Høyskolen
Kristiania

Digital Technology

TK1104

Guest lecturer: Peyman Teymoori

peymant@ifi.uio.no

Lecturer: Toktam Ramezanifarkhani

Toktam.Ramezanifarkhani@kristiania.no

Toktamr@ifi.uio.no

Computer Networks – Transport Layer

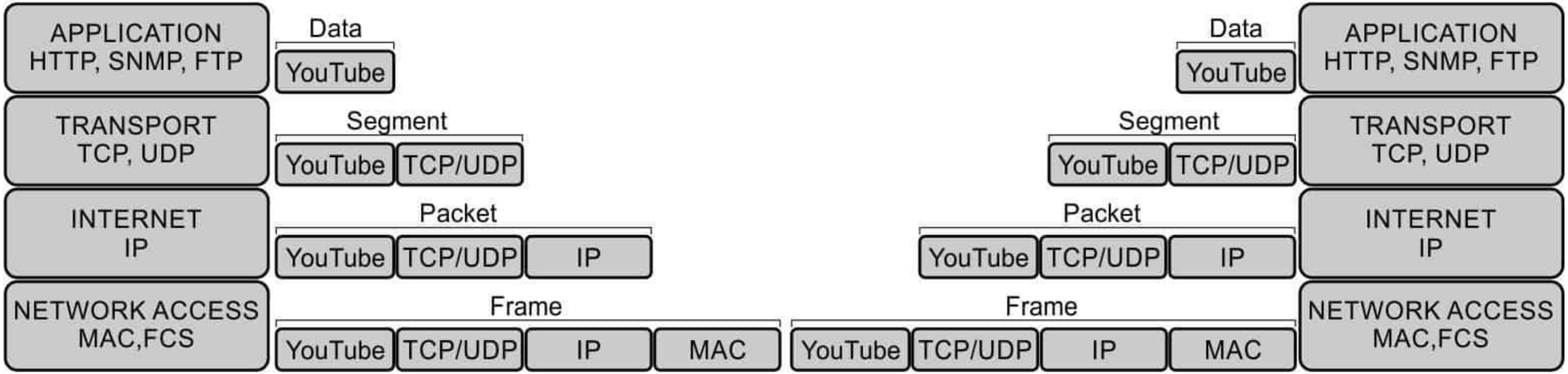
Høyskolen Kristiania

Learning objectives

Høyskolen Kristiania

- To learn the transport layer services:
  - TCP, UDP
- To learn about multiplexing/demultiplexing

Recap: Packet Encapsulation

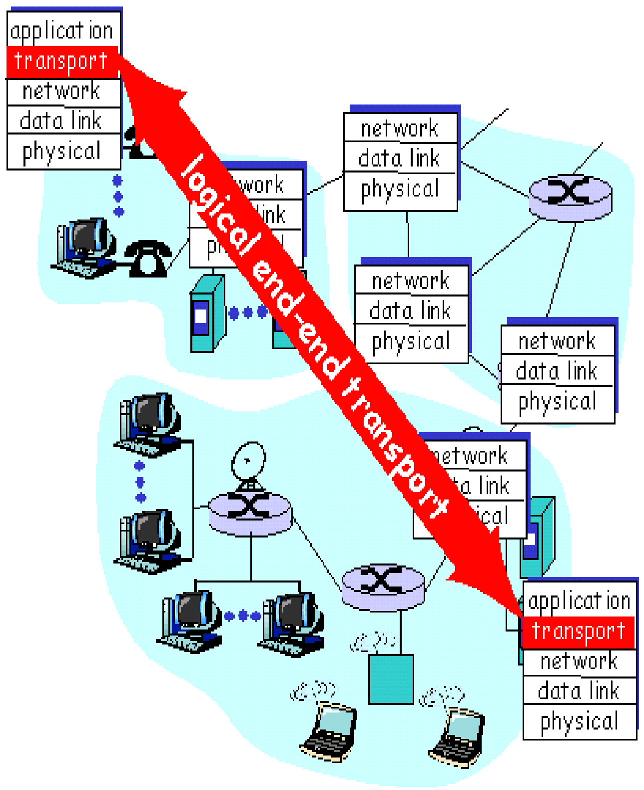


Layer´s name	designation of transmission unit	Most important tasks / functions Example of protocols / standards
Application layer	Message	Support network applications Ex: HTTP, DNS, FTP, SMTP, POP3
Transport layer	Segment	transport of application layer messages between client and server pages of an application: including mux / demux, different levels of reliability and more .. Ex: TCP, UDP ...
The network layer	Datagram	routing of datagram from / to host through the network core Ex: IP (v4 and v6) ICMP, RIP, OSPF, BGP
The data line layer	Frame	(Reliable) delivery of frame from neighbor node to neighbor node. Ex: Ethernet II, FDDI, IEEE 802.11
Physical	Bit	(Code and) Move single bit between communication partners. Ex: 10BaseT,

