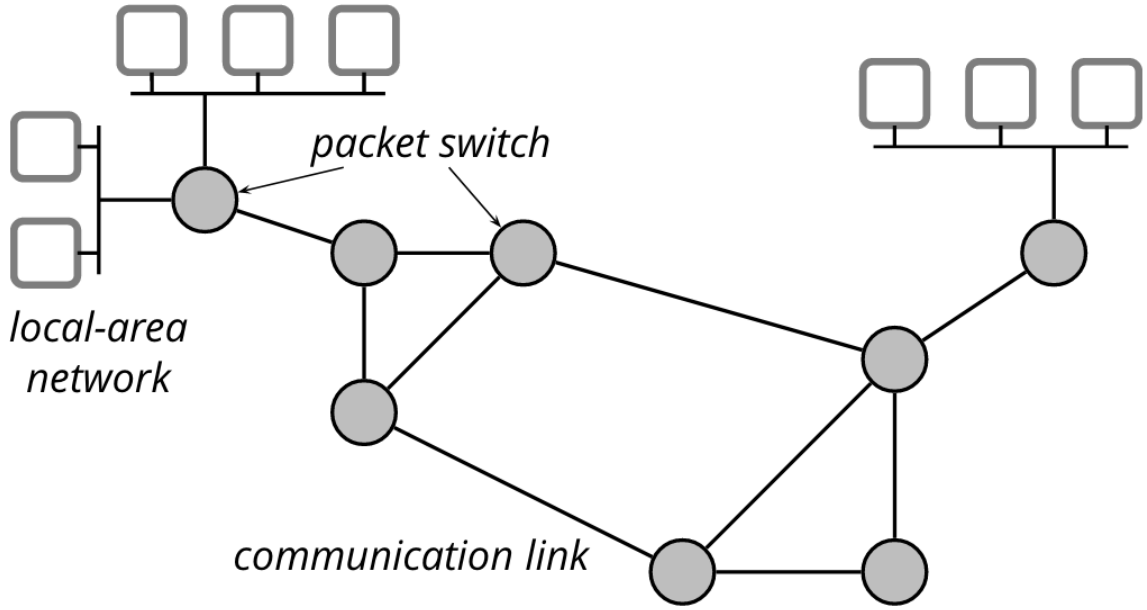
Sunday, 11 November 2018

Fundaments for Computer Networking

Packet Switching



Definitions:

- **Packet switch**: a link-layer switch or a **router**;
- **Communication link**: a connection between packet switches and/or end-systems;
- **Route**: a **sequences of switches** that a packet flows through from A to B;
- **Protocol**: it's a procedure of sending and receiving information to and from end-systems and packet switches.

The internet is a **packet-switched network**, where the information is transmitted in **packets between switches**, which operate on individual packets.

A switch (**router**) **receives packets and forwards** them along to other switches or to end-systems.

Every forwarding decision is taken depending on the information contained in the packet.

Packet switching isn't subject to any setup cost, but there is a significant amount of time and space overhead for each packet.

Processing cost due to the forwarding of the packet (must decide to which router forward to).

Space overhead because every packet must be self-contained.

Packet switching doesn't allow for easy quality of service (reservation of bandwidth), but it achieves the best utilisation of network resources.

The internet is:

- a **Connectionless** 'best-effort' communication service: the network accepts 'datagrams' for delivery and it's best-effort as in 'unreliable', although not by intention.
- **Connection-oriented**: virtual duplex communication between end-systems, similar to the telephone service, where information is transmitted reliably and in-order.

What is a protocol?

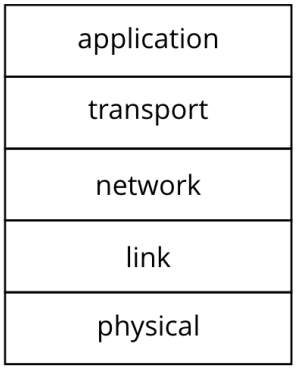
From **Wikipedia**:

> In telecommunication, a **communication protocol** is a system of rules that allow two or more entities of a communications system to transmit information via any kind of variation of a physical quantity. The protocol defines the rules, syntax, semantics and synchronization of communication and possible error recovery methods. Protocols may be implemented by hardware, software, or a combination of both.

In short: a system of rules designed to allow two end-systems to communicate with each other in a reliable manner.

A protocol is similar to a program: it is in fact a **distributed program**, where processes can send messages to each other, featuring:

- An executable specification
- Unambiguity
- Complete (all cases and scenarios are considered and handled)
- Definition of all the necessary message formats



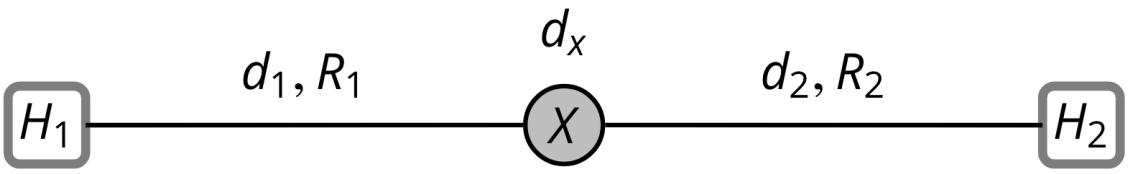
Delay, Throughput, Loss

How to measure the speed and capacity of a network connection?

Delay (or **Latency**): the time that it takes for one bit to go from endpoint A to endpoint B.

Transmission Rate (or **Throughput**): the amount of information that is sent or received within a time unit (usually 1 second).

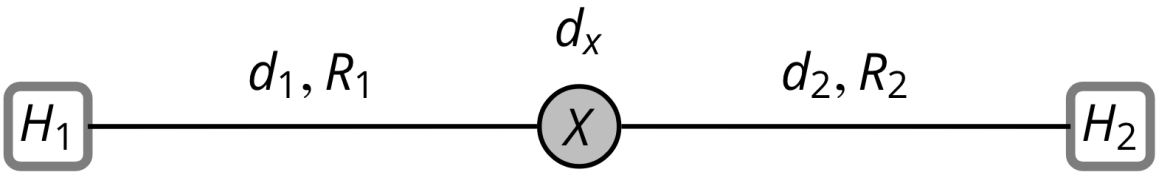
Stream



The time required to transfer a file from H_1 to H_2 is:

$$d_{end-end} = d_1 + d_x + d_2 + \frac{l}{\min(R_1, R_2)} \text{ sec}$$

Store-And-Forward



The time required to transfer a file from H_1 to H_2 is:

$$d_{end-end} = d_1 + \frac{l}{R_1} + d_x + \frac{l}{R_2} + d_2 \text{ sec}$$

Queuing Delay

What happens if the input rate to a router is greater than its output rate?

