.4 How do we characterize communication and its performance?

A network connection is characterized by its `delay`, `loss rate`, and `throughput`.

1.4.1 Sources of Network Delays

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

- Transmission delay
- Propagation delay
- Processing delay (tends to be very small)
- Queuing delay

Those add up to the `total delay`.

> The `transmission delay` is the amount of time required to push all bits onto the link:
>
> $$\text{Transmission Delay } [\text{sec}] = \frac{\text{packet size } [\backslash\# \text{ bits}]}{\text{link bandwidth } [\backslash\# \text{ bits/sec}]}$$

> The `propagation delay` is the amount of time required for a bit to travel to the end of the link:

$$\text{Propagation Delay } [\text{sec}] = \frac{\text{link length } [\text{m}]}{\text{propagation speed } [\text{m/sec}]}$$

The `queuing delay` is the amount of time a packet waits in a buffer to be transmitted on a link. It is the hardest to evaluate and is characterized with statistical measures.

We introduce the following notations and terms:

- Average packet arrival rate: a $[\text{packet/sec}]$
- Transmission rate of outgoing link: R $[\text{bit/sec}]$
- Fixed packet length: L $[\text{bit}]$
- Average bits arrival rate: La $[\text{bit/sec}]$
- `Traffic intensity` : $\frac{La}{R}$

When the traffic intensity is > 1, the queue will increase without bound, and so does the queuing delay.

In practice, queues are not infinite. There is therefore an *upper bound on the queuing delay* which is given by $\frac{NL}{R}$, where N is the number of packets the queue can fit at once.

The `throughput` is the rate at which a host receives data:

$$\text{Average Throughput } [\backslash\#\text{ bits/sec}] = \frac{\text{data size } [\backslash\#\text{ bits}]}{\text{transfer time } [\text{sec}]}$$

To compute the throughput, one has to consider the `bottleneck link` . The throughput will be equal to the the transmission rate of the slowest link.

2. Application Layer

2.1 Domain Name Service (DNS) - How do we name and discover services?

The Internet has one global system for *addressing hosts* (IP) and *naming hosts* (domain names), called the `Domain Name Service (DNS)`. DNS provides a mapping between domain names and IP addresses.

We might map names to more than one IP for `load-balancing` or vice-versa, map an IP to more than one name to `reuse infrastructure`.

2.1.1 Naming Structure

Web addresses are `hierarchical`. We read the from right to left.
At the top sits the `Top Level Domain (TLD)`, such as .com, .org, .net, .ch, etc. `Domains` such as .epfl, .ethz, .nzz, etc. are subtrees of the TLD. A `name` such as inf.ethz.ch represents a *leaf-to-root path* in the hierarchy.

2.1.2 Management

The DNS system is is administrated over a hierarchy of authority over names:

- Root servers are managed by the IANA.
- TLDs are managed by private or non-profit organizations (for example, .ch is managed by the *Swiss Education & Research Network*).
- Domains managed by ISPs or local organizations, such as the ETH Zürich Informatikdienste ICT-Networks for .ethz.

2.1.3 Infrastructure

There are 13 root servers, named from a to m, which are managed professionally and serve as the root. Instances of the k-root server are hosted in more than 75 locations worldwide, two of them in Switzerland (Zurich and Geneva).

To `scale` root servers, operators rely on `BGP anycast` (Internet's routing protocol). This enables seamless replication of resources by finding the shortest-paths. If several locations announce the same prefix, then routing will deliver the packets to the "closest" location.

To ensure `availability`, each domain must have at least a primary and secondary DNS server. This has the following advantages:

- It ensures name service availability.
- It allows DNS queries to be load-balanced.
- On a timeout, clients can use an alternate server.

In order for DNS lookups to work, we have to ensure that:

- Each nameserver knows the address of the root server.
- Each root server knows the address of all TLD server.
- Each TLD server knows the address of all its domains.

2.1.4 Resource Records

The DNS server stores resource records that are tuples of the form $(name, value, type, TTL)$. We show the different types of records in the following table:

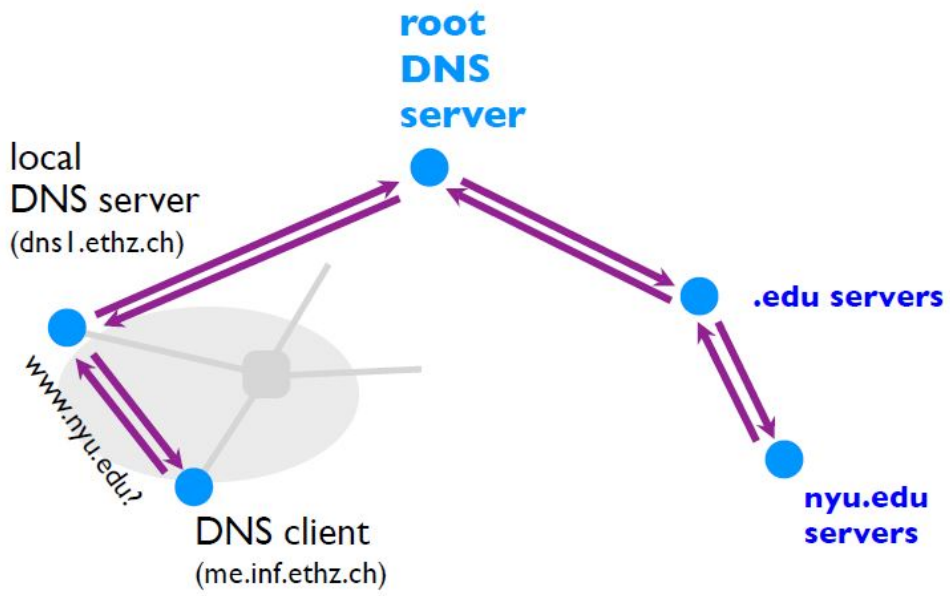
Record Type	Name	Value
A	hostname	IP address
NS	domain	DNS server name
MX	domain	Mail server name
CNAME	alias	Canonical name
PTR	IP address	Corresponding hostname

2.1.5 DNS Resolution

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

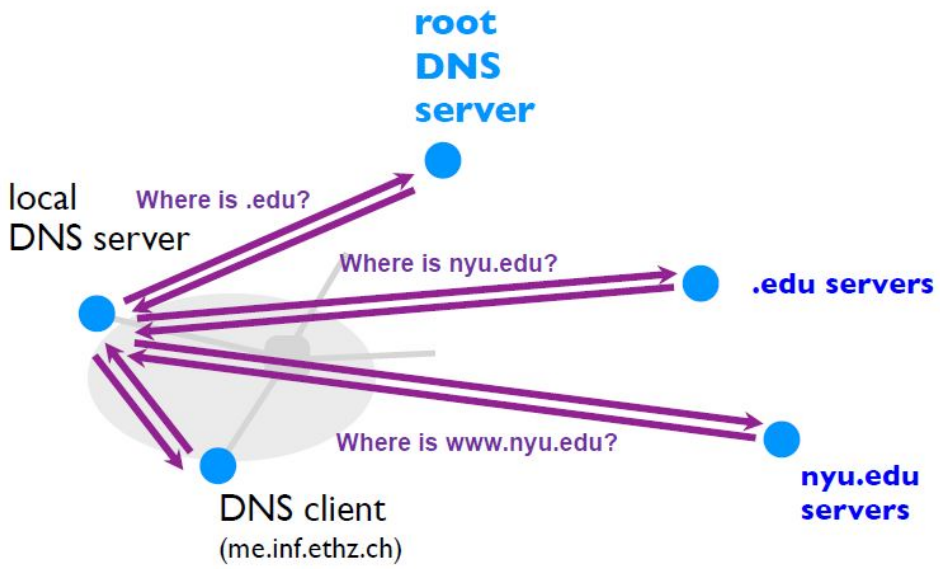
Recursive

The client offloads the task of resolving to the next server. This is never (exclusively) used in practice, root servers to actually not allows recursive queries on them.



Iterative

The DNS client sends a query to the local DNS server. The local DNS server then handles the rest, but each sever it queries returns the information it has directly back to the local DNS server.



2.1.5 Caching

TO reduce resolution times, DNS relies on `caching`. DNS server cache responses to former queries. Authoritative servers associate a lifetime to each record, the so called `Time-To-Live (TTL)`. The DNS records can only be cached for TTL seconds, after which they must be cleared.

Computer Networks - Lecture 5 & 6

- Author: Ruben Schenk
- Date: 01.06.2021
- Contact: ruben.schenk@inf.ethz.ch

2.2 The Web - How do you see weather.com?

THe WWW as we know it consists of three major components:

- *Infrastructure*: Client/Browser, servers, proxies, etc.
- *Content*: Objects, such as files, pictures, videos, etc, organized in websites, which is a collection of objects.
- Implementation: URL, HTTP, etc.

2.2.1 URL

A `Uniform Resource Locator (URL)` refers to an Internet resource and is of the form:

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

The URL consists of the following components:

- `protocol`: HTTP(S), FTP, SMTP, etc.
- `hostname`: DNS name or IP address.
- `port`: Defaults most often to protocols standard (HTTP:80 and HTTPS:443).

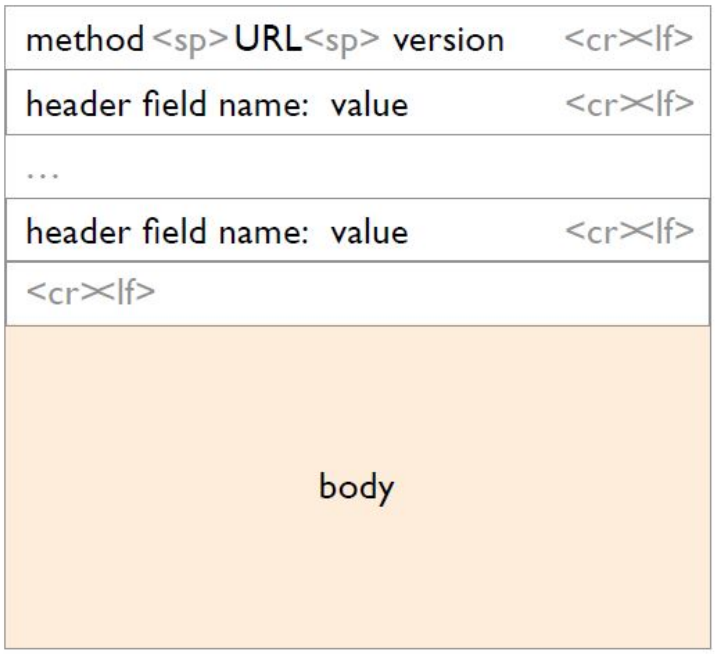
- `directory_path/resource` : Identify the resource at the destination.

2.2.2 HTTP

The `Hypertext Transfer Protocol (HTTP)` is a rather simple *synchronous request-response protocol*. It is layered over a bidirectional byte stream (almost always TCP), text-based (ASCII) and stateless, which means it maintains no information about past client requests.

HTTP sends two kinds of messages:

HTTP Request Message



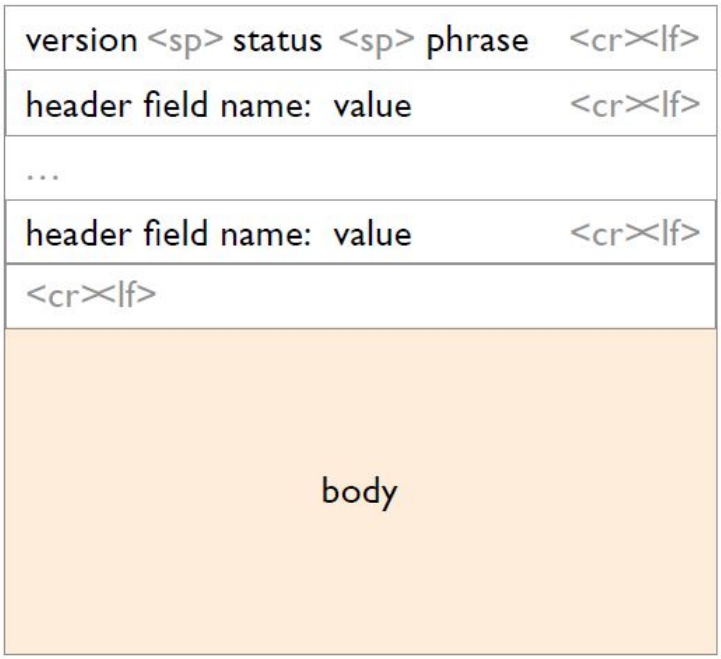
The request header specifies a `method` field, which can be one of the three following values:

- `GET` : Requests the server to return a resource.
- `HEAD` : Same as the GET method nut only asks for the header of the resource.
- `POST` : Requests that the server accepts provided data.

The request `header fields` are of variable lengths, but still human readable. They can hold the following information:

- Authorization info.
- Acceptable document types/encodings.
- Senders.
- If-Modified-Since.
- User Agent, i.e., client software
- etc.