Transport layer

Transport service

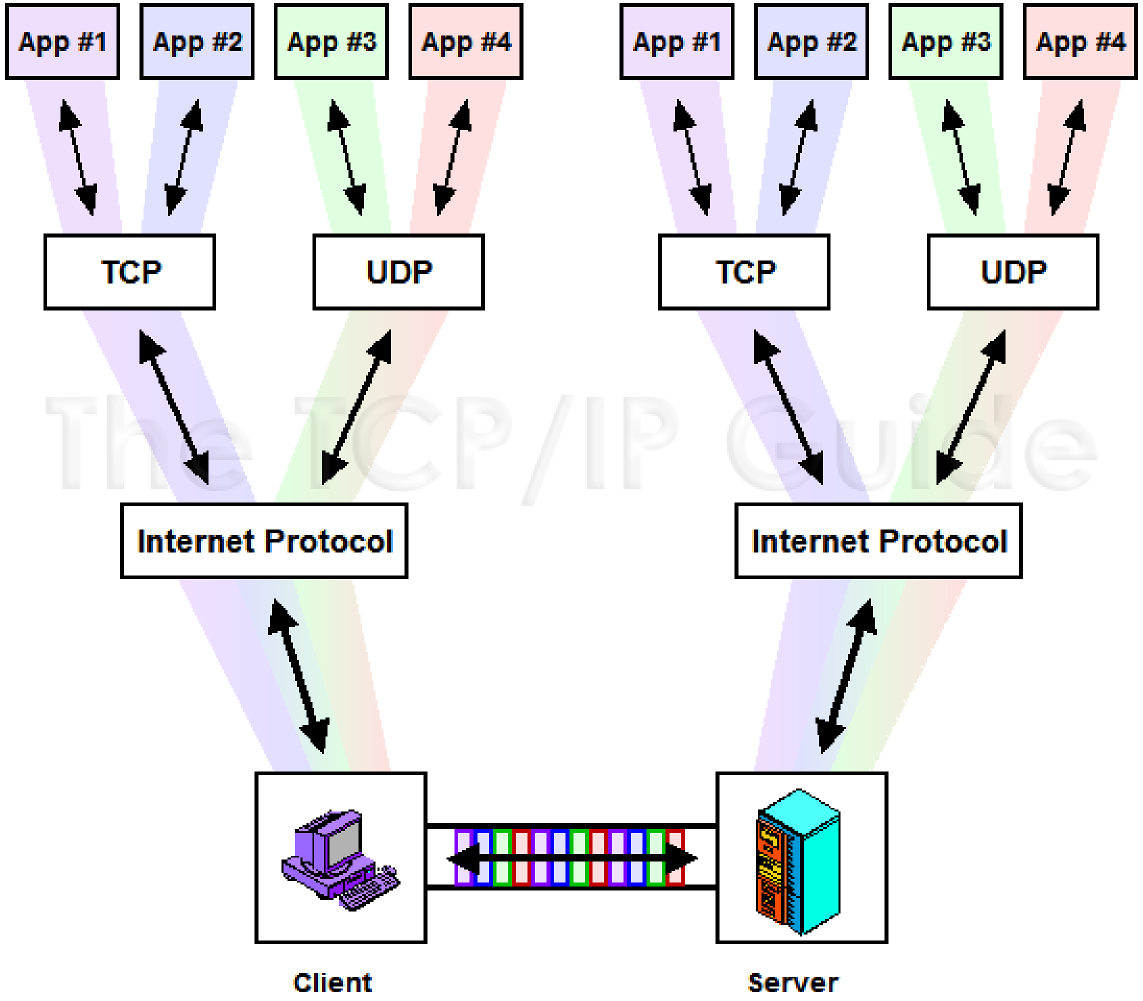
- Sets up logical communication between application processes on different clients
- The transport protocol is run in each end system
- Data transfer between processes



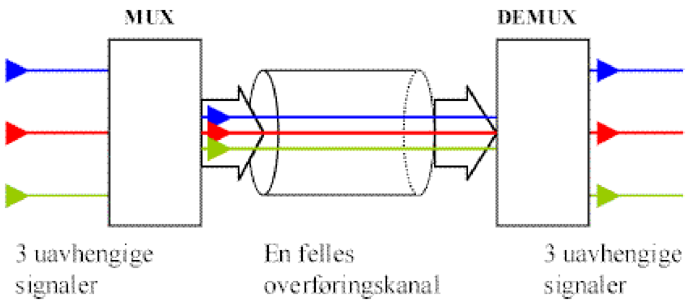
Transport-Layer Protocols

- The internet uses the network-protocol IP
  - does its best, but gives no guarantees
  - «Best effort»
- UDP
  - Sends a datagram, which may consist of several parts, to the recipient and hopes it arrives
  - Improves IP only with end-to-end control and error-checking
- TCP
  - Creates a "fixed" connection.
  - Adds move-control,
    sequence-number,
    time control,
    error checking and
    traffic-cork control (congestion control)