This means the router cannot process packets fast enough, hence the incoming packets are placed into a queue:

$$d_x = d_{cpu} + d_{queue}$$

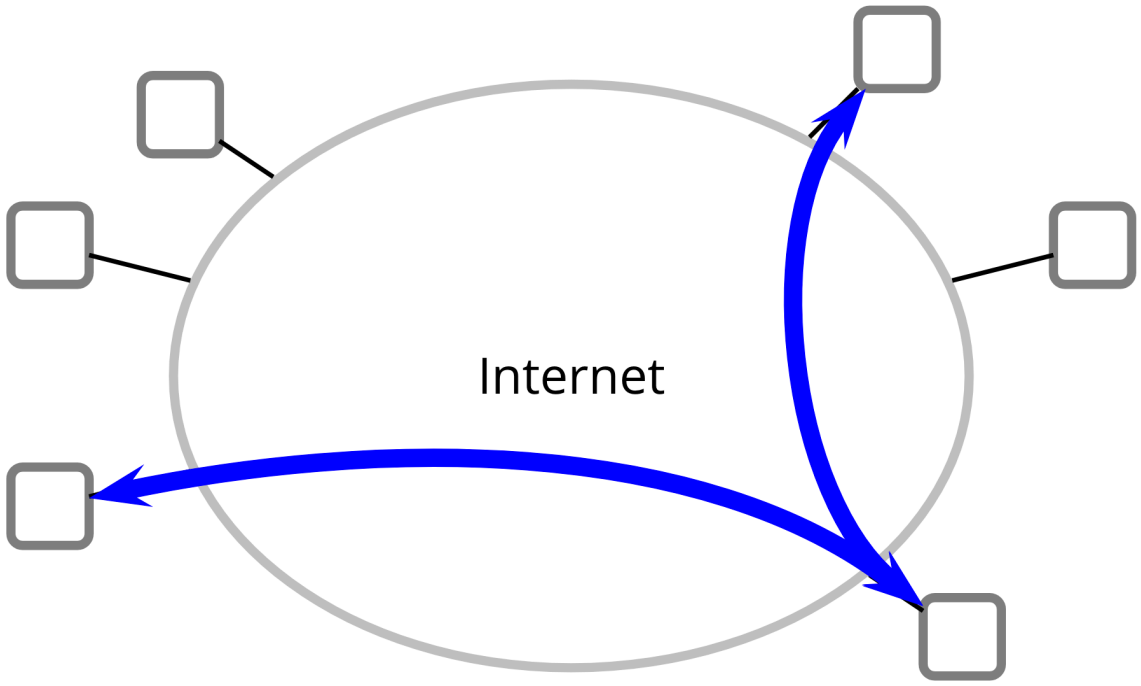
with: $d_{queue} = \frac{|q|}{R_x}$ and R_x being the output rate

As the input rate approaches the max throughput rate, packets will experience longer and longer delays.

Since the **routers have a limited queue length** (buffers), some **packets will be dropped**.

Network Applications and the Web

Internet applications are end-systems applications



A end-system application is commonly referred to as a **process**, which is the execution of a program.

Processes may exchange messages, received messages can be considered to be the input of a process.

Different processes may be running on different end-systems, with different configurations (Operative System, Components, …)

A process must be able to address another specific process.

For each communicating process we distinguish two roles:

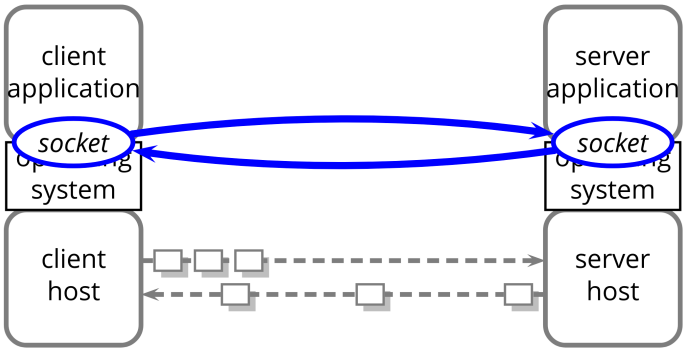
- **Client**: the process which **initiates the communication**. If the communication is carried over a connection-oriented service, the client is the one process **establishing the connection**.
- **Server**: the process that **awaits for connections**. It's the process that receives and **accepts the connection request**.

A **peer-to-peer architecture** allows a process to be **client and server at the same time**.

A end-system might be running multiple processes and is addressed by its IP over the network, a **port number** is required **to identify a process** within the end-system.

The operating system manages the network interfaces.

The applications use network through sockets.



The World Wide Web

It was developed in the early 90s and it's based on the idea of **hypertext** and **links**.

Terminology:

- **Document**: a webpage
- **Objects**: a file. A document may contain several objects (images, applets, …)
- **URL** (**U**niform **R**esource **L**ocator): specifies the address of an object.
- **Browser**: called also *user agent* is the program that a user runs to get and display web pages.
- **Web Server**: an application housing objects and serves them using **HTTP** protocol.

HTTP HyperText Transfer Protocol

The main purpose of HTTP is to provide access to web objects.

It uses a connection oriented transport layer (i.e.: TCP)

It is a series of **requests issued by the client**, and **responses by the server**, each one in reply to a single request

HTTP is stateless: each request is independent in its behaviour from other requests.

HTTP Request

PROTOCOL VERSION | URL SPECIFICATION | CONNECTION ATTRIBUTES | CONTENT/FEATURE NEGOTIATION

HTTP Reply

PROTOCOL VERSION | REPLY STATUS/VALUE | CONNECTION ATTRIBUTES | OBJECT ATTRIBUTES | CONTENT SPECIFICATION (TYPE/LENGTH) | CONTENT

A protocol should always include a version number.

A mechanism to negotiate the protocol version allows protocol design to change.

URL

http://www.inf.usi.ch/carzaniga/index.html

The host name determines where the request goes, as it maps to a network address (IPv4).

```
GET /carzaniga/index.html HTTP/1.1
Host: www.inf.usi.ch
Connection: close
User-agent: Mozilla/4.0
Accept-Language: it
```

The host name is passed as a parameter within the request, so that the server know the full URL.

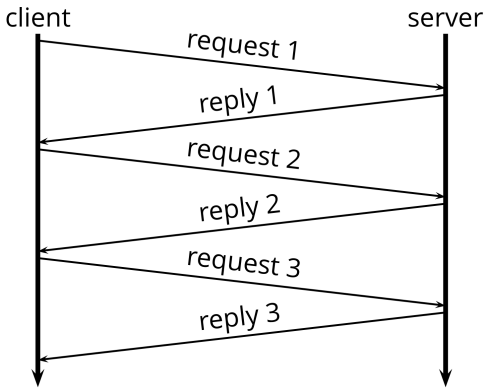
This is done to allow the server to server virtual sites and subdomains.

HTTP and Persistent Connections

HTTP/1.1 uses persistent connections: the same TCP connection can be used to issue multiple requests and transfer objects.

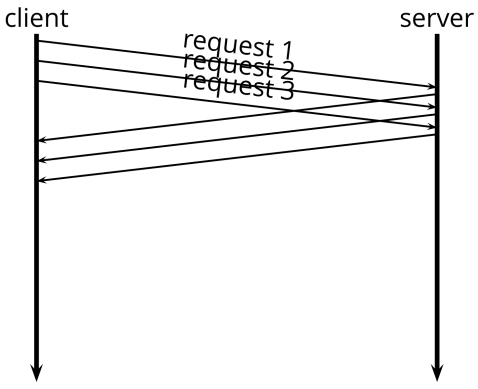
The default behaviour is to use persistent connections.