HTT Answer Message



The response message holds a 3-digit `status` fields which can be one of the following values:

3-digit response code	status
$1XX$	informational
$2XX$	success
$3XX$	redirection
$4XX$	client error
$5XX$	server error

For example, the error codes 404 and 505 correspond to the reason phrase *Not Found* and 200 corresponds to *OK*.

Answer messages might include the following additional header fields:

- Location
- Content encoding
- Content length
- Content type
- Last-Modified
- etc.

2.2.3 Cookies

HTTP is stateless, however, some applications (like E-Commerce) need or want some kind of information about the previous requests from the same client. That is what `cookies` are used for. They store a state of the client on behalf of the server.

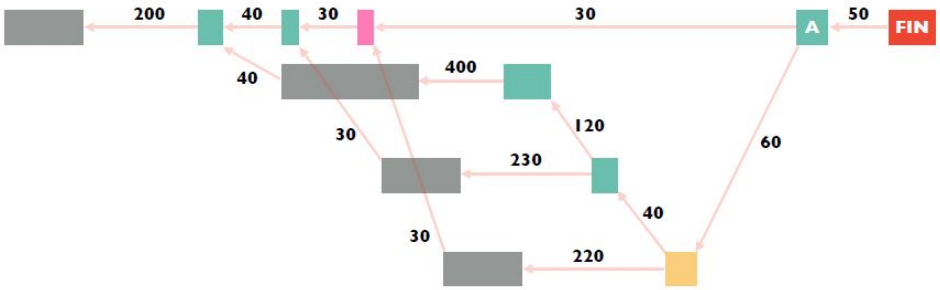
2.2.4 Speeding up the Web

Webpages today have very complex dependencies between different files such as html files, css files, JS files etc.

We can estimate the `load time` of a webpage by modelling the above mentioned dependencies as a DAG. Nodes are tasks (such as loading dependencies) and arcs indicate a "must-happen-before" dependency. The weights of the arcs represent the times it takes to complete the preceding task. Finally, we can model the load time as the `critical path (time)`:

1. Sort nodes topologically.
2. Process tasks in reverse order: Each task's finish time is the $\max$ over the tasks it depends on plus the time it takes to complete it.

Example: We might as what the longest A-start distance is:



This can be determined by $\max\left(d_{\text{pink}+30}, d_{\text{yellow}}+60\right)$ and then recursively determining the same for d_{pink} and d_{yellow}.

There are a lot of possible ways to `speed up` Web browsing, such as:

- Simplify, restructure, and redesign Web pages.
  - USe efficient image codes.
  - Inline JSS and CSS.
- Use faster computing devices.
- Increase network bandwidth.
- Make the networks RTTs smaller.
- etc.

Two other approaches for speeding up Web browsing are:

Simplifying Network Protocols

A naive HTTP connection opens a TCP connection for each of n objects. This requires about $2n$ RTTs. We introduce several different approaches to solve this problem:

- We might use `multiple parallel` (M) TCP connections, thus only needing $\frac{2n}{M}$ RTTs.
- We might use a `persistent connection` across multiple requests which leads to $n+1$ RTTs.
- We might `pipeline requests` in an asynchronous way and pack them into batches that fit into one TCP segment. For small segments this uses only 2 RTTs.

Caching

We might cache often used content at a server. HTTP allows for `conditional requests`, that is, a client conditionally requests a resource using the "if-modified-since" header in the HTTP requests.
The server compares this against the "last modified" time of the resource and returns either *Not Modified* or *OK* with the latest version.

We may perform caching and one of the following three different locations:

- Client: Browser cache.
- Close to the client: Forward proxy or via a Content Distribution Network (CDN), managed by the ISPs.
- Close to the destination: Reverse proxy.

2.2.5 Replication

The idea behind `replication` is to duplicate popular content all around the globe. This has the following effects:

- Spreads the load on sever, for example, across multiple data-centers.
- Places content closer to clients.
- Helps to speed up some uncachable content.

`Akamai` is one of the largest CDNs in the world. It used a combination of `pull caching`, which is a direct result of client requests, and `push replication`, which is done when high access rates are expected.

2.3 Internet Video - How does video streaming work?

The general goal for internet video is to have the best quality without experiencing load times.

2.3.1 End-End Workflow

We `encode` a video stream into different bitrates offering different levels of qualities and resolutions. The content is then *replicated* using a CDN. Finally, a video player will pick bitrates adaptively by estimating the connection's available bandwidth.

The video is not fetched as an entire file but rather in chunks of some short time (maybe 1 second pieces). This way the bitrate can be adapted as the available bandwidth changes.

One big problem is, that estimating the bandwidth is really hard. Available bandwidth often has large variation.

- Author: Ruben Schenk
- Date: 01.06.2021
- Contact: ruben.schenk@inf.ethz.ch

3. Transport Layer

3.1 What do we need in the transport layer? (Principles)

What do we actually need in the transport layer?

- Network should be kept minimal, i.e., easy to build, broadly applicable
- We need a global `best-effort` packet delivery service
- Applications should be restricted to app-specific functionality
- We need data delivery to the correct application, which means that the transport needs to demultiplex incoming data
- We need files or byte-streams abstractions for applications
- Reliable transfer if needed
- No overloading at the receiver and at the network

3.2 How do we build reliable transport? (Reliable Transport)

3.2.1 Principles of Reliable Transport

Since the Internet is an unreliable environment, packets may get *lost*, *corrupted*, *reordered*, or even *duplicated*.

> A `reliable transport protocol` should enable communication with the following properties:
>
> - *Correctness*: Packets should be received in the same order and without any gap
> - *Timeliness*: It should minimize the time until data is transferred.
> - *Efficiency*: It should minimize the use of bandwidth by not sending too many packets.

We might define the `correctness` of a reliable transport design the following way:

- A reliable transport design is `correct` if a packet is *always resent* if the previous packet was lost or corrupted. A packet *may be resent* at other times.

We can state the following equivalent condition:

- A reliable transport system is correct $\Leftrightarrow$ The system resends all lost or corrupted packets.

3.2.2 First Approach to Designing a Reliable Transport Protocol

Our first approach to designing a reliable transport protocol starts the following way:

```
Alice:

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

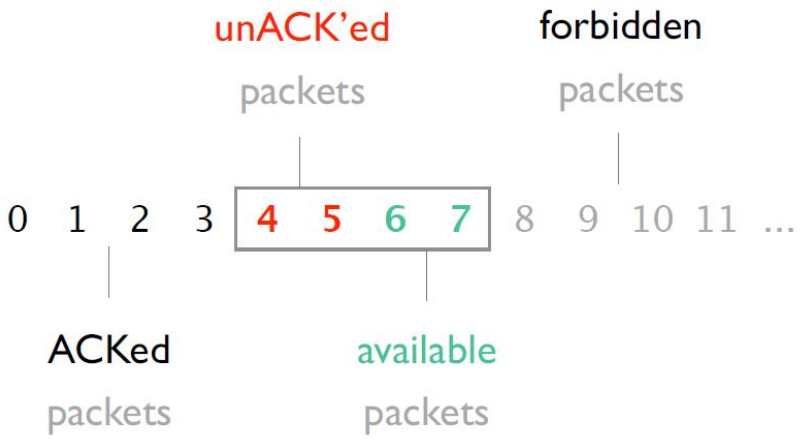
```
        upon timer going of:
            if no ACK received:
                send_packet(word);
                reset_timer();

        upon receiving ACK:
            pass;

Bob:

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

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

3.2.3 Improvements to our Design

We might improve the above idea in the following two ways:

- We can improve *timeliness* by sending multiple packets at the same time.
  - We need to add a `sequence number` inside each packet.
  - We need to add `buffers` to the sender and receiver.
    - Sender stores packets that were sent and not yet acknowledged.
    - REceiver stores out-of-sequence packets that he received.
- We can introduce *flow control* via a `sliding window`. This ways the receiver wont be overwhelmed when we send multiple packets.
  - The sender keeps a list of sequence numbers it can send, known as the `sending window`.
  - The receiver keeps a list of acceptable sequence numbers, known as the `receiving window`.
  - The sender and receiver negotiate the window size, it must be that $\text{sending window} \leq \text{receiving window}$.

The following figure shows an example of the *senders* view with a window composed of 4 packets:



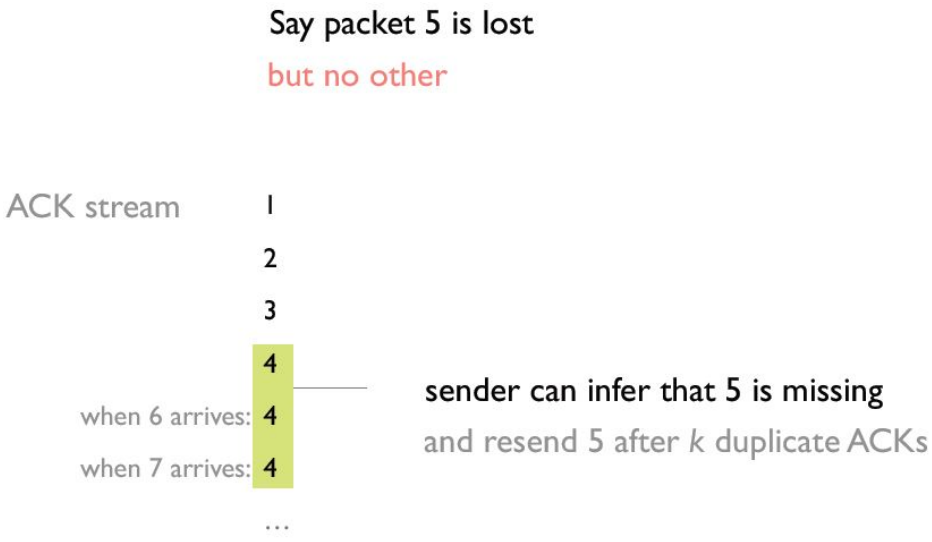
The timeliness depends on the size of the sending window. But how do we determine the "perfect" size for it? One way to answer this question is with the `bandwidth-delay-product (BDP)`:

- If Alice and Bob were connected by a link with a bandwidth of X Mbps and a RTT of d seconds, then the window size should be $X \cdot d$ Mb to maximize timeliness.

3.2.4 Receiver Feedback

There are three different ways me may encode feedback in ACKs:

- Individual ACKs: received 1, received 2, received3, received 5
  - Pros: We know the fate of each packet, we have a simple window algorithm, it is not sensitive to reordering.
  - Con: Loss of an ACK packet causes unnecessary retransmission.
- `Cumulative ACKs`: received up to 3
  - We trigger a resend after k duplicate ACKs as seen in the figure below. However it is not exactly clear what to resend (only the missing ACK? Or everything after it too?)
- Full information ACKs: received up to 3 and received 5
  - Fixes the problem of cumulative ACKS since it makes missing packets explicit.



3.2.5 Fairness

`Fairness` means that when n entities use the transport mechanism, the available bandwidth is allocated in a fair manner. We might use an equal-per-flow, which means we divide the available bandwidth evenly over each data stream, however this is not fair per se.
A universally agree upon minimal goal is to avoid `starvation`, which can be reached with equal-per-flow.

One approach is the max-min fair allocation:

> The `max-min fair allocation` is such that the lowest demand is maximized, after that the second lowest is maximized, and so on.

The max-min fair allocation can easily be computed the following way:

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

3.2.6 Corruption, Reordering, Delay and Duplication

Corruption

Dealing with `corruption` is easy. We simply rely on a checksum and treat corrupted packets as lost.

Reordering

The problem of `reordering` depends on the type of ACKs used:

- Individual: No problem
- Full feedback: No problem
- Cumulative: Can lead to creating duplicate ACKs

Delays

Long `delays` can lead to useless timeouts, for *all* designs.

Duplicates

Packet `duplicates` can lead to duplicate ACKs whose effects will depend on the type of ACKs used:

- Individual: No problem
- Full feedback: No problem
- Cumulative: Same problem as with reordering

3.2.7 Go-Back-N Protocol

The `Go-Back-N (GBN)` protocol is a simple sliding window protocol using cumulative ACKs:

- Principle: Receiver should be as simple as possible.
- Receiver: Delivers packets in-order to the upper layer. For each received segment, the receiver ACKs the last in-order packet delivered (cumulative).
- Sender: Uses a single timer to detect loss, resets the timer at each ACK. Upon a timeout, the sender resend all W packets, starting with the lost one, where W is the size of the sliding window.

3.2.8 Selective Repeat

The `Selective Repeat (SR)` protocol avoids unnecessary retransmissions by using per-packet ACKs.

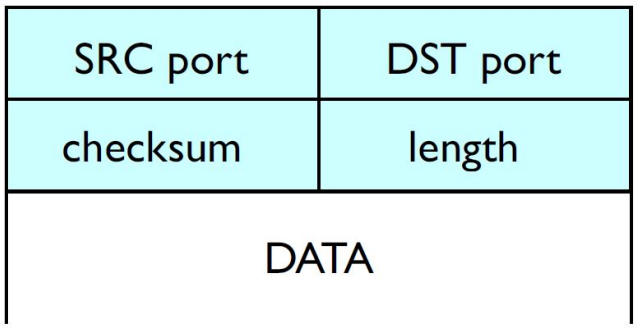
- Principle: Avoids unnecessary retransmissions
- Receiver: Acknowledge each packet, in-order or not, buffer out-of-order packets.
- Sender: Uses a per-packet timer to detect loss. Upon a loss, only the lost packet is resent.