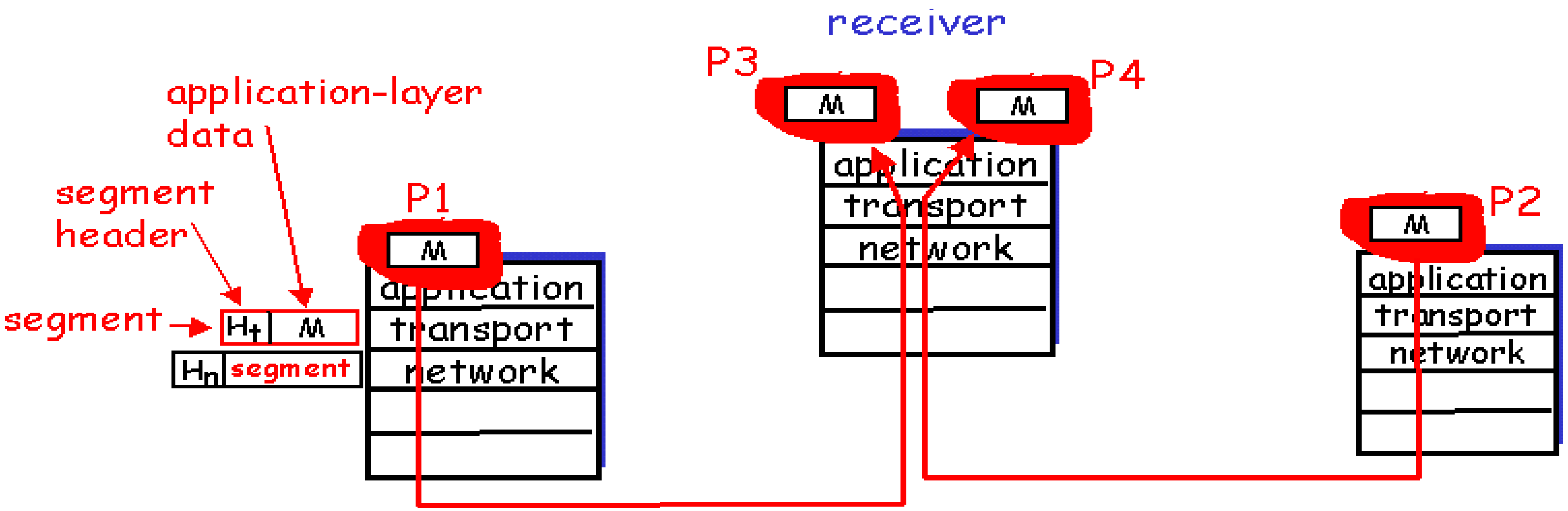
Multiplexing/ demultiplexing



Multiplexing/ demultiplexing

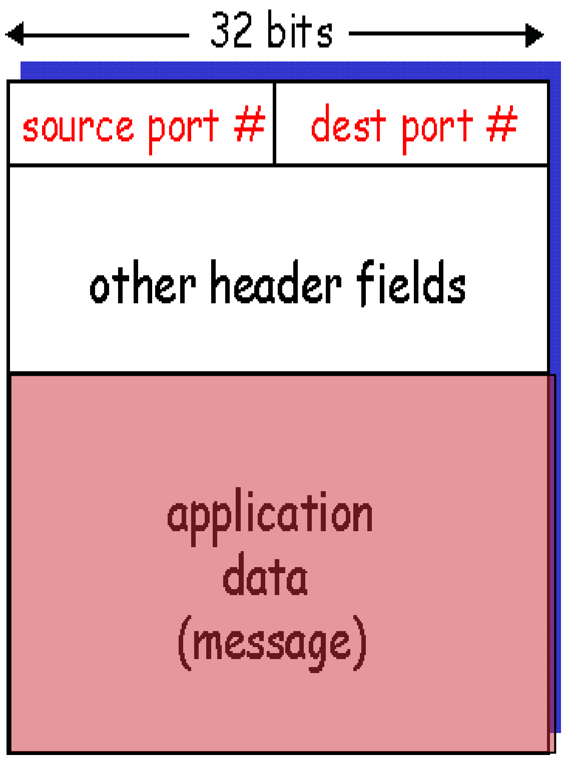


- Segment
  - Data unit exchanged between the transport layers
  - TPDU (Transport Protocol Data Unit).
- Demultiplexing
  - Deliver received segments to the correct process



Multiplexing <port number

- Collects data from application processes and packs them with a header.
- The header contains the port number of the **sender** and **receiver**.
- Port number = **16 bit** unsigned **integer**
- The ports **0-1023** are «**well known**» (RFC 1700)
    - Secure Shell: port 22
    - SMTP: port 25
    - DNS: port 53
    - HTTP: port 80
    - HTTP over TLS/SSL: port 443
- Other ports are divided into:
    - **Registered:**
        - 1024-49151 (0x0400-0xBFFF)
        - Can be used for other purposes as well, but **is registered** for a service with IANA.
    - **Private / Dynamic:**
        - 49152-65535 (0xC000-0xFFFF)



TCP/UDP segment format

Transport layer 1

Høyskolen Kristiania

- The first main task that is solved on the transport layer is thus to multiplex / demultiplex from / to local processes and the common channel (Internet)!

- Port number acts as **ID number** for local and contacted process!

Høyskolen Kristiania

User Datagram Protocol

UDP (User Datagram Protocol)

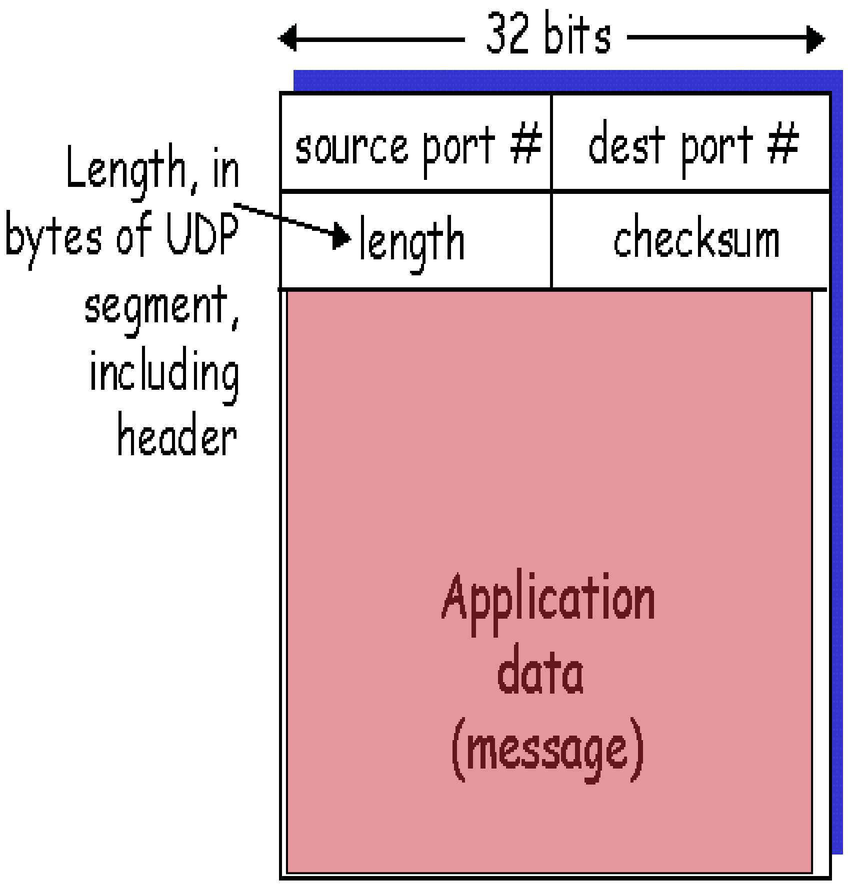
- Very simple Internet transport layer protocol
  - Segments may be lost
  - Segments can be delivered in the wrong order
  - Do not handshake between sender and receiver.
- Why UDP?
  - No connection established no delay
  - Does not set up a common state for sender and receiver
  - Small segment head
  - No traffic jam control.
  - Enables broadcasting
  - Should you need reliability,
    - you can add it to the program at the application layer level