"Connection: close" in the request and response headers **indicates the intention of both sides not to use a persistent connection**.



A persistent connection can be used to transfer multiple objects without establishing a TCP connection for each of them.

A more efficient use of a connection is by pipelining the requests:



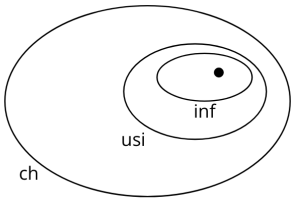
DNS Domain Name System

A network-level address is either a IPv4 or IPv6, 32 bits and 128bits respectively, which are fine for computers but hardly readable.

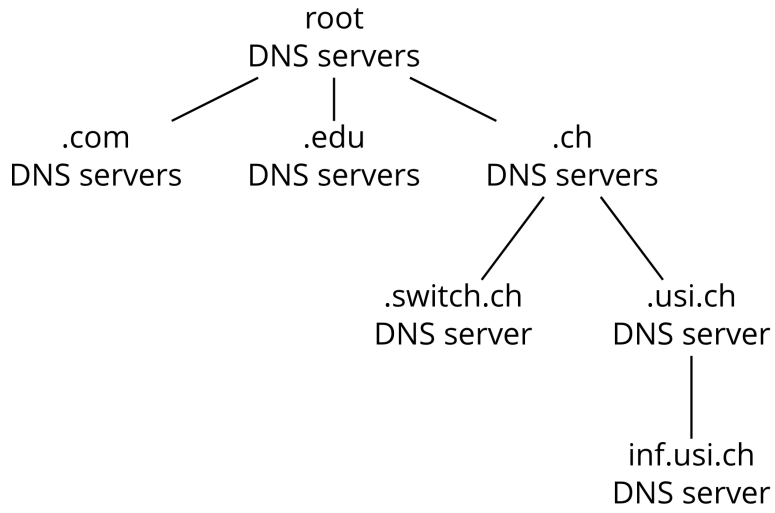
Host Names: mnemonic addresses, aliases.

The primary function and goal of the DNS is to map human readable address to network addresses.

host name ↦ *IP address*



It has a hierarchical name space and its architecture mirrors the hierarchical structure of the name space:



The root servers know only the location of the top-level servers (i.e.: .com, .edu, .ch)

For each domain there is an authoritative DNS server that holds a map of the publicly-accessible hosts within that specific domain.

DNS Caching

Performance can be poor if the DNS tree has to be traversed frequently to retrieve addresses, hence servers implement caching system to dramatically reduce the load on the infrastructure.

DNS can be seen as **databases for directory services**: a database containing **resource records** (or **RRs**)

DNS Query Types

A: main mapping which maps a *host name* ↦ *IPv4 address*

NS: a query for the name server, which maps a *domain* ↦ *authoritative name server* for that domain

CNAME: a query for the canonical name. The “primary” name for a host.

MX: a query for the **mail exchange** server for a given domain, which maps a *host name* ↦*mail server name*

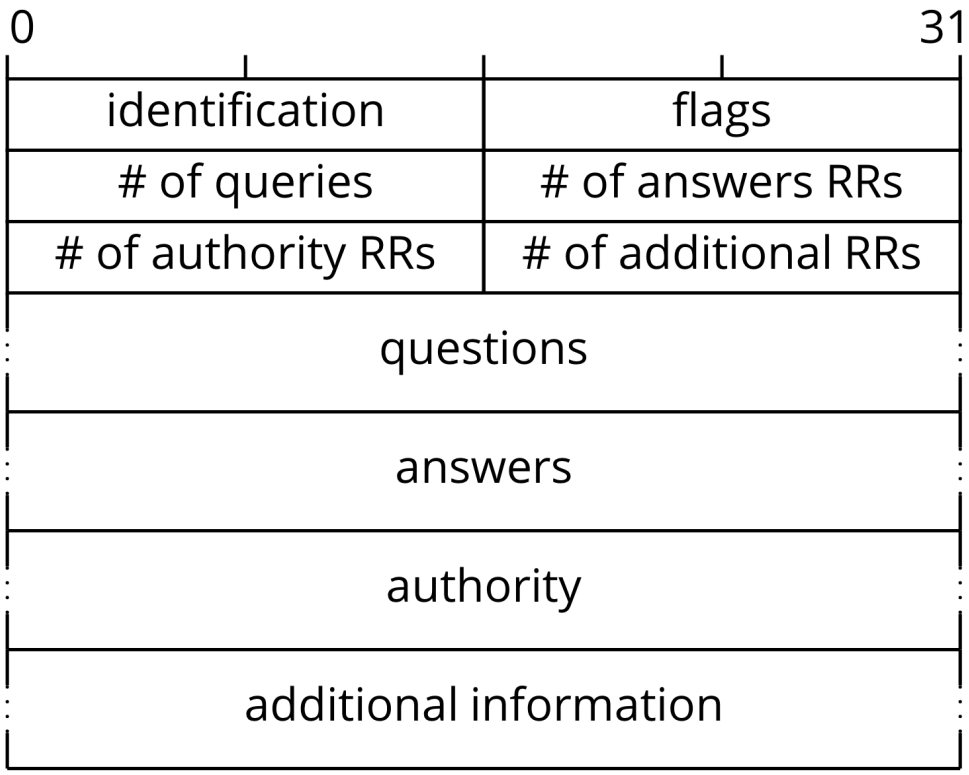
DNS Protocol

DNS is a **connection-less protocol**, which runs on top of the **UDP** transport layer, **port 53**.

Two types of messages: **query** and **reply**, which are linked in pairs by an identifier (transaction ID).

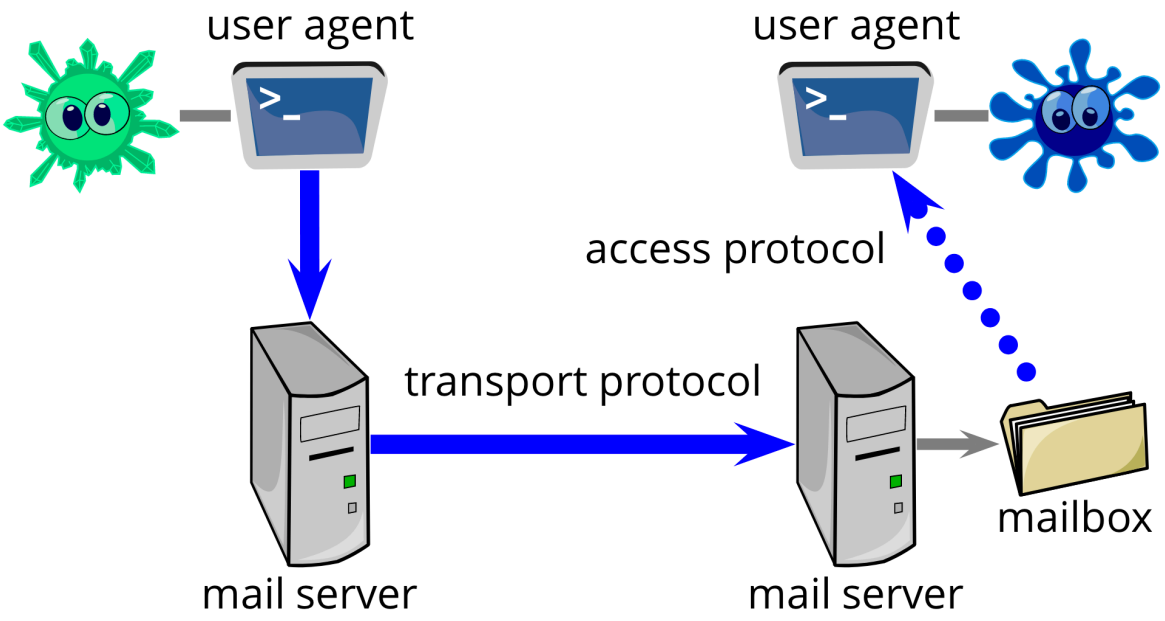
Both queries and replies have the same format, a DNS message may carry a query as well as a reply. One bit-flag within the message defines its type.