3.3 How does the Internet's transport work?

3.3.1 UDP

`UDP` is a simple extension of IP that only allows for data delivery to the correct application (optionally with checksums for corruption detection).

A UDP packet simply consists of 4 header fields and one variable length data field, as seen in the figure below:



UDP is thus an unreliable, connectionless protocol for data transfer. Although there is no recovery from losses, reordering, etc., UDP is very good for certain types of application like DNS, Gaming or VoIP, due to the following reasons:

- There is no connection establishment delay like in TCP (3-way handshake)
- The small header leads to a small packet overhead which results in faster delivery.
- Since there is no connection state, no timers, etc., UDP provides better scalability.

3.3.2 TCP

> `TCP` is a *connection-oriented, reliable byte-stream* transport service.

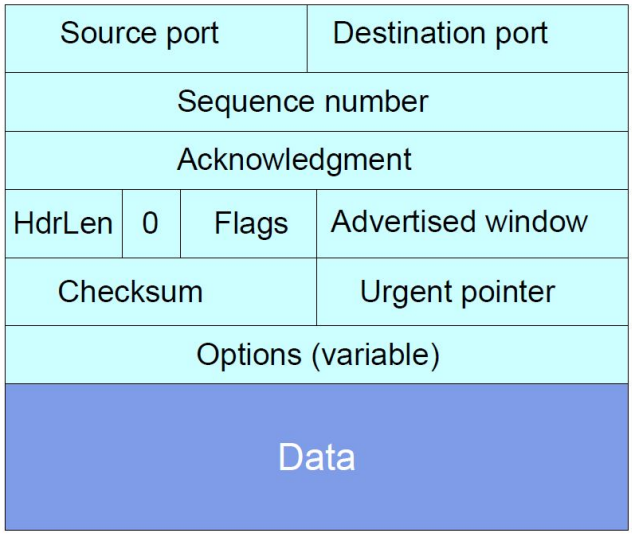
Reliability requires keeping the state at both the sender (in the form of timers and send buffers) and at the receiver (in the form of a receive buffer). Each byte-stream is called `connection/session` and has their own connection state.

TCP Design Choices

- **ACKs**: TCP uses byte sequence number and cumulative ACKs
- **Checksums**: TCP does checksum
- **Timeout/Retransmission**: Retransmission are based on timeouts and duplicate ACKs. Timeouts are based on an estimate of the RTT.
- **Sliding Window Flow Control**: Allows for W contiguous bytes to be in transfer.
- **Timer**: When a timer for a packet goes off, we resend the packet and double the timeout period. On three duplicate ACKs we can initiate a `fast retransmit`.

TCP Header

In order for a TCP packet to be sent over the network, it has to be wrapped into an IP packet, which cannot be bigger than the `maximum transmission unit (MTU)` (e.g. 1500 bytes for Ethernet).
A TCP packet consists of a `TCP header` of at least 20 bytes, shown in the figure below, and the TCP segment, which contains the data. Thus, the TCP segment is at most `maximum segment size (MSS)` bytes long, where $\text{MSS} = \text{MTU} - (\text{IP header size}) - (\text{TCP header size})$.



ACKing and Sequence Numbers

If the sender sends a packet containing B bytes and starting at sequence number X, then the bytes in the packet are $X,\ X+1, \ldots,\ X+B-1$.
Upon the receipt of the packet, the receiver sends an ACK:

- If all data prior to X was already received, then the ACK acknowledges $X+B$, since $X+B$ is *the next expected byte*.
- If the highest contiguous byte received is Y (smaller than X), then the ACK acknowledges $Y+1$, even if it has been ACKed before.

The `sequence number` field thus holds the starting byte offset of the data carried in the segment. The `acknowledgment number` field denotes what byte is expected next.

Sliding Window Flow Control

The `Advertised Window` field holds the number W of bytes which can be sent beyond the *next expected byte*. The receiver uses this field to prevent the sender from overflowing its buffer since it limits the number of bytes the sender can have in flight.

Transfer Speed

We finish our discussion of TCP with a simple example of computing the transfer speed. Assume the following:

- W in bytes is the sliding window size, assumed to be constant
- RTT in seconds is the round-trip-time, assumed to be constant
- B in bytes/second is the bandwidth of the link

We distinguish two cases:

- $W/RTT < B$, then the transfer speed is W/RTT
- Otherwise, the transfer speed is B

3.3.3 Connection Establishment

To establish a TCP connection, each host generates its `ISN (initial sequence number)` and then they exchange it in a so called `3-way handshake`:

1. Host A sends a `SYN` packet (with the SYN flag set) and its ISN in the sequence number field.
2. Host B returns a `SYN ACK` packet (with the SYN and ACK flag set) and its ISN in the sequence number field. The acknowledgment number is set to A's ISN + 1.
3. Host A sends an ACK to acknowledge the SYN ACK and can now start sending data.

If the SYN packet gets lost, no SYN ACK will arrive and thus the SYN packet will be simply retransmitted. The only problem with this is that it's totally unclear how far away the receiver is and it therefore is hard to set a timeout period. The default wait is 3 seconds.

3.3.4 Connection Teardown

Normal Termination - One Side at a Time

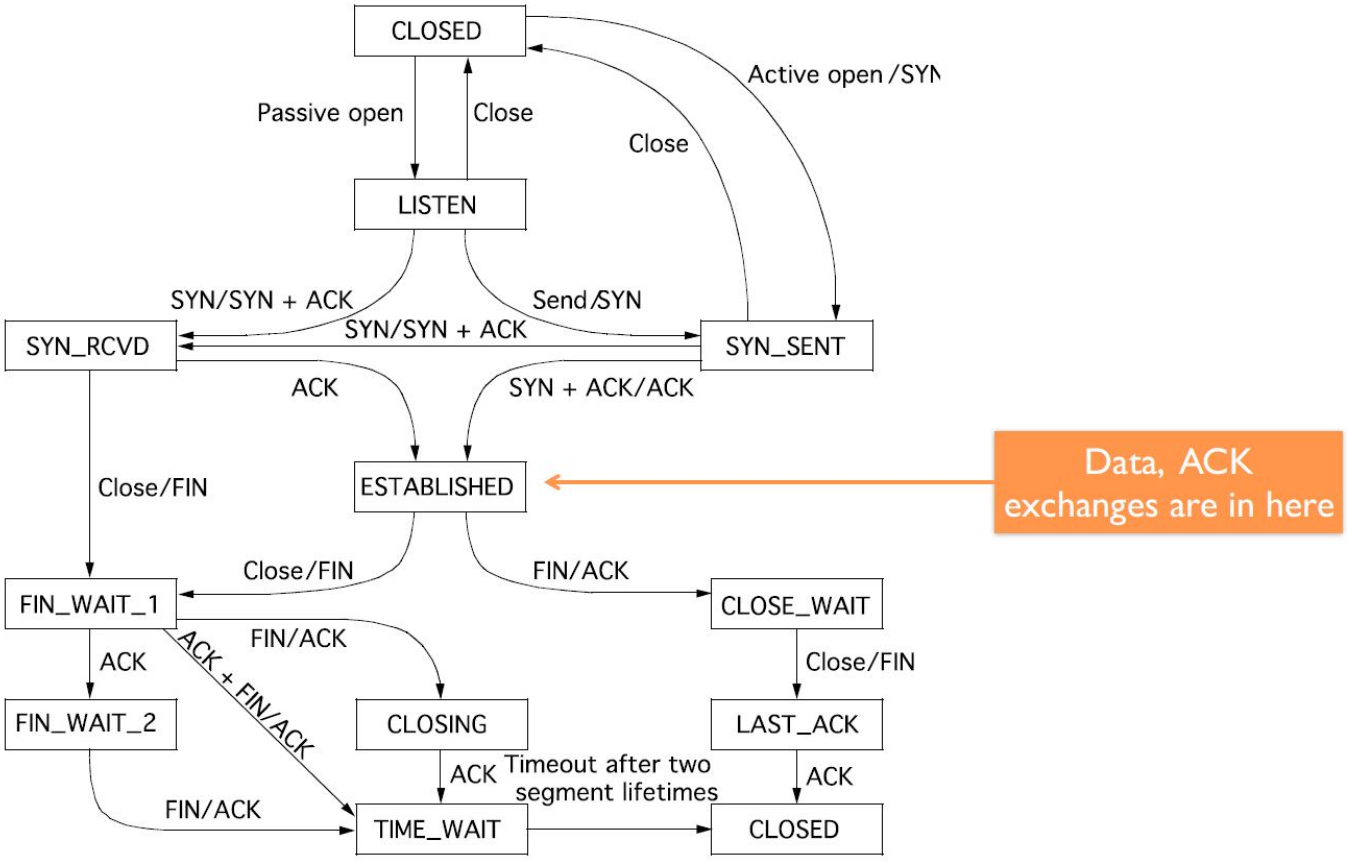
Host A sends a packet with the `FIN` flag set and waits to receive an ACK from host B. As soon as that happens, A closes its side of the connection, the connection is now *half-closed*. A can still receive bytes from B.
If B sends a FIN as well, the connection is fully closed as soon as A sends back an ACK.

Normal Termination - Both Together

Same as before, but when A sends its FIN packet, B responds with both FIN and ACK set. A then sends back a final packet with ACK flag set and the connection is closed.

3.3.5 TCP State Transition Diagram

The following figure shows a summary of the TCP state transitions:



Computer Networks - Lecture 9 & 10

- Author: Ruben Schenk
- Date: 01.06.2021
- Contact: ruben.schenk@inf.ethz.ch

3.3.6 Timeouts & Retransmissions

TCP resets a timer whenever new data is ACKed, not on a duplicate ACK. Upon a timeout, the packet containing the next byte, i.e., the seqNo of the ACK packet.

The problem here is that setting the timeout value is hard. Choosing it too short will result in duplicate packets, choosing it too long will result in inefficiencies. In order to set the timeout value, we need some estimate of the connections RTT.

> The `Kern-Partridge Algorithm` gives us an idea on how to estimate the RTT. We start with measuring the SampleRTT for original transmissions, that is, not for retransmissions. We are then able to estimate the RTT the following way:
>
> $$\text{SampleRTT} = \text{AckRcvdTime} - \text{SendPacketTime}$$
> $$\text{EstimatedRTT} = \alpha \times \text{EstimatedRTT} + (1 - \alpha) \times \text{SampleRTT}$$
>
> for $\alpha = 0.875$. From this, we determine the `retransmission timeout (RTO)` by

> $\mathrm{RTO} = 2 \times \mathrm{EstimatedRTT}$
>
> We might use an `exponential backoff`, that is, upon a timeout we double the RTO. Every time a new measurement comes if, we collapse the RTO back to $2 \times \mathrm{EstimatedRTT}$.

3.3.7 Congestion Control

`Congestion control` aims at solving three problems:

- *Bandwidth estimation*: How to adjust the bandwidth of a single flow to the bottleneck bandwidth?
- *Bandwidth adaptation*: How to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth?
- *Fairness*: How to share bandwidth fairly among flows, without overloading the network?

It is important to note that congestion control differs from flow control, but botha re provided by TCP though:

- `Flow control`: prevents one fast sender from overloading a slow receiver.
- `Congestion control`: Prevents a set of senders from overloading the network.

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

- `Receiving Window (RWND)`: How many bytes can be sent without overflowing the receiver buffer?
- `Congestion Window (CWND)`: How many bytes can be sent without overflowing the routers?

From those two windows follows the `Sender Window`. determined by $\min(\mathrm{CWND}, \mathrm{RWND})$.

Congestion Detection

There are essentially three ways to detect congestion:

- The network could tell the source but the signal itself could be lost on the way.
- We could measure the packet delay but the signal probably will be noisy.
- We could measure the packet loss. This is a fail-safe signal that TCP already has to detect.

If we detect duplicated ACKs, we have a `mild congestion signal` and if we have a timeout, we have a `severe congestion signal`.

Reacting to Congestion

TCP's approach is to gently increase when not congested and to rapidly decrease when congested.

We start with the `Slow Start Phase`, that is, we set the CWND to 1 and increase it by one for each ACK we receive. However, once we have a rough estimate of the bandwidth, we need a more gentle adjustment algorithm.

The two possible options are:

- `Multiplicative Increase or Decrease (MID)`, i.e., $\mathrm{CWND} = a \cdot \mathrm{CWND}$.
- `Additive Increase or Decrease`, i.e., $\mathrm{CWND} = b + \mathrm{CWND}$.

TCP implements additive increase multiplicative decrease (AIMD): After each ACK, we increment the CWND by $1/\mathrm{CWND}$. Per RTT, the increase is therefore at most 1. One might as the question on when the sender leaves the slow-start phase and starts AIMD. We introduce a `slows start threshold`, and adapt it as a function of congestion. On a timeout, we set $\mathrm{SSTHRESH} = CWND/2$.