put the pedal to the metal

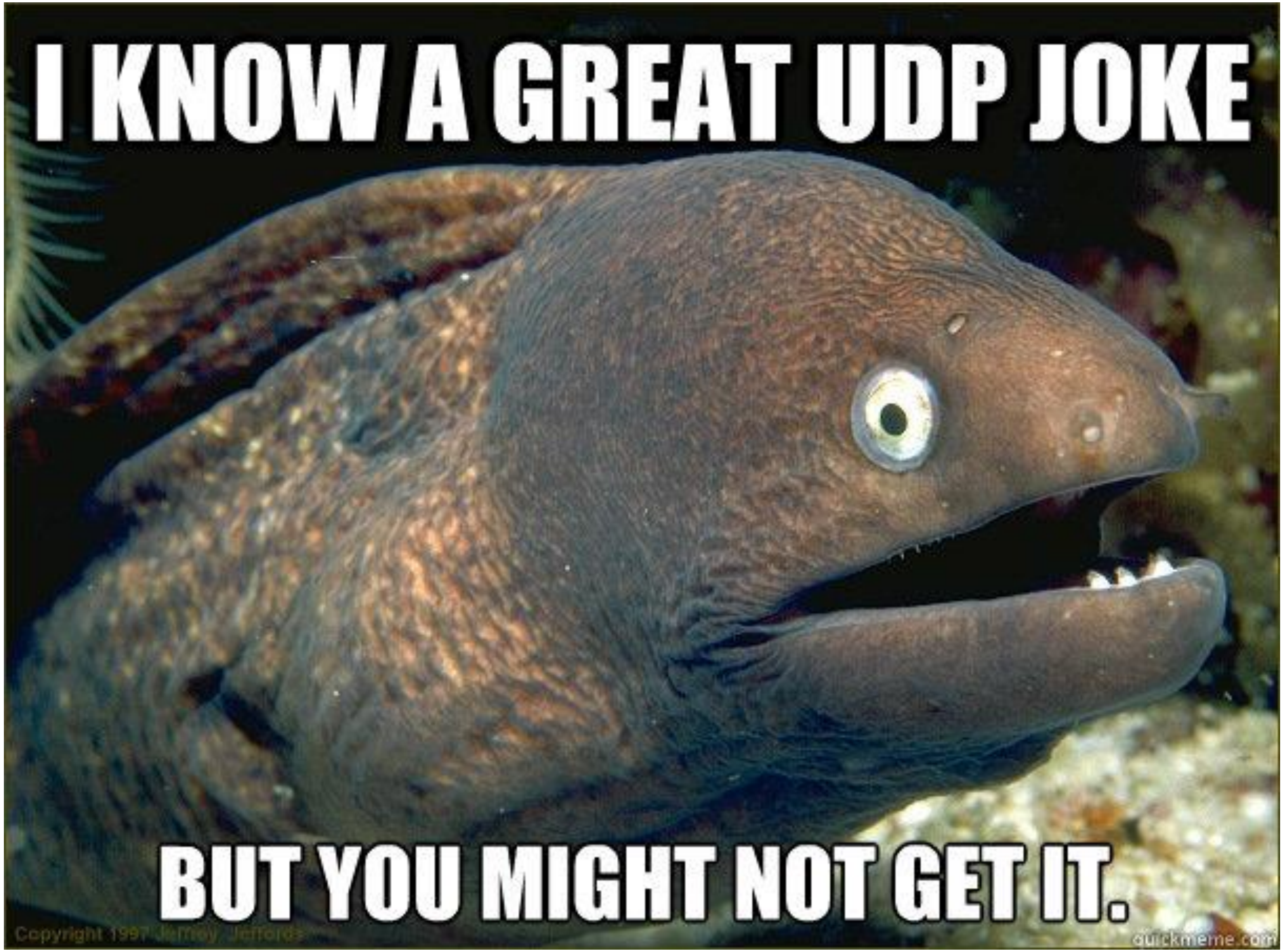
UDP

Høyskolen Kristiania

- Often used in connection with multimedia where the human brain can correct the errors
- Other uses
  - DNS
  - SNMP
  - The receiver's application can provide error handling



UDP segment format


I KNOW A GREAT UDP JOKE
BUT YOU MIGHT NOT GET IT.
Copyright 1997 Jeffrey Jeffords
quickmeme.com

Transmission Control Protocol

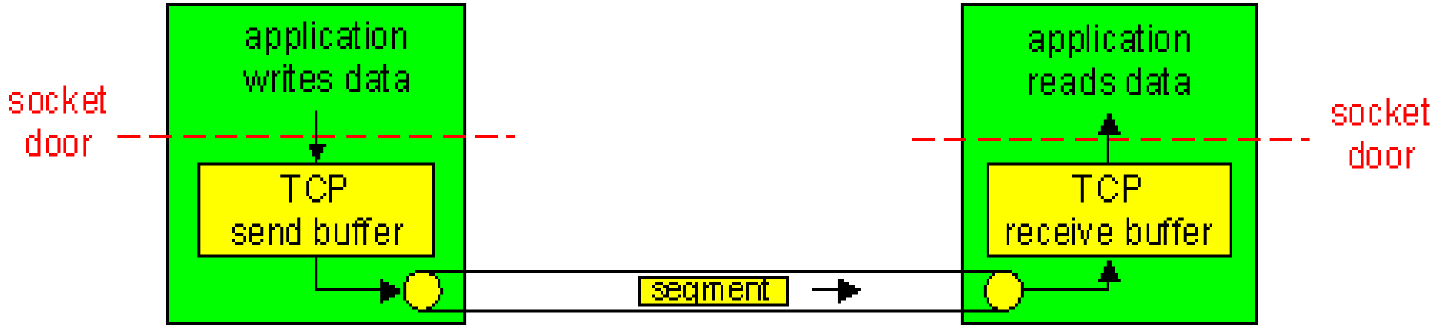
See the «theory» behind TCP / IP at the end of the slide deck for those who are interested