DNS Message Format



Internet Electronic Mail

E-Mail system is an asynchronous, possibly one-to-many communication system, featuring multi-media contents.



It does not require any authentication: messages can be forged, messages can be modified.

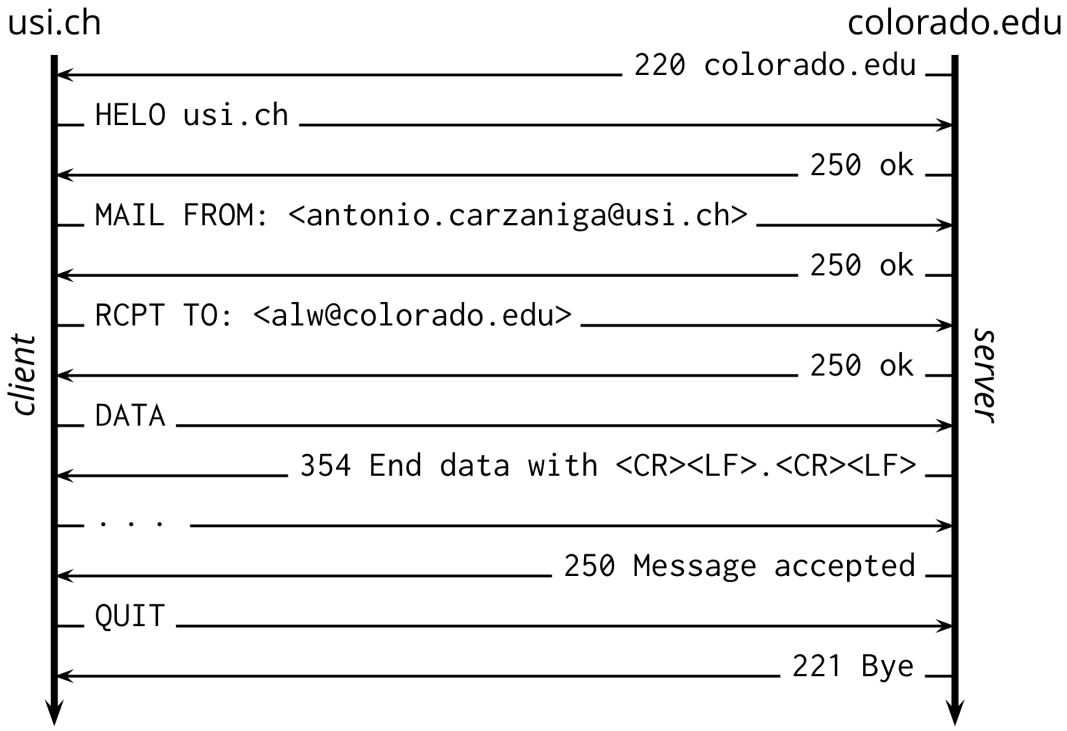
It does not guarantee any confidentiality: the message can be read by others.

Almost no delivery guarantee.

No reliable acknowledgement system.

SMTP Simply Mail Transfer Protocol

SMTP is a connection-oriented protocol.

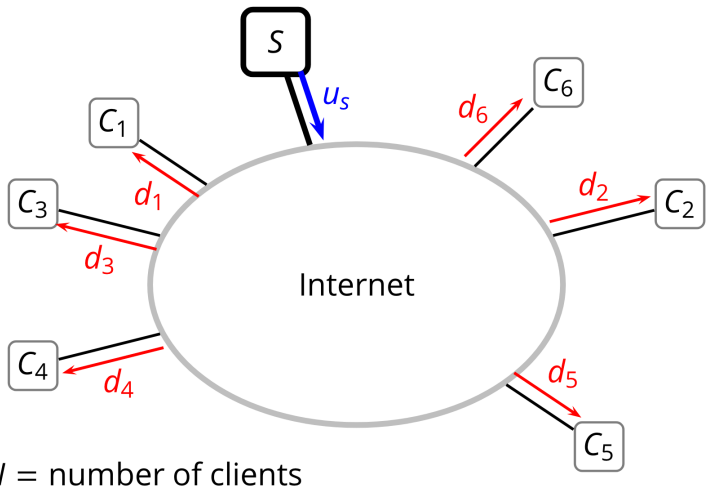


Message format:

From: antonio.carzaniga@usi.ch Date: Mon, 3 Apr 2005 16:48:22 -0600 (MDT) To: carzanig@cs.colorado.edu Subject: how to send fake e-mail messages	header lines
	empty line
Hey Dude, I heard this story about forging messages. Do you know anything about that? . . .	message body

The addresses of the message are not required to match the SMTP messages.

Transferring Big Files



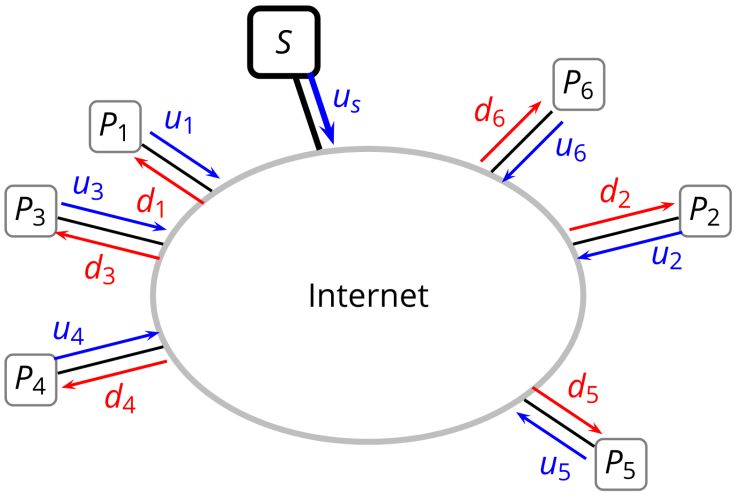
Let F = file size, N = number of clients

$$T_{CS} \geq \max\left(\frac{NF}{u_s}, \frac{F}{d_{min}}\right)$$

Each time a file is requested by a client, the server must upload the entirety of the file to the internet and the client has to download it. **This does not scale well.**

P2P Strategy

1. Split files into **blocks**
2. Simultaneously send different blocks to different clients
3. The clients exchange blocks using peer-to-peer connections



$$T_{P2P} \geq \max\left(\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right)$$

The transfer time is at least: $T_{P2P} \geq \max\left(\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right)$

Under the assumption that the upload speeds of the clients are equal:

$$T_{P2P} \geq \max\left(\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{F}{u_s/N + u_i}\right)$$

And considering a large group of peers, the term of the server upload becomes irrelevant: $T_{P2P} \geq \max\left(\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{F}{u_i}\right)$

Where the transfer time is not linearly dependent from the number of receivers

---

P2P Setup

A **tracker** keeps track of which peers are participating in the “torrent”.