3.3.8 QUIC - Quick UDP Internet Connections

`QUIC` is the biggest transport change in the last 30 years. It is a transport layer protocol designed by Google to improve performance of HTTPS. Broadly explained, QUIC takes everything from the TCP header and puts it into the data field of a UDP packet.

- QUIC rolls in TCP and TLS connection negotiation. Therefore, a QUIC handshake is enough (instead of a separate TCP and a separate TLS handshake).
- Based on a cookie, QUIC allows for 0-RTT session resumption where a *secret* learned from a previous session is used to encrypt/decrypt messages.

3.4 Sockets: the application and the transport interface

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

Note: Socket are an OS abstraction and ports are a networking abstraction. In this context, they do not correspond to any physical port like on routers and switches, but are rather logical interfaces that one uses.

3.4.1 Ports

To determine which app (*socket*) gets which packets, `ports` act as a 16-bit transport layer identifier . A packet then carries source/destination port number is the transport header and the OS stores the mapping between sockets and ports.

The 16-bit address space is built up in the following way:

- Ports 0-1023 are *well known* and there is an agreement on which services run on these ports, e.g., SSh:22, HTTP:80, HTTPS:443, etc. A client using these services therefore knows the appropriate port and can listen on it directly.
- Ports 1024-65535 are *ephemeral* and ar given to clients ar random.

3.4.2 Multiplexing & Demultiplexing

A host receives IP datagrams with a source and destination IP address as well as a source an destination `port number`. The IP address and the port are then used to deliver the segment to the appropriate socket.

The mappings are as follows:

- For UDP (`SOCK_DGRAM`): OS stores (local port, local IP address) $\Leftrightarrow$ socket
- For TCP (`SOCK_STREAM`): OS stores (local port, local IP, remote port, remote IP) $\Leftrightarrow$ socket

This is due to the nature of the protocols. Since TCP is connection oriented, it needs to remember the source IP and port number of an incoming IP datagram, whereas UDP as a connectionless protocols need not remember that information.

4. Algorithms in Networking

4.1 Three example algorithmic problems in networking

4.1.1 Example 1: Traffic engineering

`Max-Flow-Problem` : Given a network, displayed as a directed graph, what is the maximum traffic from some endpoint S to another endpoint T ? Solved by determining the min-cut of the graph (which will be equal to the max-flow).

4.1.2 Example 2: Matchings and circuits

`Matchings` are often seen inside circuit switches that consist of micro-mirrors and we want to allow as many concurrent connections/circuits as possible.

4.1.3 Example 3: Shortest paths

Again, assume a network, shown as a directed graph, and trying to find a shortest path from some source node S to a destination node T.

4.2 Linear programming: a powerful, generic tool

4.2.1 What is a linear program?

A `linear program` is based on the following three definitions one needs to assign to define a linear program:

1. `Variables` : E.g., flows on edges, distance from start, etc. Must be real numbers, $x \in \mathbb{R}$
2. `Objective` : E.g., maximize flow, minimize distance, etc. Must be a linear combination of the variables.
3. `Constraints` : Flow on edge $\leq$ capacity of edge, etc. Must too be a linear combination of variables.

Computer Networks - Lecture 11 & 12

- Author: Ruben Schenk
- Date: 07.06.2021
- Contact: ruben.schenk@inf.ethz.ch

4.2.2 Previous algorithms as linear programs

Max-Flow

We can state the max-flow problem as a linear program in the following way:

- Objective: Maximize flow from s to t.
- Constraint: Obey edge capacities $c_{u,\,v}$.

In the linear-programming language, our constraints could look as follows:

- Variables: Flow $s \to t$: f, flow on edge $u \to v$: $f_{u,\,v}$
- Objective: Maximize f
- Constraints: Capacity $f_{u,\,v} \leq c_{u,\,v}$, flow conservation $\sum_{s \to v} f_{s,\,v} = f = \sum_{v \to t} f_{v,\,t}$ where s is the source and t is the sink, $\sum_{w \to u} f_{w,\,u} = \sum_{u \to v} f_{u,\,v} \quad \forall u \in V \setminus \{s,\,t\}$.

Max-Flow with multiple commodities

We are trying to maximize different flow f_x on the same graph/network. Our objective is therefore to maximize/minimize $\sum_x f_x$. We have the following constraints:

- Capacity: $\sum_x f_{x,\,uv} \leq c_{u\,v}$
- Flow conservation: in-flow = out-flow
  - $\sum_{w \to u} f_{x,\,wu} = \sum_{u \to v} f_{x,\,uv} \quad \forall u \in V \setminus \{u,\,d(x)\}$, where $d(x)$ is the destination of x (this constraint makes sure that *flows don't mix*).
  - $\sum_{x \to v} f_{x,\,xv} = f_x$
  - $\sum_{v \to d(x)} f_{x,\,vd(x)} = f_x$

Shortest Path

Our goal is to minimize an $s - t$ path length with given edge lengths $w_{u,\,v}$, i.e. minimize $\sum_{u,\,v} x_{u,\,v} w_{u,\,v}$.

We have the following constraints:

- Path must be connected: $\sum_{u \to v} x_{u,\,v} - \sum_{v \to w} x_{v,\,w} = 0 \quad \forall v \in V \setminus \{s,\,t\}$ ($= 1$ if $v = t$ and $= -1$ if $v = s$).
- $x_{u,\,v} \in \{0,\,1\}$, however in this case $x_{u,\,v} \in [0,\,1]$ is enough.

4.2.3 Integer Linear Programs

Constraints like $x \in \{0,\,1\}$ are not linear anymore. Whereas LPs are solvable in polynomial time, the general class if ILPs is NP-hard!

4.3 Exploiting randomness for networking

4.3.1 Balls-into-Bins Problem

Assume you run a popular network application. How do we distribute requests? Our goal is to keep response time uniformly low and we wish to have a uniform load at the servers.

- *round-robin*: Might not work since requests might follow a pattern such as every second request is a complex one. This way, every even numbered server is fucked.
- *send to the least loaded server*: Keeping track of the different loads of each server generates a big overhead and is quite complex to implement.

Our solution is to pick a server *uniformly at random*, similar to the `balls-into-bins` problem.

Recap: Let there be m balls and n bins, then $(P[X_{i,\,j}] = 1) = \frac{1}{n}$, which is the probability that two balls collide.

The expected number of collisions is therefore:

$$\mathbb{E}\Big[\sum X_{i,\,j}\Big] = \frac{1}{n}\binom{m}{2}$$

The expected maximum load on any bin is therefore:

$$O\Big(\frac{\ln n}{\ln \ln n}\Big)$$

which is actually not that good. We can improve this by choosing two bins at random and then selecting the onw with the smaller load. This improvement leads to the following expected load:

$$O\Big(\frac{\ln \ln n}{\ln 2}\Big)$$

which is exponentially better. We can extend this idea by choosing k instead of 2 bins and then choose the one with the smallest load.

We normally only use random distribution once for each session. We determine the server by `hashing` the user-id and taking the hash $\% \, n$. However this means, that upon failure of a server, we will reassign most sessions (since now we calculate $\% \, (n-1)$). We can prevent this with `consistent hashing`, where we hash the server-id's into the same *hash-space* as we do with the user-id's. Each requests will simply be mapped to the server with the next greater hash-value as the user-id of the request.

4.3.2 Membership testing & counting

With a `bloom filter` we use m bits of memory in table T with k hash functions $h_1,\, h_2, .. ,\, h_k$ in the hash range $\{1,\, 2, \ldots,\, m\}$. We are trying to represent n elements.

A `check for membership` occurs when we hash the value with all k functions and check whether all fields are 1, if not we can be certain that it isn't a member. If all fields are 1, then it is very likely that it is a member.

After n insertions, the probability that the t-th bit is still 0 is given by:

$$P[T(i) = 0] = \Big(1 - \frac{1}{m}\Big)^{kn} \simeq e^{\frac{-kn}{m}}$$

Thus, the probability of a match being a false positive is approximately

$$(1 - e^{\frac{-kn}{m}})^k$$

If we choose $k = \frac{m}{n}\log 2$ we can minimize the false positive probability, which then becomes 2^{-k}.

4.4 Distributed decision-making

Distributed settings are challenging since selfish player can cause large sub-optimalities. The best choice typically depends on the setting and may change over time.

Computer Networks - Lecture 13 & 14

- Author: Ruben Schenk
- Date: 07.06.2021
- Contact: ruben.schenk@inf.ethz.ch

5. Network Layer

The `network-layer protocols` are among the most challenging in the protocol stack. The network layer can be decomposed into two interacting parts, the `data plane` and the `control plane`.

The primary role of the network layer is deceptively simple - to move packets to appropriate output links from a sending to a receiving host. For that there are two functions:

- `Forwarding`: A packet arrives at some input link of a router and needs to be moved to an appropriate link. This one of the functions of the *data plane* and happens router-locally.
- `Routing`: Network layer must determine a route a packet takes as it flow from sender to receiver. This is a network-wide operation that is very complex and expensive and is part of the *control plane*.

5.1 Network Service Models

The `network service model` specifies what service the network layer provides to the transport layer and how it is implemented. We distinguish a `connectionless datagram service` like IP and a `connection-oriented virtual circuits service`.

Both models are implemented with `store-and-forward packet switching`:

- Routers receive complete packets, store them temporarily (if necessary) before forwarding it. Most of the time *statistical multiplexing* is used to share the link bandwidth over time.

The switching element provides an internal buffer for each output port to handle contention.

5.1.1 Datagram Model

Each (IP) packet contains a destination address. This is used by the router to forward each packet individually on different paths. For this, we consult the `forwarding table` which is keyed by the address and stores the next hop for each destination address.

The datagram model uses the `internet protocol (IP)` which is based on datagrams and carries source and destination address in each packet.

5.1.2 Virtual Circuit Model

The virtual circuit model uses circuit switching, but in a virtual sense: there is no bandwidth reservation but rather statistical sharing of links. There are three phases in a virtual circuit:

1. *Connection establishment*: Circuit is set up, i.e., path is chosen and the circuit information is stored in the routers.
2. *Data transfer*: Circuit is used, i.e., packets are forwarded along the path.
3. *Connection teardown*: Circuit is deleted, i.e., circuit information is removed from routers.

Each packet thus only needs to carry a short label that identifies the circuit. This label, however, has no global meaning and is link specific, such that at every router it needs to be translated. The forwarding tables are keyed by the circuits identifier and returns the output port the circuit uses.

5.1.3 Multi-Protocol Label Switching

ISPs use the virtual circuit switching technology by setting up circuits in their backbone ahead of time. When a packet enters the network, it adds a 4 byte MPLS label to the IP packet and removes it upon leaving the network.

The advantage MPLS offers are:

- Potential increase of switching speed since the IP header is not needed.
- MPLS provides the ability to forward packets along routes that would not be possible using the standard IP routing protocols.

The following table provides a short summary over the advantages and disadvantages of both network service models:

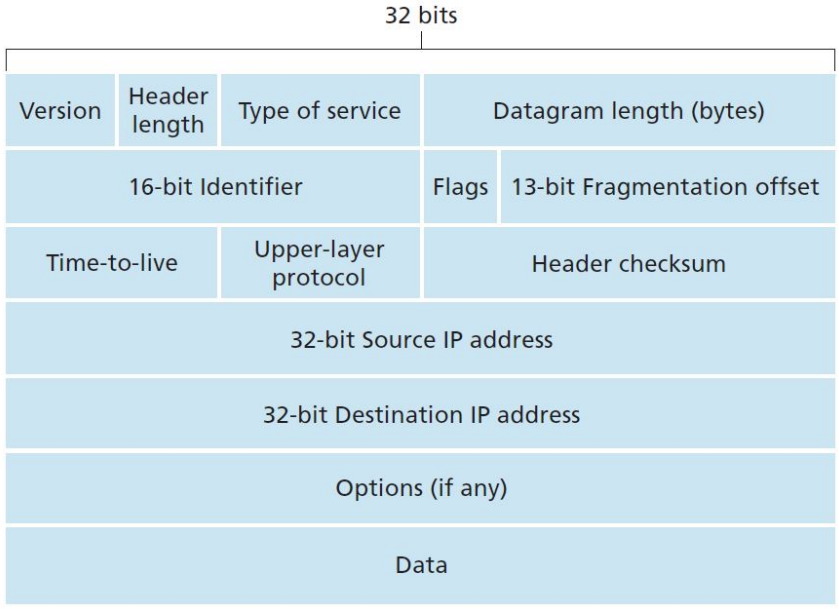
Issue	Datagram	Virtual Circuit
Setup Phase	Not needed	Required
Router state	Per destination	Per connection
Addresses	Packet carries full address	Packet carries short label
Routing	Per packet	Per circuit
Failures	Easier to mask	Difficult to mask
Quality of service	Difficult to add	Easier to add

5.2 Internet Protocol - Version 4 (IPv4)

5.2.1 IP Datagram

Most of the fields of a IPv4 datagram are self-explanatory, the rest is explained in short here:

- *Type of Service*: Allows to distinguish between different types of datagrams, e.g., real-time (VoIP) vs non-real time (FTP)
- *Datagram length*: Total length of datagram in bytes, at max 2^{16} bytes, but normally not larger than 1500 to fit into an Ethernet frame.
- *Fragmentation offset and flags*: Used for IP fragmentation.
- *Protocol*: Specifies the transport layer protocol (TCP or UDP).
- *Options* Allows the IP header to be extended but leads to processing overhead (not included in IPv6 anymore).