Here we will focus on how TCP works in practice

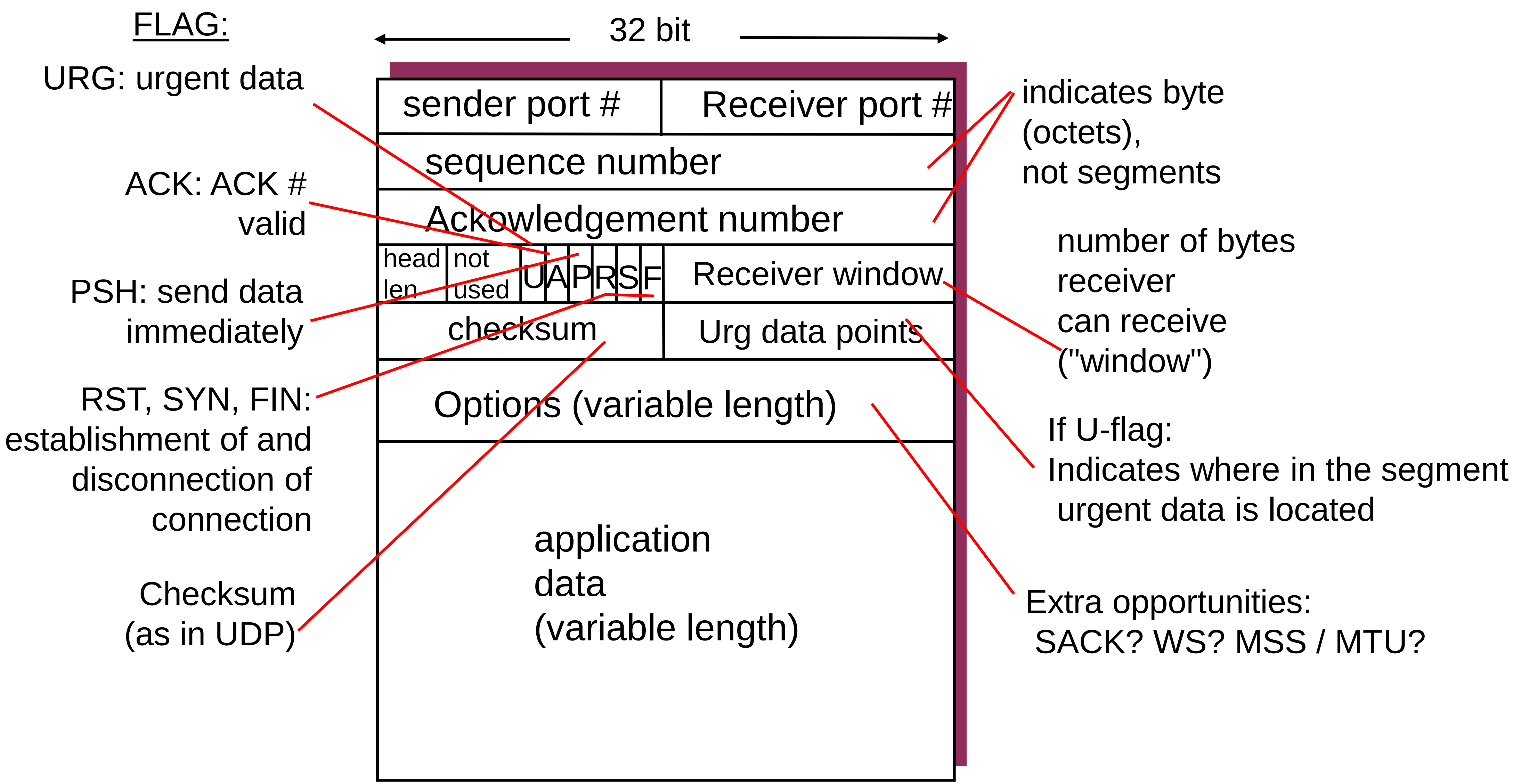
Høyskolen Kristiania

TCP (Transmission Control Protocol)

- Point to point
  - One sender, one receiver
- Reliable, arranged byte-stream
- Pipeline
  - Flow and congestion control determines window size.
- Sender and receiver buffer
- Full duplex data
  - Both can send and receive at the same time.
- Connection-oriented
  - Handshake before data transfer
- Flow control
  - The sender does not drown the recipient