A peer requires a list of peers from the tracker, it then establishes connections with them and periodically notifies the tracker that it’s still partecipating.

The **torrent** is split into **equal-sized chunks**: peers accumulate chunks and keep track of which ones they have already. Neighbouring peers exchange their lists of chunks until eventually they “trade” all their chunks.

Periodically a peer asks for the list of chunks from its neighbours and requests the rarest one first to fasten the distribution to the system of that chunk.

The Transport Layer

TCP Transfer Control Protocol: a connection-oriented protocol to establish connections between end-systems.

UDP User Diagram Protocol: a connectionless protocol to exchange messages between end-systems.

The transport layer packets are called **segments**.

Using the following assumptions about the underlaying network layer:

- No guarantees on the integrity of the segments
- No guarantees on the order in which the segments are delivered

Transport Layer Service

Connects applications running on hosts, as opposed to connecting hosts.

Transfers data reliably, conserving integrity and possibly order delivery.

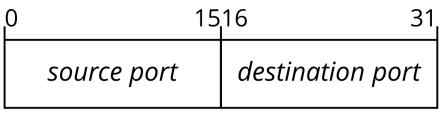
Establish connections, which can be used as streams $\Longrightarrow$ ordered delivery

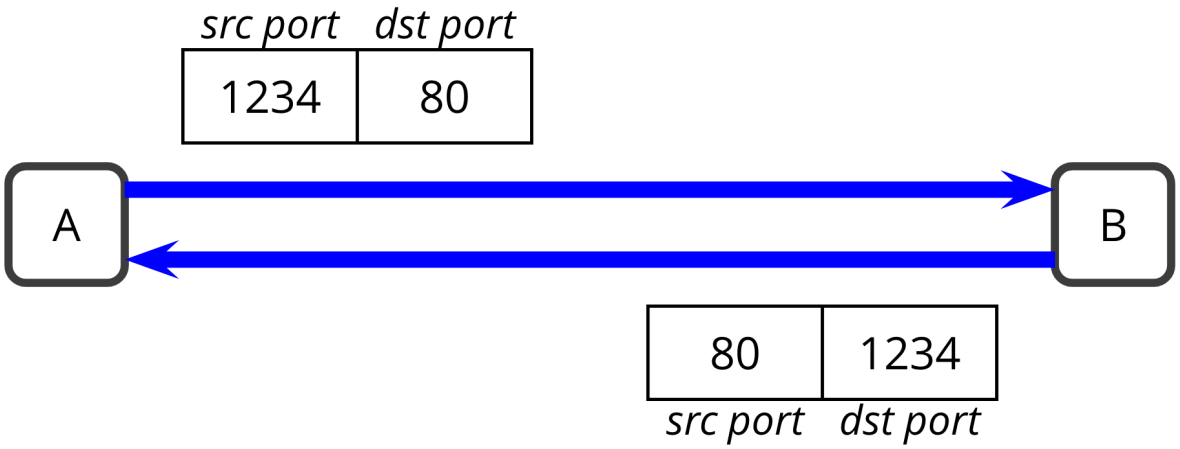
Allow the congestion control: end-to-end traffic control (admission), to avoid destructive congestions within the network.

Ports

To identify a connection between two applications running on two (possibly) different hosts, it uses two pairs (*IP address, port*).

The message format for both UDP and TCP starts with the source and destination port numbers:

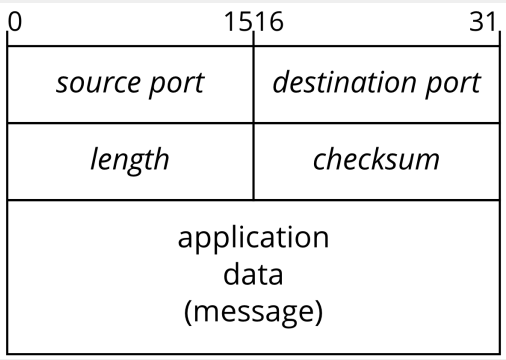




The UDP message format is very simple. UDP provides the two most basic functionalities of a transport protocol:

- Identifies two applications
- Integrity check via checksum

UDP Packet Format



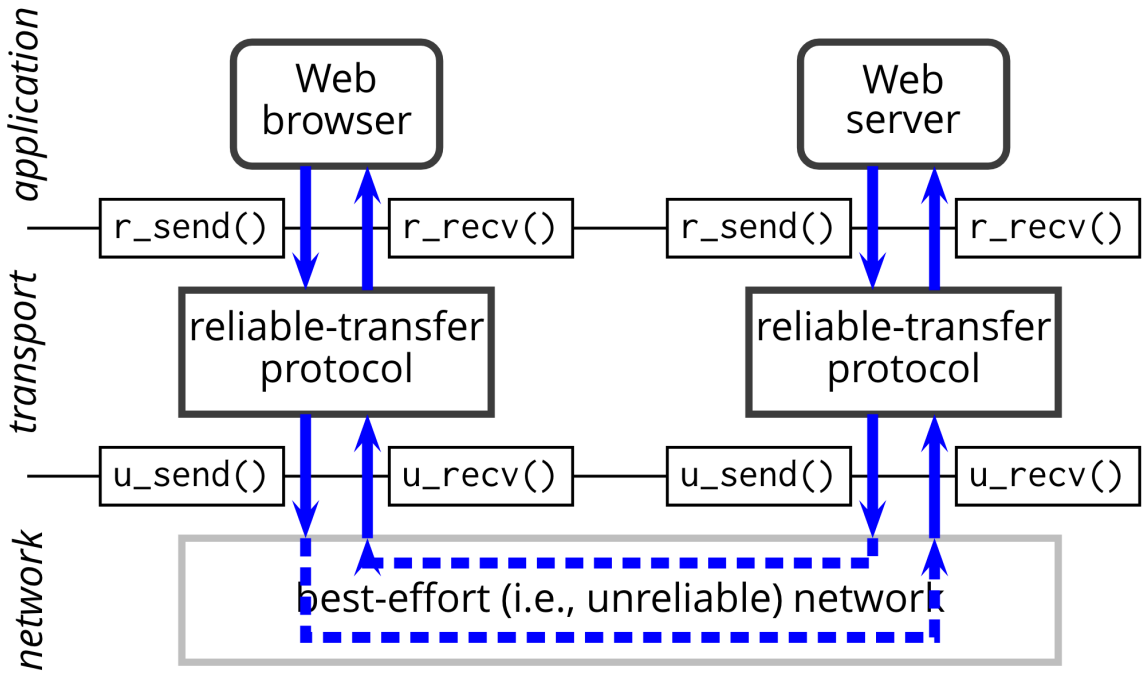
Finite State Machines

States represent **states of a protocol**.