5.2.2 IP Addresses and Prefixes

> An `IPv4 address` is a 32-bit number written in *dotted-quad* notation `a.b.c.d` where a, b, c, and d are 8-bit numbers.

IP addresses are allocated in `prefixes`. Addresses in an L-bit prefix have the same top L bits. Thus, there are 2^{32-L} addresses in an L-bit. IP prefixes are written in IP address notation, e.g., `128.13.0.0/16` denotes the first address in the prefix ($L = 16$ in this case).

We say a prefix is *more specific* if it is longer, and hence has a smaller number of IP addresses. A *less specific* prefix has a shorter prefix and hence a larger number of IP addresses.

The first and last address of a prefix are typically not used since they have a special function:

- *Network Identifier*: First address in a prefix, e.g., `128.13.0.0` for `128.13.0.0/16`.
- *Broadcast Address*: Last address in a prefix, e.g., `128.13.255.255` for `128.13.0.0/16`.

Prefixes are also specified by using a `network mask`. Performing a logical AND on the two produces the prefix (i.e. the network identifier). Example: `255.255.255.0` is the network mask for a 24-bit prefix.

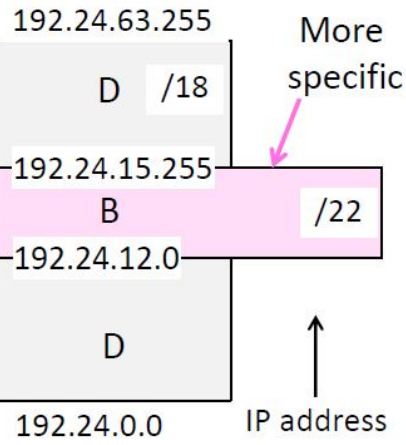
5.2.3 IP Forwarding

Each router uses a forwarding table that lists the next-hop address for *IP prefixes*. Note that this may lead to overlapping prefixes in the table, i.e., an entry might list a more specific prefix than another entry. This leads to the following rule:

> The `Longest Prefix Matching` forwarding rule is described as follows: For each packet, find the longest (most specific) prefix that contains the destination address and forward the packet to the next-hop router for that prefix.

Rule	Prefix	Next Hop
1	192.24.0.0/18	D
2	192.24.12.0/22	B

Destination	Matching rules	Next hop
192.24.6.0	1	D
192.24.14.32	1, 2	B
192.24.54.0	1	D



Host Forwarding

Since routers do all the routing, hosts have to send remote traffic to the nearest router. For that, they use a small forwarding table, consisting only of two entries: Note that `0.0.0.0/0` represents a `default route`. Any other prefix is more specific and is thus captured first by the longest prefix matching rule.

Prefix	Next-Hop Destination
My own network prefix	Directly send to that IP
`0.0.0.0/0`	Send to router

5.2.4 IP Helper Protocols

In order for IP forwarding to work, we still need to fill in some gaps, namely how, e.g., hosts get their IP addresses and how IP is mapped to link addresses.

Getting IP Addresses - Dynamic Host Configuration Protocol (DHCP)

When a node wakes up for the first time it doesn't know its IP address, the network prefix or the IP of the default router and the address of the DNS resolver. We use `DHCP` to automatically configure those addresses.

DHCP leases IP addresses to nodes and provides other important parameters, such as the network prefix, the `default gateway (local router address)` and various servers (DNS, time, etc.). DHCP is a client-server application that uses UDP ports 67 and 68 and can therefore snoop on these ports for messages. DHCP works as follows:

1. The client sends a $\mathrm{DISCOVER}$ broadcast message with the address bits all being 1, i.e. the IP address `255.255.255.255` and `ff:ff:ff:ff:ff:ff` for Ethernet/MAC address.
2. The server answers with an OFFER message, offering a few IP addresses as well as other options.
3. The client chooses one of the offers and requests with a $\mathrm{REQUEST}$ message.
4. The server responds with an ACK such that the client can use the configuration.

Renewing an existing lease is as easy as sending a $\mathrm{REQUEST}$ message and getting back an ACK.

Address Resolution Protocol - ARP

We consider the following problem: A node needs the link layer address to send a frame over a local link, but how does it get the destination link address from the destination IP address?

An ARP module is a sending host that takes any IP address *on the same LAN* as input, and returns the corresponding MAC address, meaning that it only resolves IP addresses for hosts and router interfaces that are on the same subnet. This works as follows:

1. The node sends a $\mathrm{REQUEST}$ broadcast message with an IP address.
2. The target that uses this IP address replies with its MAC address.

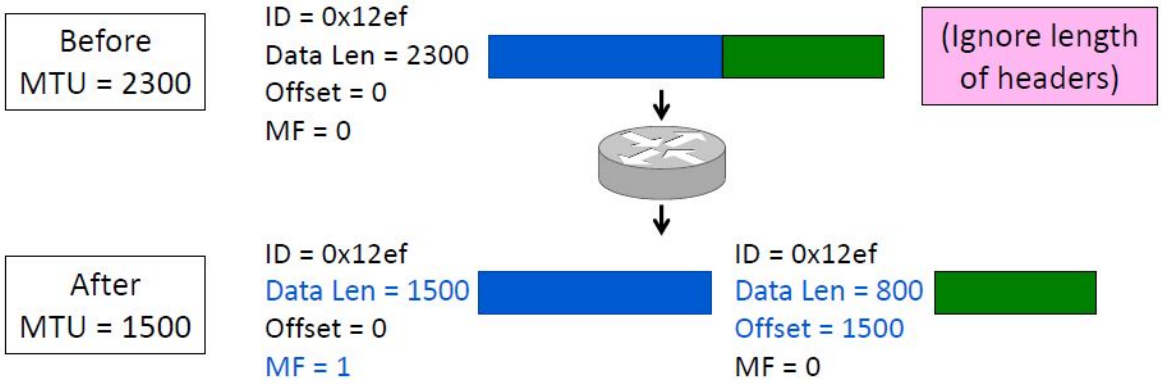
5.2.5 Packet Fragmentation and MTU

When connecting networks with different maximum packet sized we need to split up packets, or discover the largest size we can use.
We can thus either use `fragmentation` to split up large packets or use `discovery` to find the largest packet size that fits on the network path.

IPv4 Fragmentation

Routers fragment packets that are too large and the receiving host reassembles to reduce the load on routers. For that, there are multiple fields in the IP header (identification, flags, fragment offset, fragment length, etc.). The splitting procedure works as follows:

1. Break the data that is contained into pieces.
2. Copy the IP header into the previously split pieces and adjust the length.
3. Set the offset to indicate the position and set the MF flag on all pieces except for the last.



Path MTU Discovery

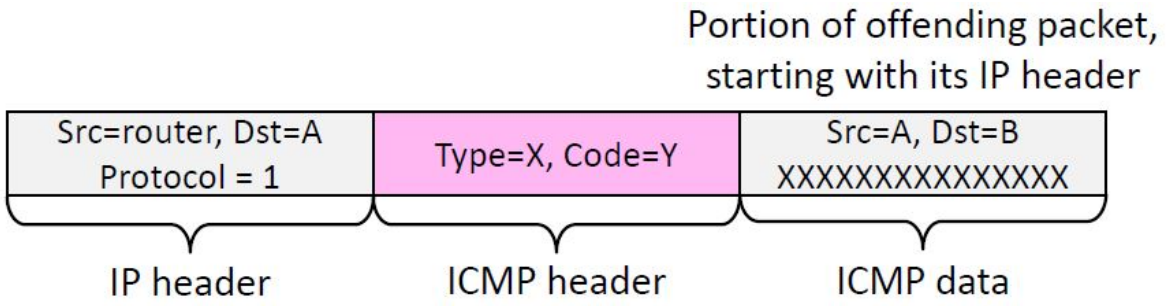
IP uses `path MTU discovery` today. The host tests the path with a large packet and the routers provide feedback. If the packet is too large, the router tells the host what size would fit. This is implemented in ICMP with the DF flag, such that ICMP can provide the necessary feedback messages and error messages in general, e.e., when something goes wrong during forwarding.

5.2.6 Internet Control Message Protocol (ICMP)

ICMP is a companion protocol to IP, they are implemented together and ICMP sits on to of IP.
IF the router encounters some error while forwarding, the ICMP sends back a report to the IP source address and discards the problematic packet.

ICMP Message Format

A ICMP message is carried in an IP packet and has a type, code, and a checksum. As a payload it often carries the start of the offending packet:



5.3 Network Address Translation (NAT)

The layering principle means that routers should not look beyond the IP header, e.g., the TCP segment is not looked at. However, in modern networks there are many `middleboxes` that sit inside the network but perform processing beyond what IP does on packets to add new functionality. Examples are NAT boxes, Firewalls or Intrusion Detection Systems.

5.3.1 NAT box

A `NAT box` connects an internal network to an external network. That way, many internal hosts can be connected to the external network using a single external IP address. This way, the home network looks like a single computer.

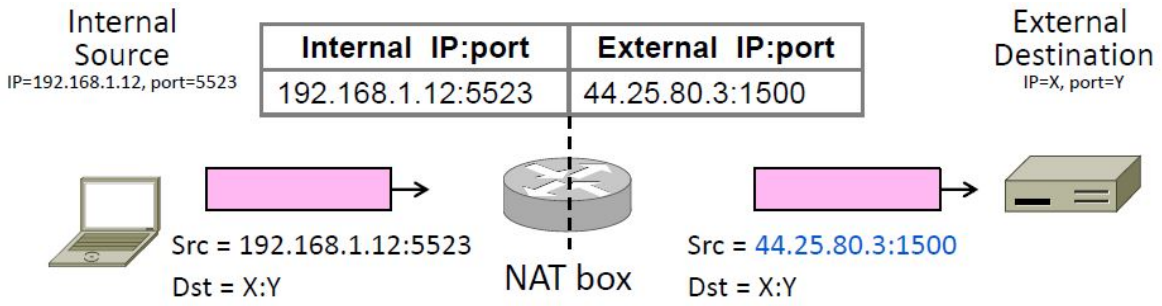
5.3.2 How NAT works

The NAT box keeps a table where it stores a mapping between internal and external (address, port) tuples.

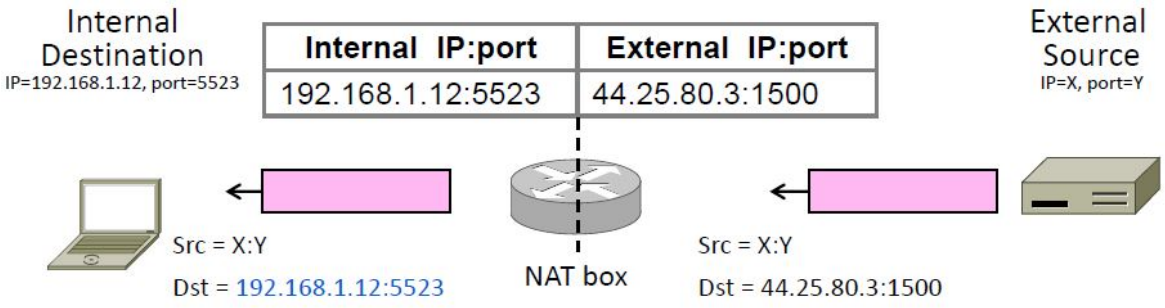
Internal IP : Port	External IP : Port
`192.168.1.12:5523`	`44.25.80.3:1500`
`192.168.1.13:1234`	`44.25.80.3:1501`
`192.168.2.20:1234`	`44.25.80.3:1502`

The mapping is created as the first packet from the client into the external network is being sent, i.e., when a new TCP connection is opened. If connections go to different destinations, it is possible to use the same port numbers, since returning packets will also contain the destinations IP address and thus make it able to distinguish the connection using the same port.

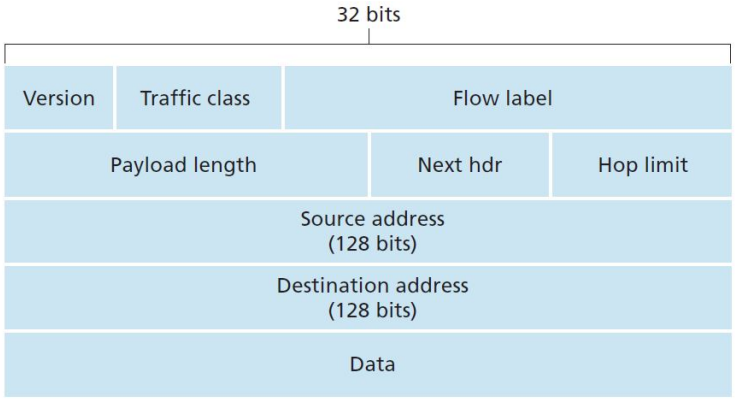
Sending a Packet



Receiving a Packet



5.4 IPv6



Some important changes in the header are:

- *Streamlined header processing*: No option fields mean fixed size of 40 bytes in the header, which allows for faster processing.
- *Flow labelling*: Allows for grouping of packets that belong to the same floe, e.g. packets in a video stream or for a high-priority user.
- *Traffic class*: Allows to give priority to certain datagrams within a flow from certain applications.