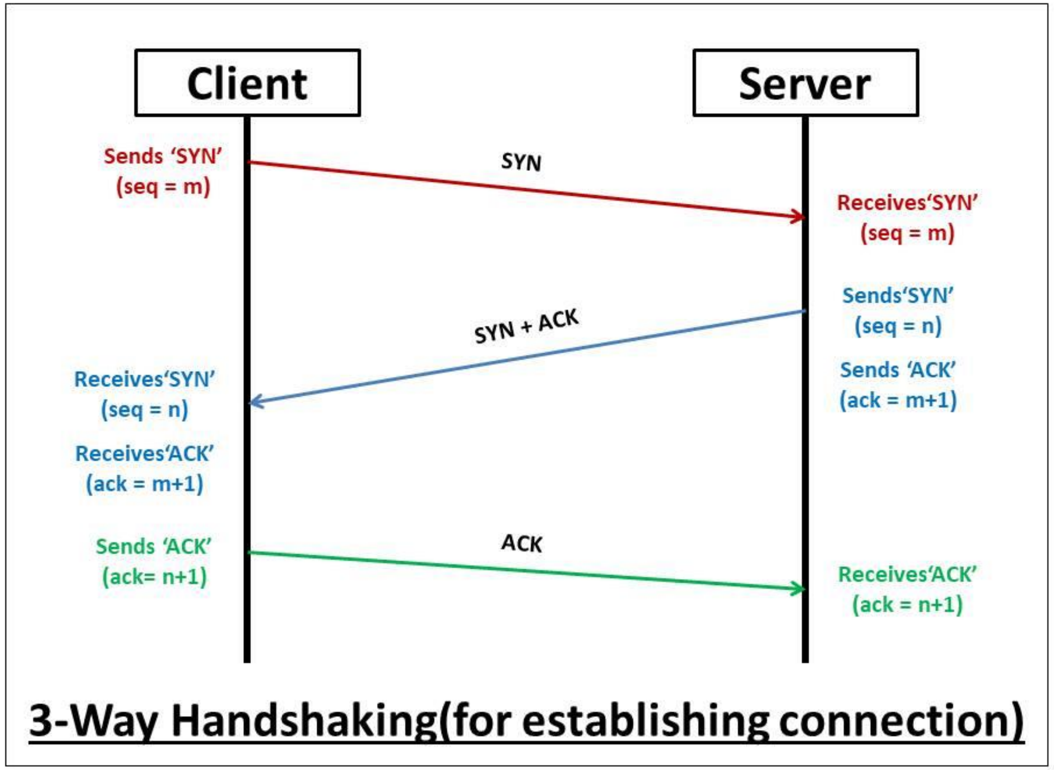


TCP header structure

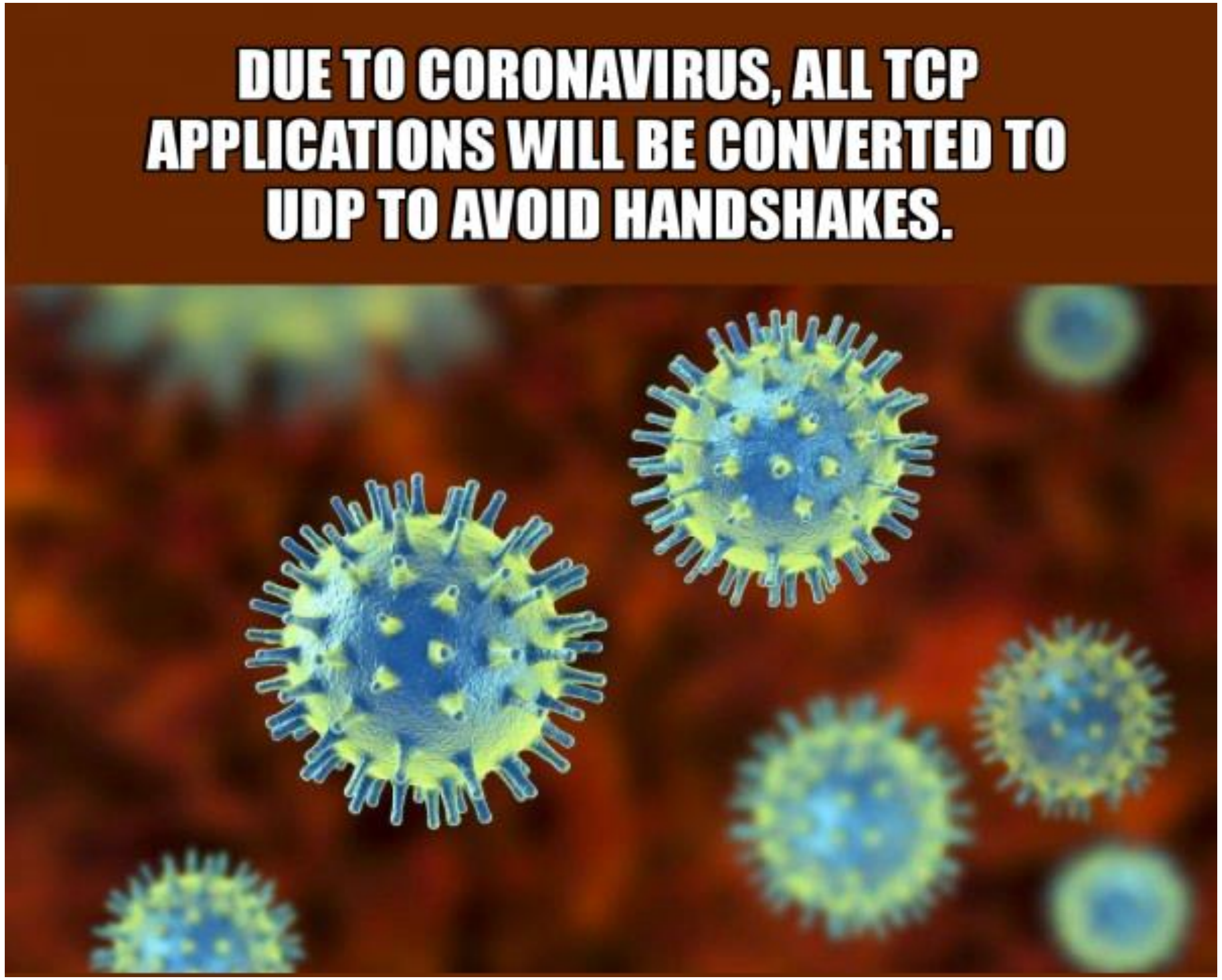


TCP: Connection startup

- Transmitter and receiver establish a connection before data segments are exchanged.
    - Initializes TCP variables
        - Sequence number, buffers, windows .....
- Client sends a special TCP segment with SYN
    - The SYN flag in the header set
    - Specifies the start sequence number
- Server responds with SYN+ACK
    - SYN and the ACK flags in the header set
    - Sets up start sequence number, buffers, windows etc.
- Client responds with ACK