Transitions are characterised by an **event/action** label:

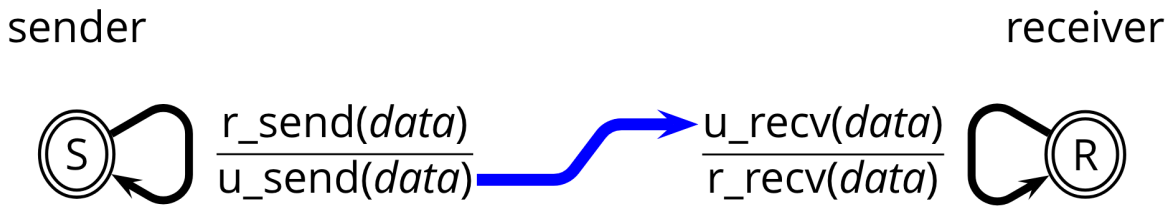
- **event:** typically an **input message** or a **timeout**
- **action:** typically an **output message**

Reliable Data Transfer



Baseline Protocol

Reliable transfer protocol on top of a reliable network:

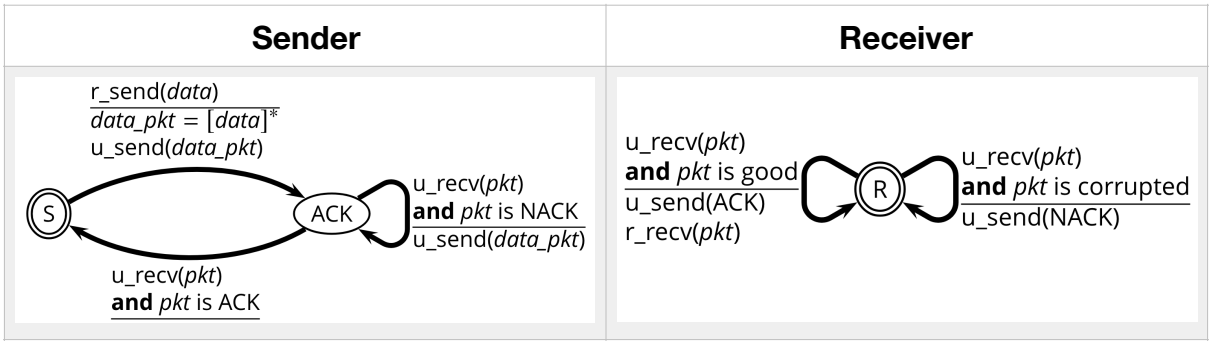


Bit Errors

Reliable data transfer must feature **a way to detect error** (***error detection***, the receiver must be able to tell when a package received is corrupted) and a way to notify the sender that a packet was corrupted (***receiver feedback***). The sender should then be able to retransmit the corrupted segments (***retransmission***).

Noisy Channel

$[data]*$ indicates the packet containing data **and** an error-detection code.



This protocol is “synchronous” or “stop-and-wait” for each segment: meaning that the sender must await for a positive acknowledgment before it can send the next segment.

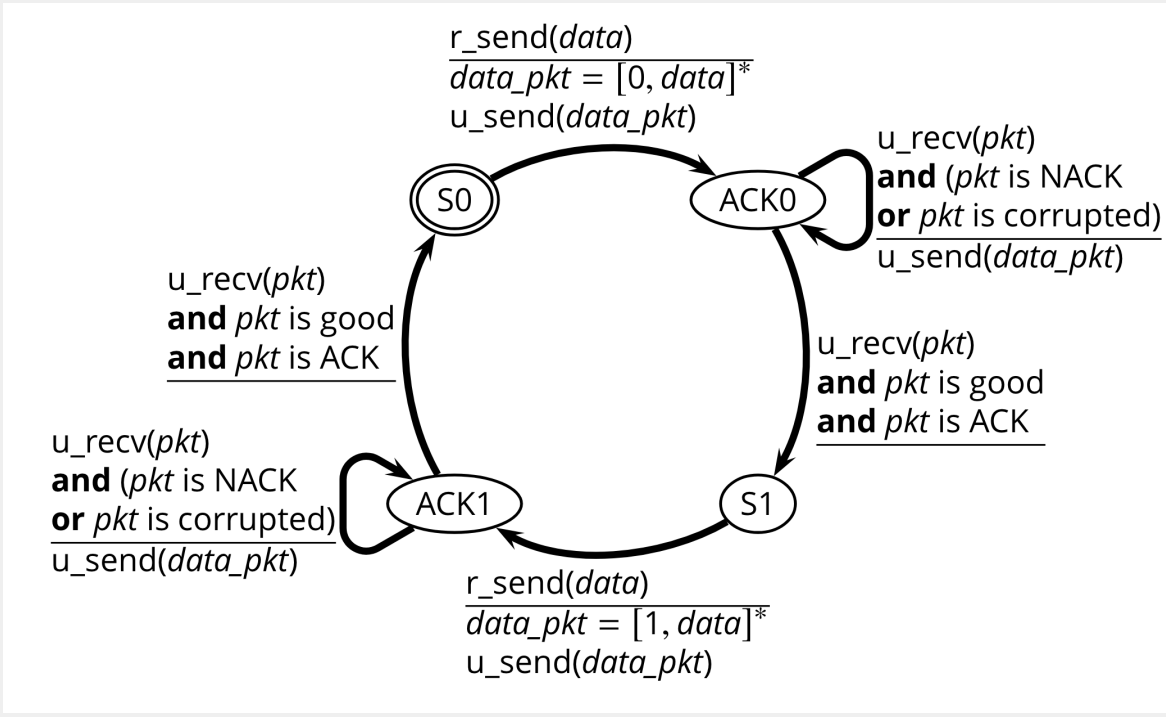
This protocol does not deal with bad acknowledgements, that is, an error occurred within the ACK/NACK segment.