Computer Networks - Lecture 15 & 16

- Author: Ruben Schenk
- Date: 08.06.2021
- Contact: ruben.schenk@inf.ethz.ch

5.4.1 IPv6 Addresses

Since the IPv4 address space is running out and there is still room for different improvements there is IPv6, which features 128-bit addresses, which are denoted in 8 groups of 4 hexadecimal digits. Leading zeros are omitted and groups of zeros can be replaced by a second colon.

Example: `2001:0db8:0000:0000:0000:ff00:0042:8329` may be written as `2001:db8::ff0:42:8329`. Remember that we may only remove *one* groupe of consecutive zeros, e.g., `2001:0db8:0000:0000:1a12:0000:0000:1a13` may either be written as `2001:db8::1a12:0:0:1a13` or as `2001:db8:0:0:1a12::1a13`.

5.4.2 Tunneling

The goal of `IPv6 tunneling` is to allow IPv6 communication over IPv4. We want a *tunnel* to act as a single link across an IPv4 network. We encapsulate an IPv6 packet as payload into an IPv4 packet. The IPv6 is extracted as soo as it reaches an IPv6 link.

5.5 Routing

The goal for any `routing algorithm`, no matter which routing scheme it uses, is that it should obey the following properties:

- *Correctness*: Finds paths that work
- *Efficient paths*: The given path should be minimal for some metric
- *Fair paths*: The path doesn't starve any nodes
- *Fast convergence*: The path recovers quickly after changes
- *Scalability*: Works well as the network grows large

5.5.1 Shortest Path Routing (Dijkstra Algorithm)

To find a `shortest path` we do the following steps:

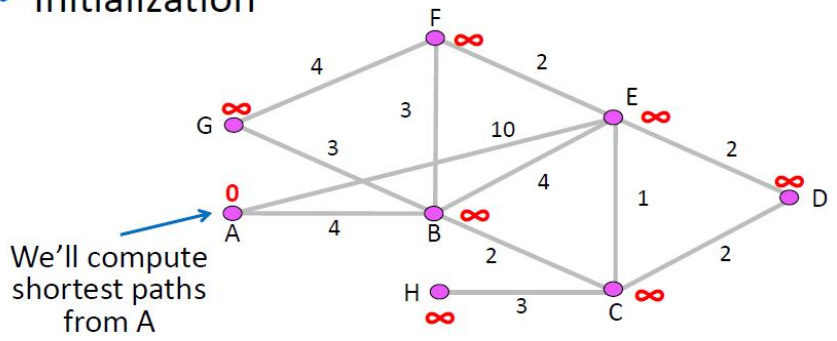
1. Assign each link a *cost* (distance).
2. Define the best path between each pair of nodes as the path that has the lowest total cost.
3. Pick randomly to break any ties.

One property when choosing the shortest path as described above, is that *sub-paths of shortest paths are also shortest paths*.

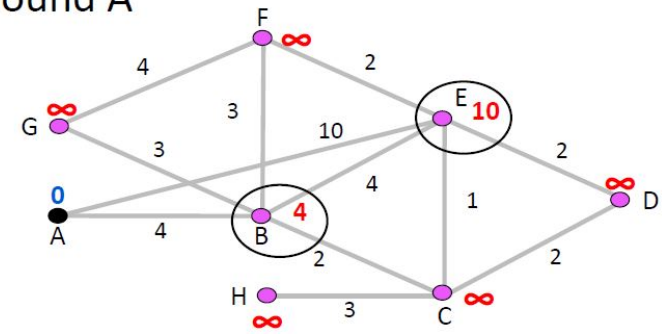
We furthermore define a `sink tree` of some node as the union of all shortest paths (i.e. the shortest path from each node) to the destination node.

The following figures show an example of how to use `Dijkstra's Algorithm`:

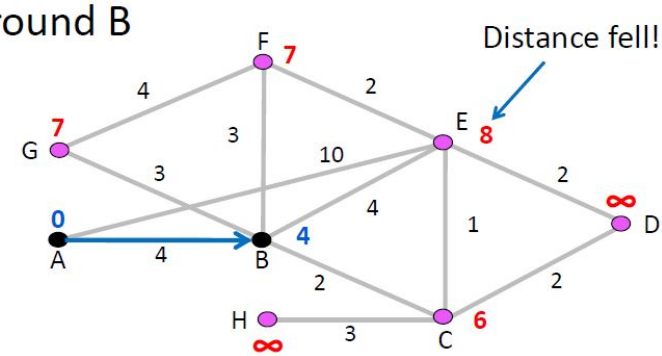
- Initialization



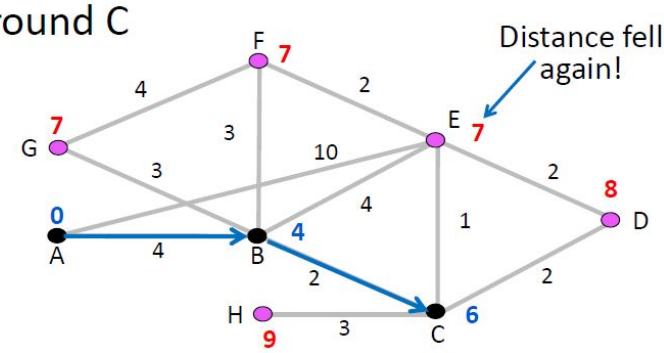
- Relax around A



- Relax around B

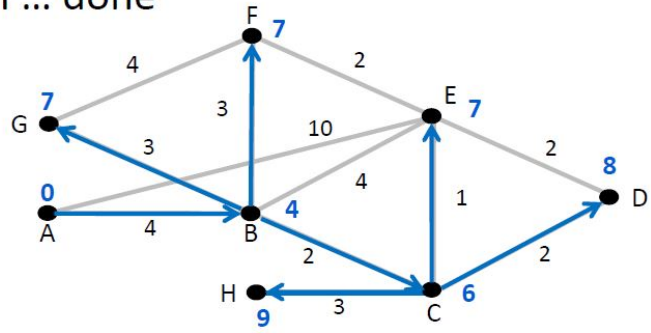


- Relax around C



(*Some intermediate steps are left out*)

- Finally, H … done



5.5.2 Hierarchical Routing

There are several key impacts of routing growth, listed below:

- Forwarding tables grow
- Routing messages grow
- Routing computation grows

Some techniques to scale routing are:

1. IP prefixes (route to block of hosts, not individual hosts)
    - We group hosts under an IP prefix and connect them directly to the router, this way there is only one entry needed for all hosts.
2. Network hierarchy (route to network regions)
    - The idea is to introduce a larger routing unit. We then route first to the region, then to the IP prefix within the region.
3. IP prefix aggregation (combine and split prefixes)

The idea is to create subnets (*splitting*) and joining (*aggregation*) IP's based on their prefix, such that we may address a "pool" of IP's and once in there, route to the more specific IP's in the subnet. More specific:

- `Subnets`: Internally split one less specific prefix into multiple more specific prefixes.
- `Aggregation`: Externally join multiple more specific prefixes into one large prefix.

5.5.3 Distance Vector Routing

The `distance vector routing` algorithm follows a distributed BEllman-Ford approach. It works well and was used in RIP, but converges slowly after some types of failures:

- Setting: Nodes know only the cost to neighbors, not the topology. They can communicate with their neighbors via messages.
- Process: Each node maintains a `vector of distances` and next-hops to all destinations. The algorithm proceeds as follows:
    1. Initialize each vector with cost to oneself = 0 and cost to others = ∞.
    2. Periodically send this vector to the neighbors.
    3. Every round, after receiving the vectors of all neighbors:
        1. For each neighbor, add the cost of the link to the neighbor to the vector received from that neighbor.
        2. Set all vector entries (except the oneself) to the minimum of all received values and set the corresponding neighbor as the next-hop.

5.5.4 Flooding

`Flooding` is used to broadcast a message to all nodes in a network in a very simple but highly inefficient way:

1. Send an incoming message to all neighbors, but
2. Remember the message (using sequence numbers) such that a message is flooded only once to the neighbors.

5.5.5 Link State Routing

The `Link State Routing Algorithm` works as follows:

1. The nodes flood the topology in the form of link state packets such that each node learns the full topology.
2. Each node computes its own forwarding table by running Dijkstra (or equivalent)

5.5.6 Distance Vector Routing vs. Link State Routing

Goal	Distance Vector	Link State
Correctness	Distributed Bellman-Ford	Replicated Dijkstra
Efficient Paths	Approximate with shortest paths	Approximate with shortest paths
Fair paths	Approximate with shortest paths	Approximate with shortest paths
Fast convergence	Slow (many exchanges)	Fast (flood and compute)
Scalability	Excellent	Moderate

Computer Networks - Lecture 17 & 18

- Author: Ruben Schenk
- Date: 10.06.2021
- Contact: ruben.schenk@inf.ethz.ch

5.6 Border Gateway Protocol (BGP)

So far, we have looked at `intra-domain routing protocols` like distance vector and link state algorithms. These work fine within, e.g., an autonomous system, but as soon as a network gets too big, they quickly become infeasible:

- Distance vector protocols converge slowly, so for a network as big as the internet, convergence would never happen.
- Link state algorithms need the whole network topology, which is impossible for the internet.

Since the internet is a network of networks, referred to as an `autonomous system (AS)`, we use special `inter-domain routing protocols` like `BGP` to connect the autonomous systems. By using BGP, autonomous systems exchange information about IP prefixes that they can reach. The protocol needs to solve three key challenges:

- *Scalability*: The number of networks and prefixes is huge.
- *Privacy*: Networks don't want to expose internal topologies.
- *Policy Enforcement*: The network needs to control where to send and receive traffic in the absence of an internet-wide link-cost metric.

BGP relies on `path-vector routing`, similar to distance-vector routing, but with the key idea, that we advertise an *entire AS-level path* instead of distances.

5.6.1 BGP Policies

Two ASes connect only if they have a business relationship. We distinguish between two types of relationships:

- Customer-Provider Relationship:
  - In a customer-provider relationship, a customer pays a provider to get internet connectivity globally, e.g. Swisscom is a customer of Deutsche Telekom. The amount the customer pays is based on the peak usage.
- Peer-Peer Relationship
  - In a peer-peer relationship, peers don't pay each other for connectivity bur rather connect out of common interests, mainly since they exchange a large amount if traffic, e.g. Deutsche Telekom and AT&T.

Policy Rules

BGP obeys the following policy rules:

- Providers transit traffic for their customers.
- Peers to not transit traffic between each other.
- Customers do not transit traffic between their providers.

Selection and Export

In `selection`, we must decide which path to use for *outbound* traffic. The general rule is to prefer routes coming from customers over peers over providers.

In `export` we must decide which path to use for *inbound* traffic. Routes coming from customers are propagated to everyone else, that is, to peers and providers, where as routes coming from peers and providers are only propagated to customers.
Note that this requires Tier-1's to be connected through a *full-mesh of peer links*, otherwise the Internet would be partitioned.

5.6.2 BGP Protocol

There are two different types of BGP sessions:

- `external BGP (eBGP)`: Those sessions connect border routers in different ASes and are used to learn routes to external destinations.
- `internal BGP (iBGP)`: Those sessions connect the routers in the same AS and are used to disseminate externally-learned routes internally.

BGP as a protocol is rather simple and consists of four basic types of messages:

- `OPEN`: Establish TCP-based BGP session.
- `NOTIFICATION`: Report unusual conditions.
- `UPDATE`: Inform neighbor of $a)$ a new best route, $b)$ a change in the best route, or $c)$ the removal of the best route.
- `KEEPALIVE`: Inform a neighbor that the connection is alive.

BGP Updates

A `BGP update` message carries an IP prefix together with a set of attributes that describe route properties that can be used in route selection/export decisions. Such attributes can either be local (only seen in iBGP sessions) or global (seen on both iBGP and eBGP sessions). Possible attributes are:

- `NEXT-HOP`: A global attribute which indicates where to send traffic next. It identifies the egress point and is set when a route enters an AS and does *not* change within the AS.
- `AS-PATH`: A global attribute that lists all ASes a route has traversed (in reverse order).
- `LOCAL-PREF`: A local attribute set at the border, it represent how "preferred" a route is. A higher value results in all routers using this route to reach any external prefixes, even if they are closer to another egress point.
- `MED (Multi-Exit Discriminator)`: A global attribute which encodes the relative "proximity" of a prefix with respect to the announcer. In contrast to `LOCAL-PREF`, a lower `MED` indicates closeness and is preferred over a higher value.

BGP Decisions

Given the set of all acceptable routes for each prefix, the `BGP decision process` elects a single route. BGP thus is a single path protocol.