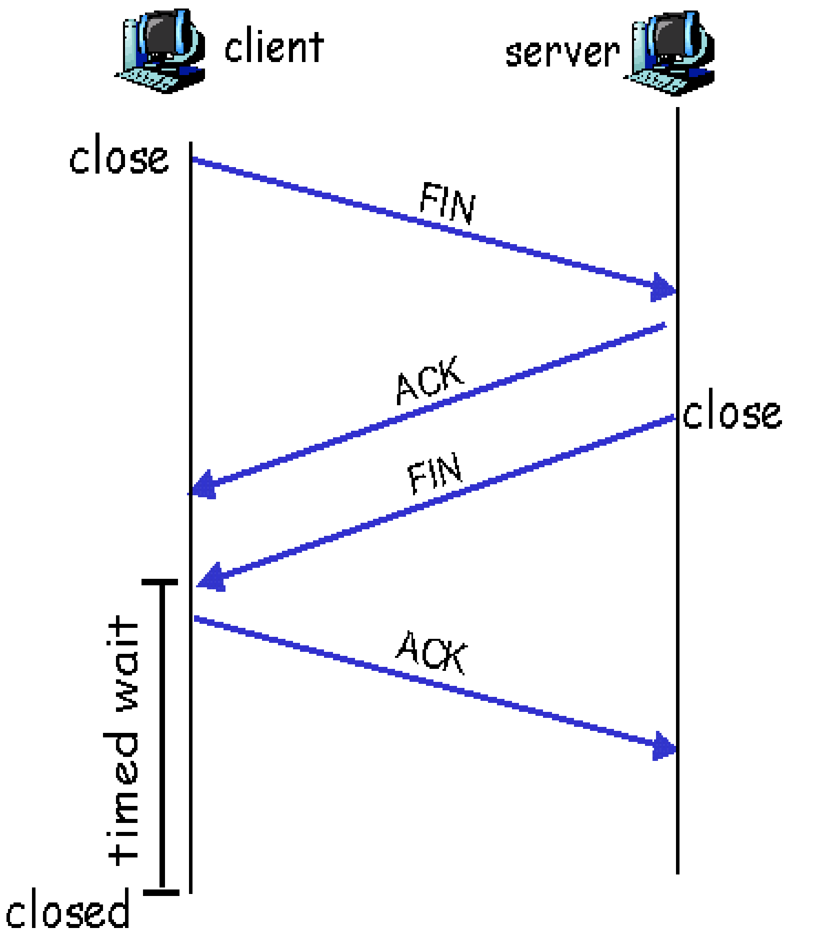
Pandemic-related News!



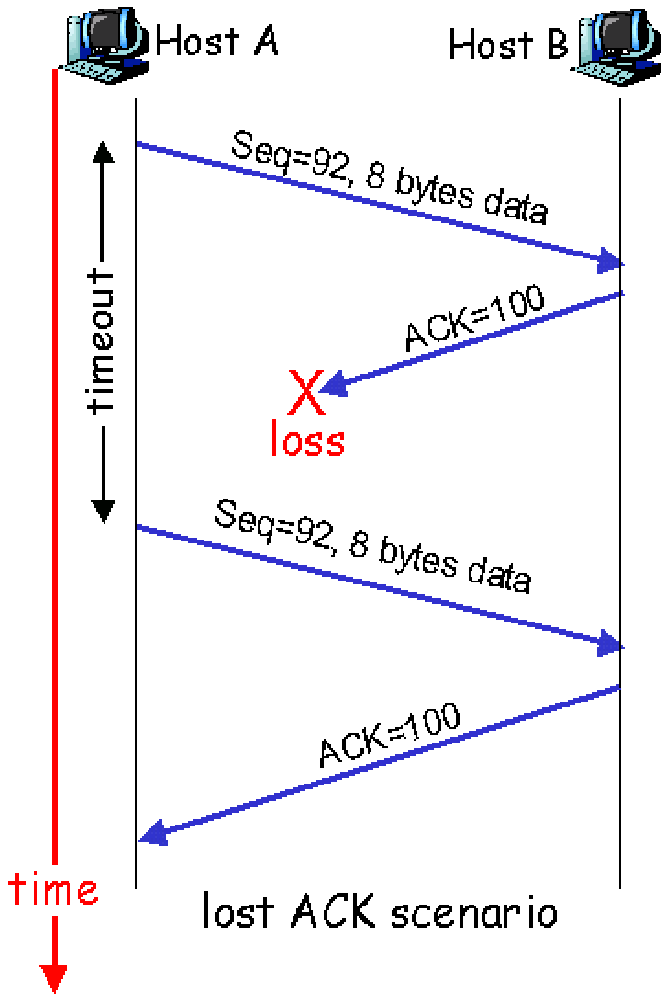
Finishing the connection

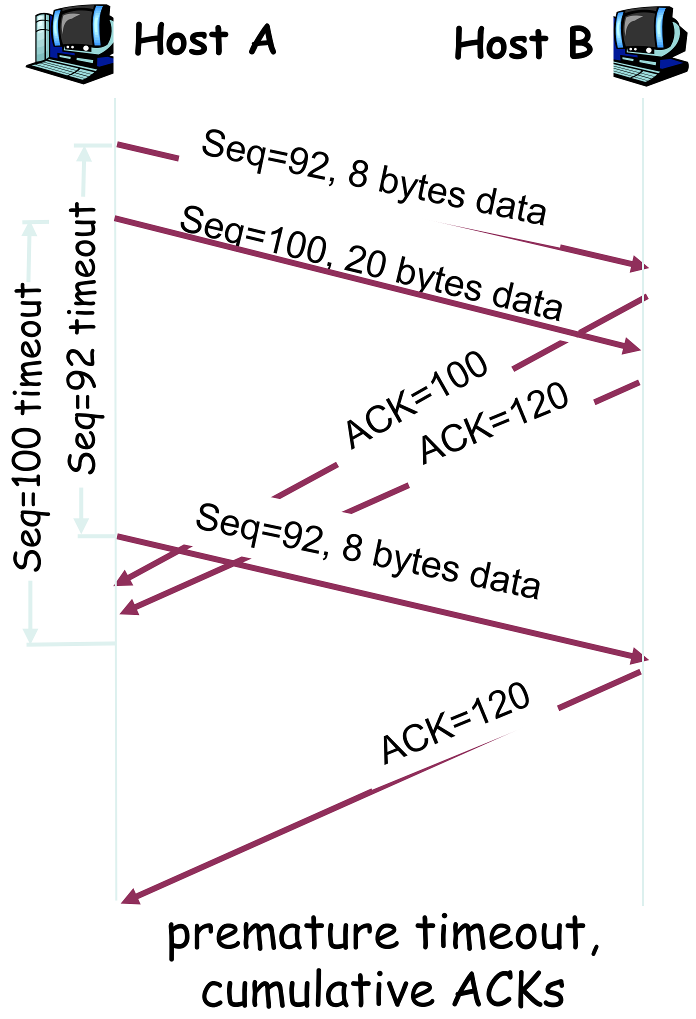
- Client app closes the socket.
- Client-OS sends TCP FIN to server
- Server-OS receives FIN, sends ACK
- Server-app closes socket.
- Server-OS sends FIN to client
- Client-OS receives FIN, sends ACK
- Server-OS receives ACK
- The connection ended.