Dealing with duplicate segments

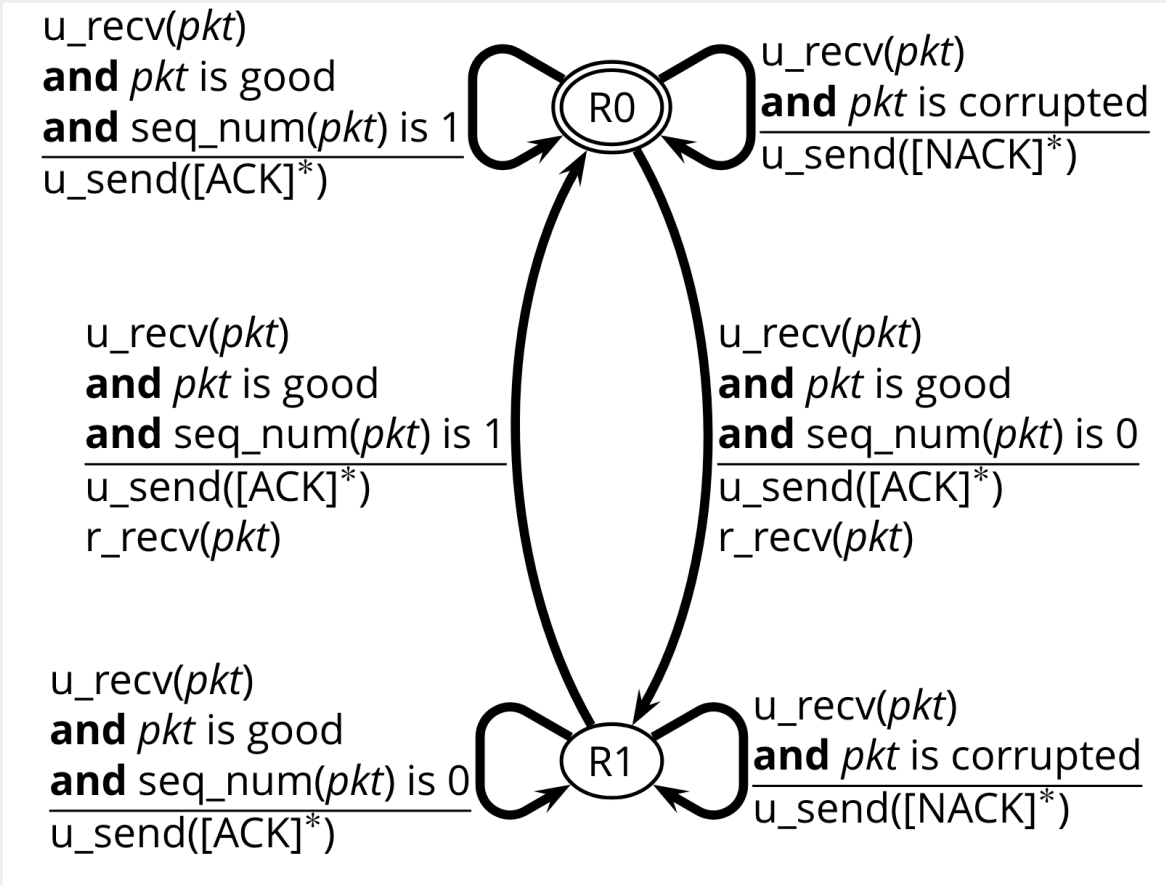
The sender adds a sequence number to each segment, so that the receiver can tell if it’s a retransmission.

For a stop and wait protocol, one bit is sufficient to distinguish between two consecutive segments.

Sender



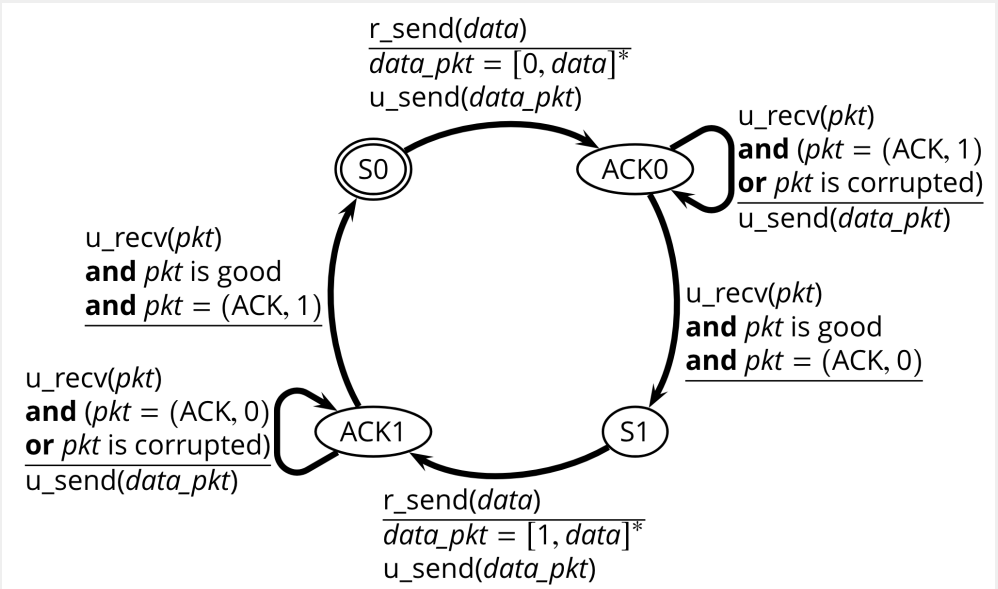
Receiver



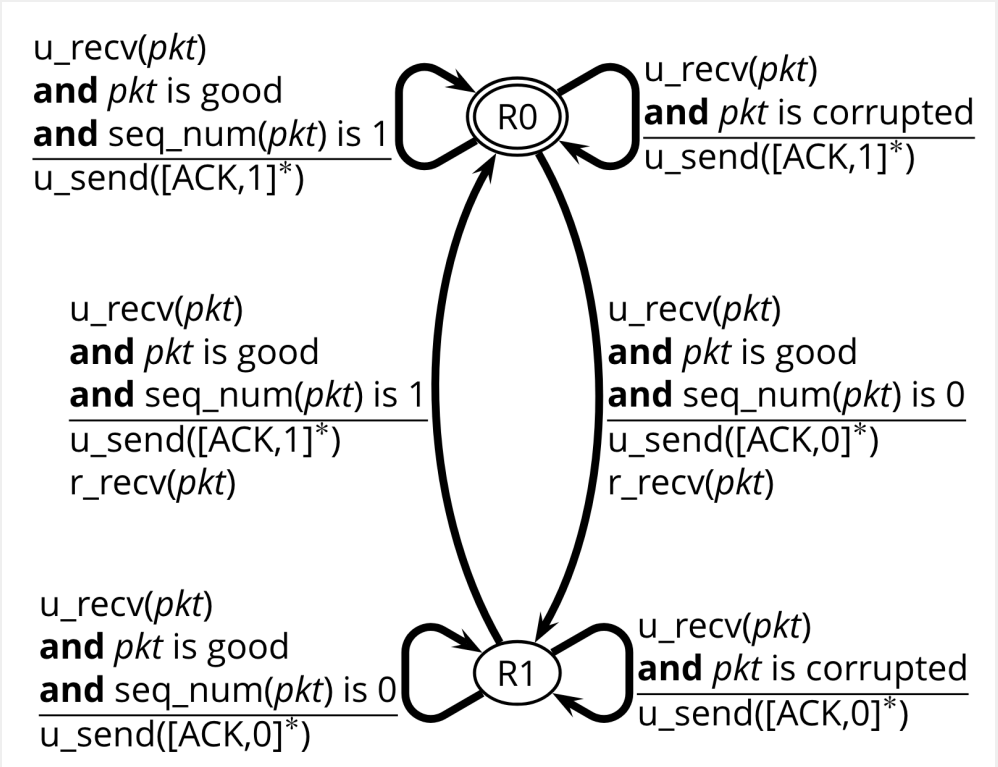
Better use of ACKs

Instead of sending generic ACK and NACK segments, now that we are using sequence identifiers, we can send an ACK segment for the last segment that we successfully received. The sender, will then know that if it received ACK(*n-1*) and he sent already *n*, he'll have to retransmit segment *n*