> `Route picking` in BGP works as follows: Out of all possible routes, BGP decision processing picks exactly one with the following precedence:
>
> 1. Highest *LOCAL-PREF*.
> 2. Shortest *AS-PATH* length.
> 3. Lower *MED*.
> 4. On a tie of the first three attributes, we pick the routes that were learned via eBGP over routes that were learned internally via iBGP:
>   - A lower IGP metric to the next hop
>   - A smaller egress IP address (tie-breaker)

5.6.3 Problems with BGP

There are several problems with BGP:

- Reachability: BGP does not guarantee reachability even if a graph is connected.
- Security: Simply absent. AS can advertise any prefix, AS can arbitrarily modify content og an AS-PATH, and can forward traffic along different paths than the advertised one.
- Convergence: With arbitrary policies, the protocol might fail to converge tue to policy oscillations.
- Performance: Path selection happens for economic reasons, not based on performance.
- Anomalies: BGP is bloated and underspecified at the same time such that there are conflicting interpretations.
- Relevance: BGP policies are rapidly changing.

> If all ASes policies follow the customer/peer/provider rules, also called Gao-Rexford rules, then BGP is guaranteed to `converge`.

6. Link Layer

The `link layer` is concerned with transferring messages over one or more connected links, thus providing a service to the network layer by building on top of the physical layer.
We'll refer to any device that runs the link layer as a `node` and to the communication channels that connect adjacent nodes as `links`. Over a given link, a transmitting node encapsulates a *datagram* in a `link-layer frame`.

6.1 Framing

The physical layer gives us a stream of bits. But how do we interprete it as a sequence of frames?

The job of the link layer is to interprete that bitstream as a sequence of frames.

6.1.1 Byte Count

The first idea is to use a `byte count`. In this approach we start each frame with a length field that denotes the size of the frame in bytes.

The problem with this approach is that once a framing error is made, there is no way to recover from it, and all the subsequent frames are decoded incorrectly.

6.1.2 Byte Stuffing

A better idea is `byte stuffing`. The approach is to have a special `FLAG` byte that denotes the start and end of a frame and a special `ESC` byte.

- If `FLAG` appears in the data, replace it with `ESC FLAG`.
- If `ESC` appears in the data, replace it with `ESC ESC`.

This way, any unescaped (ESC) `FLAG` is a start/end of a frame.

6.1.3 Bit Stuffing

We can also stuff at the bit level:

- We call a FLAG six consecutive 1s
- On transmit, after five 1s in the data, insert a 0
- On receive, a a after five 1s is deleted

Computer Networks - Lecture 19 & 20

- Author: Ruben Schenk
- Date: 14.06.2021
- Contact: ruben.schenk@inf.ethz.ch

6.2 Error Detection and Correction

Bit-errors are introduced by signal attenuation and electromagnetic noise and are, in general, not avoidable when using certain types of media for transport. Our aim is to detect and possibly even correct these errors at a very low level, namely at the link layer. A possibility is to add `redundancy`, i.e., check bits that allow some errors to be detected. Adding even more check bits even makes it possible to directly correct some errors.

Goal: Structure code to detect many errors with few check bits and modest computational effort.

6.2.1 Intuition & Usage

A `codeword` consists of D data bits and R check bits. The sender computes R based on the data D and sends the codeword of $D + R$ bits (concatenated). A receiver then, upon receiving $D + R$ bits, recomputes R' based on D. If R' doesn't match R, then there is an error somewhere.

Note: For data bits D, the set of correct codewords should only be a tiny fraction of the set of all codewords, such that the probability of a randomly chosen codeword being correct is very small.

Hamming Distance

Let the `distance` of two codewords $D + R_1$ and $D + R_2$ be the number of bits that need to be flipped to turn $D + R_1$ into $D + R_2$ (or vice versa).

> The `hamming distance` of a code is the minimum distance between a pair of codewords.

> A code of hamming distance $d + 1$ can `detect` up to d errors.

> A code of hamming distance $2d + 1$ can `correct` up to d errors by mapping to the closest codeword.

6.2.2 Error Detection

Parity Bit

Take D data bits and add only a single `check bit` c such that c is the sum of all D bits modulo 2, i.e.

$$c \equiv_2 \sum_{i=1}^{D} x_i$$

where x_i denotes the i-th data bit.

The distance of the code is 2, it can thus detect exactly one error (but not locate it) and correct none.

Checksums

Idea: Sum up data in N-bit words, that gives a stronger protection than parity and is widely used in TCP/IP/UDP etc. `Internet checksum` works as follows:

Sending side:

1. Arrange data in 16-bit words.
2. Put 0 in the checksum position and add the words.
3. Add any carryover back to get 16 bits.
4. The checksum is now given by the complement (negation).

Receiving side:

1. Arrange the data in 16-bit words.
2. Add the words and the checksum.
3. Add any carryover back to get 16 bits.
4. Negate the result and check if it is 0, if not, then an error occurred.

Cyclic Redundancy Check (CRC)

Let the data D consist of d data bits. Sender and receiver then agree on a $k+1$ bit pattern, which is called the generator G (and can be expressed as a polynomial over the finite field $GF(2)$). The sender will then choose r bits R to append to D such that the resulting $d+r$ bit pattern is congruent modulo 2 to G, i.e., $D + R \equiv_2 G$.
The receiver then checks whether the received $d+r$ bits are divisible by G with a 0 remainder. If not, an error occurred.

6.2.3 Error Correction

The general problem that makes error correction so hard is that check bits aren't reliable either. If we can construct a code of Hamming Distance $\geq 2d+1$, then codewords containing at most d bit errors are uniquely mapped to the closest valid codeword.

Hamming Code

A `hamming code` is a code with hamming distance 3. It can hence correct 1 bit error. For a message of n bits, choose k such that $n = 2^k - k - 1$ holds. We insert check bits at positions of powers of 2, starting with position 1.
When written in binary, the positions of the check bits have exactly one bit set to 1. The check bit $p8$ for example records the parity of all positions that have the 4-th bit set to 1. Check bit $p4$ records parity of all positions that have the 3rd bit set to 1.

Now to decode, we recompute the check bits, arrange them as a binary number, called the `syndrome`. It tells us the error positions where the bit needs to be flipped. If the syndrome is 0, no error occurred.

6.2.4 Detection vs. Correction

Instead of correcting bit-errors, one could simply try to detect them and, upon detecting an error, requesting a retransmit. This might be more efficient than correction, but it is largely dependent on the setting:

- *Error correction* is needed when we expect error at a small rate or when there is no time for retransmission.
- *Error detection* is more efficient when errors are not expected and when errors are large when they occur.

6.3 Retransmissions

Instead of correcting errors, one can simply detect them and retransmit the frames in which they occurred.

Automatic Repeat Request (ARQ)

ARQ is often used when errors are common or if they must be corrected (e.g. TCP). IT works in the following way:

- Receiver: Automatically ACKs correct frames
- Sender: Automatically retransmits after timeout until ACK is received.

6.4 Multiple Access

`Multiplexing` is the network word for the sharing of a resource. A classic scenario is the sharing of a link among different users with one of the following approaches:

- `Time Division Multiplexing (TDM)`: Users take turns on a fixed schedule which lets them send at a high rate but only during a fraction of time.
- `Frequency Division Multiplexing (FDM)`: Puts users on different frequency bands which lets them send at a low rate but at all the time.

6.4.1 Multiplexing Network Traffic

Network traffic is bursty, the load varies greatly over time, and it is thus very inefficient to allocate the peak need with TDM/FDM. We therefore need `multiple access schemes` that multiplex users according to their demand.

Randomized Multiple Access

Assume a distributed network with no master node, i.e. no-one is in charge. We will now look at a randomized multiple access protocol which is also referred to as the `medium access control (MAC) protocol`, which provided the basis for the classic *Ethernet*.

ALOHA Protocol

The `ALOHA Protocol` connected the Hawaiian islands in the 1960s. The protocol is really simple:

- Nodes just send whenever it has traffic.
- If there was a collision, i.e. no ACK was received, then wait a random time and resend.

Under low load, this works fairly well. However, under high load the efficiency is very bad.

Carrier Sense Multiple Access (CSMA)

CSMA denotes a improvement of ALOHA for LAN. It works by listening for activity before sending. This way we have less collisions, but they can still occur due to sending delays.

- `CSMA/CD` is a CSMA protocol with added *collision detection*. This will reduce the cost of collisions by detecting them and aborting the rest of the frame time.

Let D be the distance between the two furthest away nodes in the network. Then the time window in which a node may hear of a collision is $2D$. We therefore impose a minimum frame size that lasts for $2D$ seconds such that the nodes can0t finish before the collision is detected. This leads to a minimum frame size of 64 bytes for Ethernet.

Another problem arises when a node detects that another node is sending. When it simply waits and sends when the other node is done, this might lead to multiple waiting loads queuing up and even amplifying the problem. One simple solution is, that for N queued senders, each sends with probability $\frac{1}{N}$. The waiting time interval is doubled for each successive collision which leads to a binary exponential backoff.

Wireless Multiple Access

Wireless is much more complicated than the wired case:

- Nodes may have a *different area of coverage*. This may lead to the fact that for some nodes there is interference and for others there isn't. We distinguish between:
  - `Hidden terminals`: Two nodes that cannot reach each other, yet they collide ad some intermediate node.
  - `Exposed terminals`: Two nodes that can reach and thus "hear" each other, yet they don't collide.
- Nodes can't hear while sending and therefore, collision detection is a waste of time.

`Multiple Access with Collision Avoidance (MACA)` is a possible solution. The protocol works with the following rules:

1. A sender node transmits a RTS (Request-To-Send, with frame length).
2. The receiver replies with a CTS (Clear-To-Send, with frame length).
3. Sender transmits the frame while nodes hearing the CTS stay silent.

Note: MACA solves both the hidden and exposed terminal problem.
Note: Collisions on the RTS/CTS are still possible, but less likely.

Token Ring

Another protocol where nodes are arranged in a ring. A token rotates "permission to send" to each node in turn.

- This leads to a fixed overhead, no collisions, predictable, and a regular chance for each node to send.
- However, what if a token is lost, what if a token manager crashes? Overhead is way too high for lower loads.

Note: In practice, it is hard to beat random multiple access protocols.

Computer Networks - Lecture 21 & 22

- Author: Ruben Schenk
- Date: 15.06.2021
- Contact: ruben.schenk@inf.ethz.ch

6.5 LAN Switching

Modern Ethernet is based on the usage of switches rather than multiple access, i.e., hosts are connected with to switch ports. We distinguish three types of devices:

- `Hub/Repeater` : Only at the physical layer, strengthens the signal.
- `Switch` : At the link layer, looks where to forward the packet.
- `Router` : At the network layer.

6.5.1 Inside a Switch

A `switch` :

- uses frame addresses to connect input port to the right output port. It allows fro multiple frames to be switched in parallel.
- ports are full-duplex, allowing for both input and output. There is no multiple access control.
- in case there is contention, there are both input and output buffers. On overload, frames are lost.

Switches and hubs have replaced shared cable of classic Ethernet, since they provide a better reliability and scalable performance.

6.5.2 Switch Forwarding

A switch needs to find the right output port for the destination address in the Ethernet frame.

Backward Learning

Switches use a port/table to forward frames and they generate the table as follows:

1. To fill the table, it looks at the source address of the input frames.
2. To forward, it sends it to the port, or else broadcasts it to all ports.

This method works as long as there are no loops in the system.

Switch Spanning Tree

One solution to loops in the network are spanning trees.
Switches collectively find a `spanning tree` for the topology and then only forward to along the tree. When a frame is *broadcasted*, it goes all the way up to the root and then all the way down to all the branches.

Spanning Tree Algorithm

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

7. Physical Layer

The `physical layer` concerns how signals are used to transfer message bits over a link. We want to send digital signals but the medium we use uses analog signals.

7.1 Properties of Media

7.1.1 Link Model

WE abstract the physical model as follows: We consider the `rate` (or bandwidth) in bits/second and the `delay` or `latency` in seconds as the key properties of such a physical channel.

Message Latency

The `message latency` L consists of two parts:

- `Transmission delay` T: The time used to put a M bit message on the wire:

$$T = \frac{M\,[\text{bits}]}{Rate\,[\text{bits/sec}]} = \frac{M}{R}\,[\text{sec}]$$

- `Propagation delay` P: The time used for the bits to propagate across the wire:

$$P = \frac{length}{speed\,of\,signals} = \frac{length}{\frac{2}{3}c} = D\,[\text{sec}]$$

This yields a total latency or delay of $L = \frac{M}{R} + D$. For rates we use powers of 10, for storage/data sizes we use powers of 2.

Bandwidth-Delay Product

The amount of data in flight is the `bandwidth.delay product`, given by $BD = R \cdot D$, and is measured in either bits or messages.