Sender



Receiver



Summary of Techniques

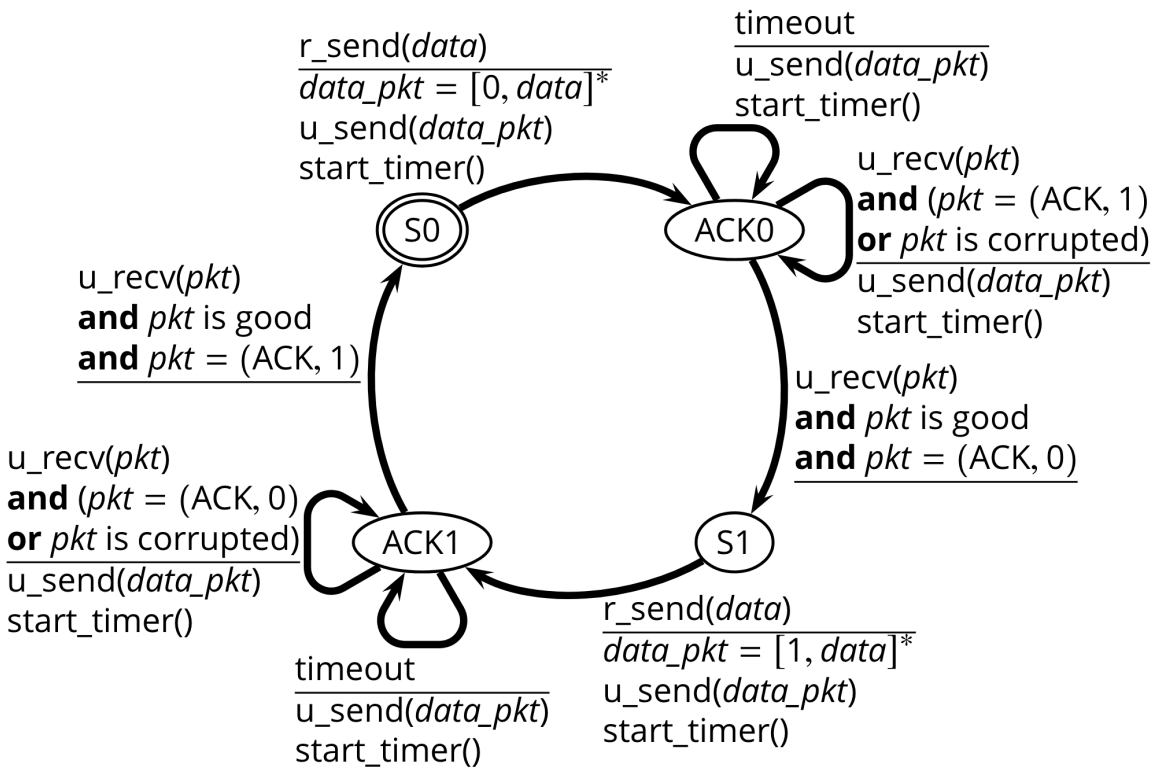
- **Error detection codes** can be used to detect corrupted segments.
- **Retransmissions** allow to correct for transmission errors
- **ACK** and **NACK** give feedback to the sender about the transmission success, also protected by error detecting codes.
- **Corrupted ACKs** are interpreted as **NACKs**, possibly generating duplicate segments.
- **Sequence Numbers** allow the receiver to ignore duplicate data segments.

Noisy and Lossy Channel

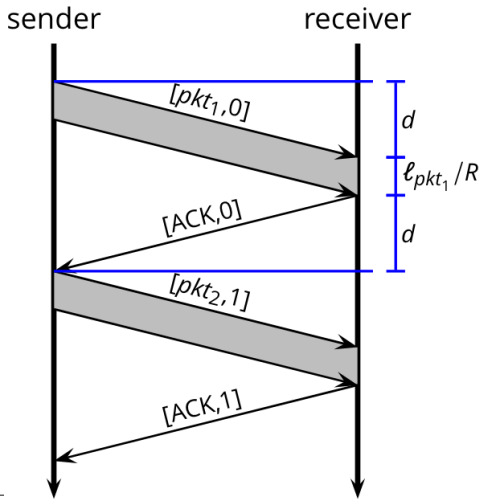
If a channel is also lossy, meaning that some segments might fail to be delivered altogether, **the sender must be able to tell that a packet was lost** on its way to the receiver.

Lost packets can be treated in the same way we treated corrupted packets.

Using Timeouts



Poor Utilisation: Stop-And-Wait Protocol

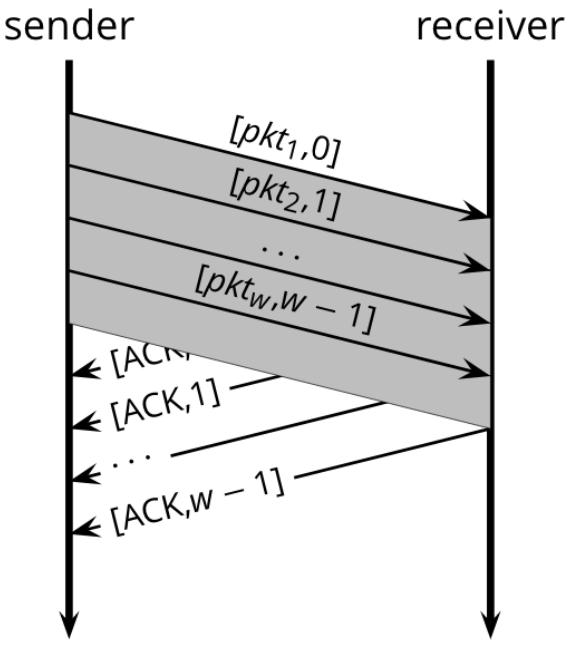


The factor that indicates the utilisation of the channel is:

$$U = \frac{l_{pkt}/R}{2d + l_{pkt}/R}$$

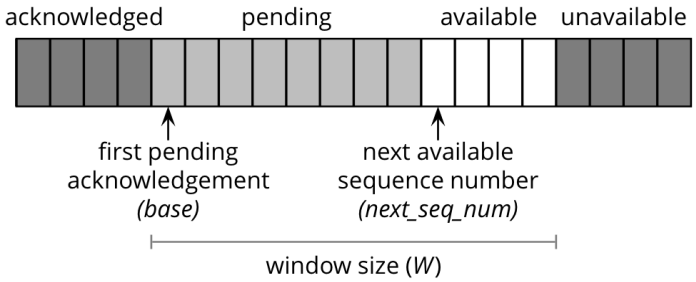
To achieve a better utilisation factor we transmit multiple packets without waiting for an acknowledgment

Go-Back-N Protocol



The sender has up to *W* unacknowledged segments in the pipeline $\Longrightarrow$ the finite state machine becomes very complicated.

We represent the sender's state with its queue of acknowledgements.



The sender has two counters, one segment buffer and one timer:

- A counter for the first sequence number that hasn't been acknowledged yet (the oldest unacknowledged)
- A counter for the last sequence number that it has sent (the most recently sent)
- The buffer of segments to store the acknowledged, pending, available segment acknowledgements.
- The timer to measure a timeout, the amount of time after which no reply from the receiver got back to the sender.

The receiver has only one counter and no buffer of segments, keeping track of the expected sequence number of the segment that he should receive.

He only has to wait for an uncorrupted segment with the expected sequence number.

The receiver than either acknowledges the expected sequence number or sends back the acknowledgement for the previous sequence number that of the segment was correctly received.