7.1.2 Types of Media

We introduce the following different types of media:

- *Wires - Twisted Pair*: Used in LANs and telephone lines. Twists reduce radiated signals and effects of external interference.
- *Wires - Coaxial Cable*: A copper core shielded by insulating material, braided outer conductor and protective plastic covering. Provides a better performance due to better shielding.
- *Fiber*: Long, thin, pure strands of glass allow for enormous bandwidths over long distances. Works by having a light source at one end and a photo-detector on the other end. Transmits at around $\frac{2}{3}c$.
- *Wireless*: Sender radiates signal over and entire region into all directions. This potentially leads to interference.

7.2 Simple Signal Propagation

Analog signals are used to encode digital bits. A signal over time can be represented by its frequency components (called *Fourier analysis*). As signals `propagate over a wire`, the following happens:

1. The signal is delayed (since it propagates at $\frac{2c}{3}$)
2. The signal is attenuated
3. Frequencies above a cutoff are highly attenuated
4. Noise is added to the signal

When a signal, however, is `propagated over fiber`, the light propagates with very low loss in three very wide frequency bands (furthermore, at these frequencies the attenuation is very low).

Signals that are sent over `wireless` travel at the speed of light, spread out and attenuate at a fast rate. Multiple signals on the same frequency

7.3 Modulation Schemes

We need signals to represent bits. One way to do this is the `Non-Return to Zero (NRZ)` scheme. In this scheme, high voltage ($+V$) represents a 1, low voltage ($-V$) represents a 0.

This simple scheme uses only 2 levels, if we were to increase it to 4 levels, it could support 2 bits per symbol.

7.3.1 Clock Recovery

The receiver needs frequent signal transitions to decode bits. It needs to know how many 0's in a row were transmitted. This can be hard for very long sequences of 0's.

For that reason, the `4B/5B Clock Recovery` maps every 4 data bits into 5 code bits while eliminating long runs of 0's. This is done with an encoding table, and each string that is mapped to has at most 3 zeros in a row.
Another approach is to invert signal on a 1 to break long runs of 1's, which is called `NRZI`.

7.3.2 Passband Modulation

We have so far only seen `baseband modulation` for wires, which can be applied when a signal is sent directly on a wire.
However, these signals do not propagate well on fiber or on wireless, so we need to send at higher frequencies. This is exactly what `passband modulation` does: It carries a signal by modulating a carrier (a signal oscillated at a desired frequency). This modulation happens by changing *amplitude, frequency*, and *phase*.

7.4 Fundamental Limits

Key properties of a channel include bandwidth B, signal strength S, and noise strength N. It holds that:

- The rate of transitions is limited by B
- The number of signal levels that can be distinguished is limited by S and N.

7.4.1 Nyquist Limit

The maximum symbol rate is $2B$ (on alternating 1s and 0s). Thus, if there are V signal levels, ignoring noise, the maximum bit rate is:

$$R = 2B \log_2 B \, [\text{bits} \, / \, \text{sec}]$$

7.4.2 Shannon Capacity

How many levels that can be distinguished depends on the `signal-to-noise ration` (or S/N). `SNR` is given on a log-scale in deciBEls according to:

$$\text{SNR}_{\text{dB}} = 10 \log_{10} (S/N)$$

The `Shannon limit` for a capacity C is the maximum information carrying rate of the channel and is given by:

$$C = B \log_2 \left(1 + \frac{S}{N}\right) [\text{bits} \, / \, \text{sec}]$$

7.4.3 Digital Subscriber Line (DSL)

`DSL` reuses a twisted pair telephone line to the home. Since only the lowest $4 \, \text{kHz}$ of approximately $2 \, \text{MHz}$ of bandwidth are used by the telephone service, the rest can be used for different purposes.

DSL uses passband modulation which creates separate bands for up- and downstream with different bandwidth sizes. The modulation varies both amplitude and phase. On high SNR, there are up to 15 bits per symbol, where as on low SNR, there is only 1 bit per symbol, so the connection is slower.

8. Routing Security

8.1 Basic Security Properties

8.1.1 Terminology

We define the following meanings for the 4 key terms:

- `Secrecy` : Keep data hidden from unintended receivers.
- `Confidentiality` : Keep someone else's data secret.
- `Privacy` : Keep data about a person secret.
- `Anonymity` : Keep the identity of a protocol participant secret.

Furthermore, we want to distinguish the following terms:

- `Data Integrity` : Ensure that data is *correct* and prevent unauthorized or improper changes.
- `Entity Authentication/Identification` : Verifies the identity of another protocol participant.
- `Data Authentication` : Ensures that data originates from a claimed sender.

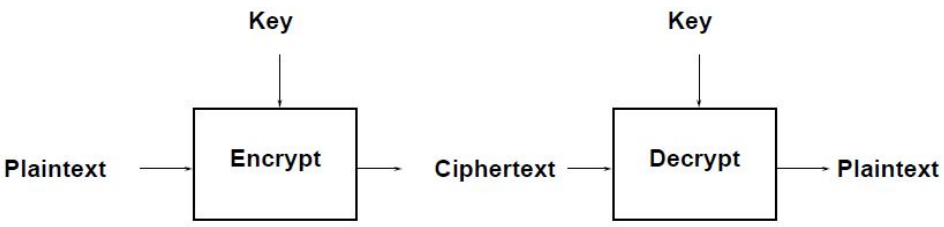
8.2 Basic Cryptographic Mechanisms

8.2.1 Symmetric Encryption Primitives

In this protocol, the following holds:

- Encryption key E_K = decryption key D_K
- *Encryption*: E_K(plaintext) = ciphertext
- *Decryption*: D_K(ciphertext) = plaintext

We write $\{\text{plaintext}\}_K$ for E_K(plaintext).

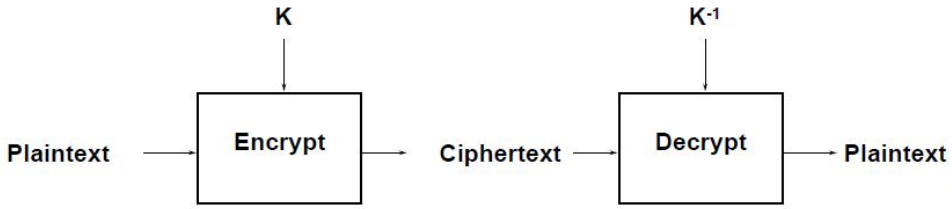


8.2.2 Asymmetric Encryption Primitives

In this protocol, the following holds:

- Encryption key K is publicly know: `public key`
- Decryption key K^{-1} is secret: `private key`
- *Encryption*: E_K(plaintext) = ciphertext
- *Decryption*: $D_{K^{-1}}$(ciphertext) = plaintext

We write $\{\text{plaintext}\}_K$ for $E_K(\text{plaintext})$.



8.2.3 Symmetric vs Asymmetric Encryption

We make the following observations:

- *Symmetric Encryption*
  - Need shared secret key
  - 100'000'000 ops/s
- *Asymmetric Encryption*
  - Need authentic public key
  - 1000 signatures/s or 10'000 verify/s

8.3 Security For Routing Protocols

8.3.1 Intra-Domain Routing

To perform an attack on link-state protocols, one only needs to compromise *one* router or one routing adjacency since link-state protocols rely on flooding.

In both cases, the attacker obtains a complete network view and the ability to inject messages network-wide.

The solution is quite simple: One simply needs to rely on cryptography. We only need to send authenticated announcements and cryptographically protect topology information.

8.3.2 Inter-Domain Routing

We look at the lack of security of `BGP`, the problems that follow from this and their solutions.

BGP does not validate the origin of advertisements

Regional Internet Registries assign IP address blocks. However, the origination of a prefix into BGP must be proper, i.e., by the AS who owns the prefix.

This is, however, not checked by BGP. So what's to stop someone else, i.e. another AS, from originating the prefix? This process is known as `Prefix Hijacking`.

Computer Networks - Lecture 23 & 24

- Author: Ruben Schenk
- Date: 17.06.2021

- Contact: ruben.schenk@inf.ethz.ch

BGP Does not validate the content of advertisements

We might get `bogus AS paths` by removing a part of an AS path, for example turning '701 3715 88' into '701 88'. This way we can for example avoid AS 3715 or help AS 88 look like it is closer to the Internet's core.

We might also add ASes to the path, e.g. turning '701 88' into '701 3715 88'. This way we can trigger loop detection in AS 3715 or making our AS look like it has richer connectivity.

Proposed enhancements

We could develop a `secure BGP`, with *origin authentication* and *cryptographic signatures*. `BGPsec` adds the following:

- *Address attestations*: Claims the right to originate a prefix, is signed and distributed out-of-band, and is checked through delegation chain from ICANN.
- *Route attestations*: Distributed as an attribute in BGP update messages, and signed by each AS as route traverses the network.
- *Resource Public-Key Infrastructure (RPKI)*: Per-prefix certificate issued by Regional Internet Registries (RIR), and used to authenticate the first AS hop through Route Origin Authorization (ROA).

9. SCION - A Secure Multipath Interdomain Routing Architecture

9.1 SCION Principles

`SCION` is based on the following principles:

- Stateless packet forwarding
- Instant convergence routing
- Path-aware networking
- Multi-path communication
- High security through design and formal verification
- Sovereignty and transparency for trust roots

9.2 SCION Overview

9.2.1 Control Plane: How to find end-to-end paths?

One first approach to scalability is `Isolation Domains (ISDs)`:

- Isolation Domains (ISDs) are grouped ASes
- There is an `ISD core`, which are the ASes that manage the ISD
- A `core AS` is an AS that is part of the ISD core
- Each ISD defines their `TRC (Trust Root Configuration)`, which contains the root cryptographic keys to verify ISD operations

Path Exploration

`Beaconing` describes one way of intra-ISD path exploration:

- Core ASes initiate `Path-segment Construction Beacons (PCBs)`, also called *beacons*
- PCBs traverse the ISD as a flood to reach downstream ASes
- Each AS receives multiple PCBs representing path segments to a core AS

A `PCB` contains an info field with the PCB creation time. Each AS on the path adds to this:

- Its AS name
- A hop field for data-plane forwarding including:
  - Link identifiers
  - Expiration time
  - Message Authentication Code
- An AS signature

PCBs contain `path segments` that can be used as communication paths to communicate with the core AS that initiated it. We differ between:

- `Up-path segments`: PCB is used from AS to core AS
- `Down-path segments`: PCB is used from core AS to AS

Path Registration

`Up-path segment registration` works the following way:

- AS selects path segments to announce as *up-path segments* for local hosts
- Up-path segments are registered at local path servers

`Down-path segment registration` works as follows:

- AS selects path segments to announce as down-path segments for others to use to communicate with AS
- Down-path segments are uploaded to core path server in the core AS

9.2.2 Data Plane: How to send packets?

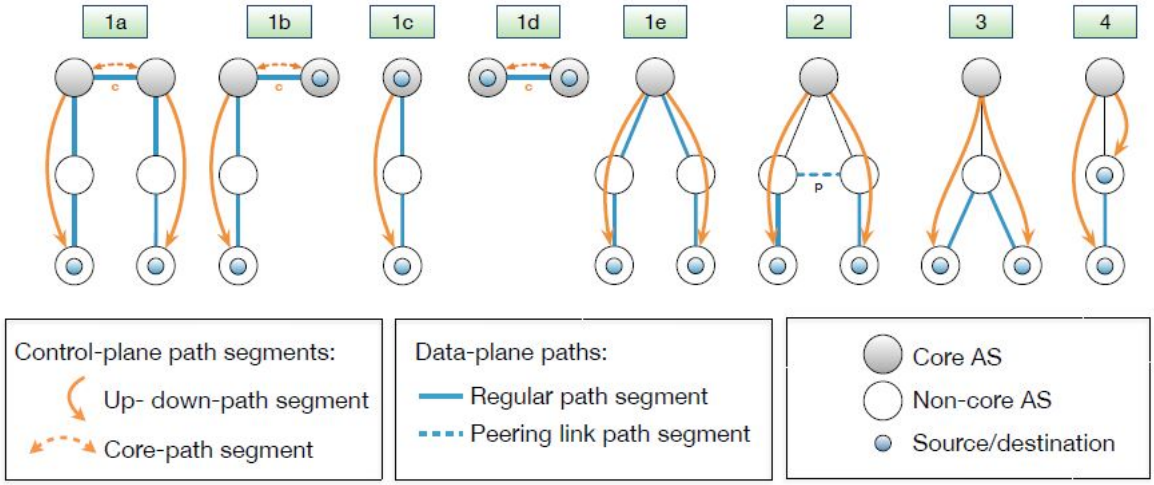
Path lookup

The following steps are performed by a host to `obtain path segments (path lookup)`:

1. The host H contacts RAINS server with a name it wants to look up:
    1. $H \rightarrow RAINS$: www.scion-architecture.net
    2. $RAINS \rightarrow H$: ISD X, AS Y, local address Z
2. Host contacts local path server to query path segments
    1. $H \rightarrow PS$: ISD X, AS Y
    2. $PS \rightarrow H$: up-path, core-path, down-path segments
3. Host combines path segments to obtain an end-to-end path

Path combination

The following path combinations are possible in SCION:



SCION Packet Header

The `SCION common header` encodes the following attributes:

- Version
- Destination and Source address types
- Total packet and header length
- Pointer to current info and hop fields
- Next header type field