- NB! Other methods are also used !!
  - For example, the RESET flag (from Server)
  - Three Way: FIN, FIN + ACK, ACK



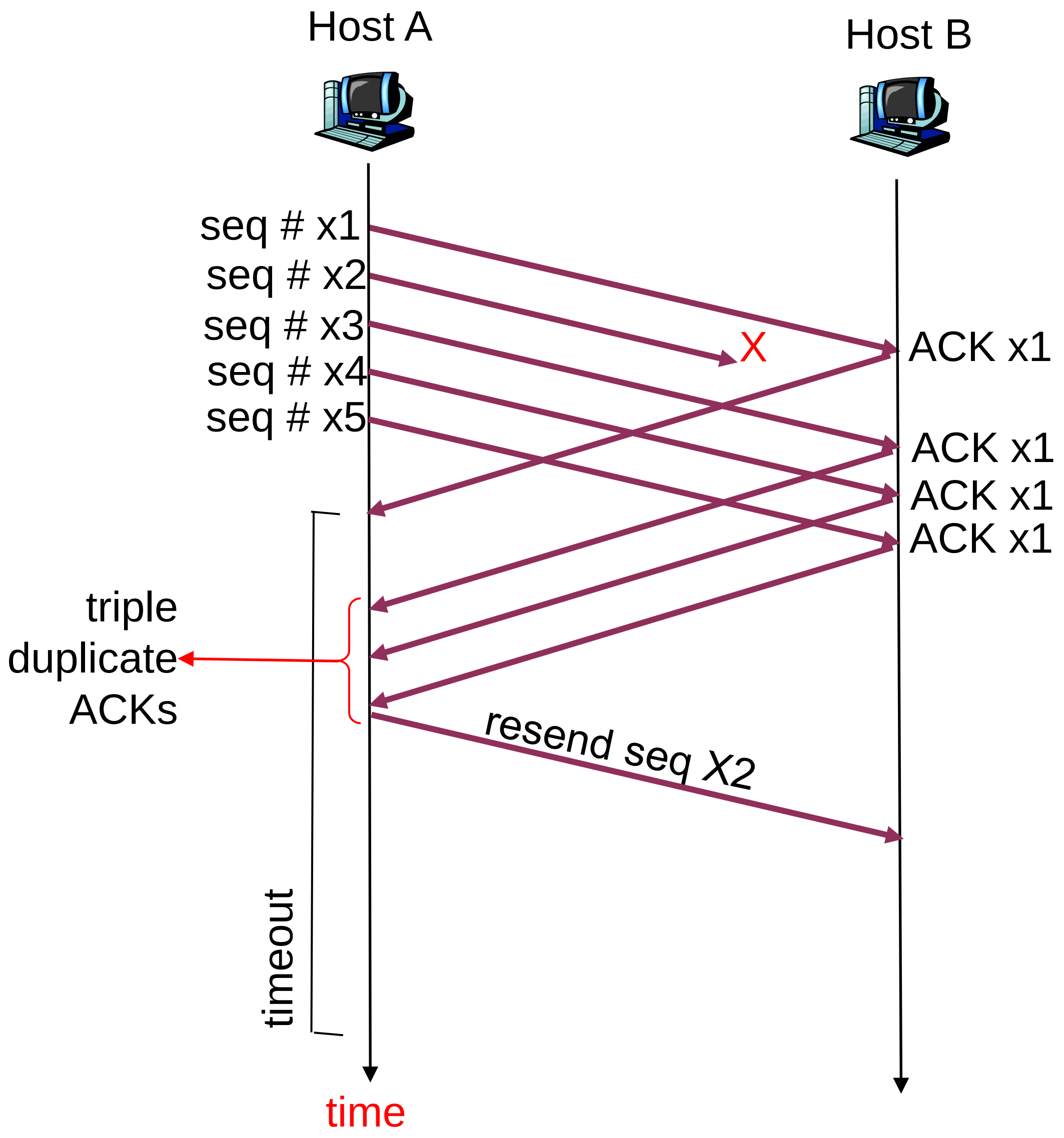
Resending


Host A
Host B
Seq=92, 8 bytes data
ACK=100
X
loss
timeout
Seq=92, 8 bytes data
ACK=100
time
lost ACK scenario


Host A
Host B
Seq=92, 8 bytes data
Seq=100, 20 bytes data
ACK=100
ACK=120
Seq=92 timeout
Seq=100 timeout
Seq=92, 8 bytes data
ACK=120
premature timeout,
cumulative ACKs

Fast shipping

Høyskolen Kristiania



- The timeout period is often relatively long
- If the sender receives 3 ACK on the same data before timeout, it is interpreted as packet loss -> retransmission of the following segment

How to tell a joke in a TCP vs. UDP style!

TCP:

- Hi, I'd like to hear a TCP joke.
- Hello, would you like to hear a TCP joke?
- Yes, I'd like to hear a TCP joke.
- OK, I'll tell you a TCP joke.
- Ok, I will hear a TCP joke.
- Are you ready to hear a TCP joke?
- Yes, I am ready to hear a TCP joke.
- Ok, I am about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline.
- Ok, I am ready to get your TCP joke that will last 10 seconds, has two characters, does not have an explicit setting, and ends with a punchline.

- I'm sorry, your connection has timed out....

- Hello, would you like to hear a TCP joke?

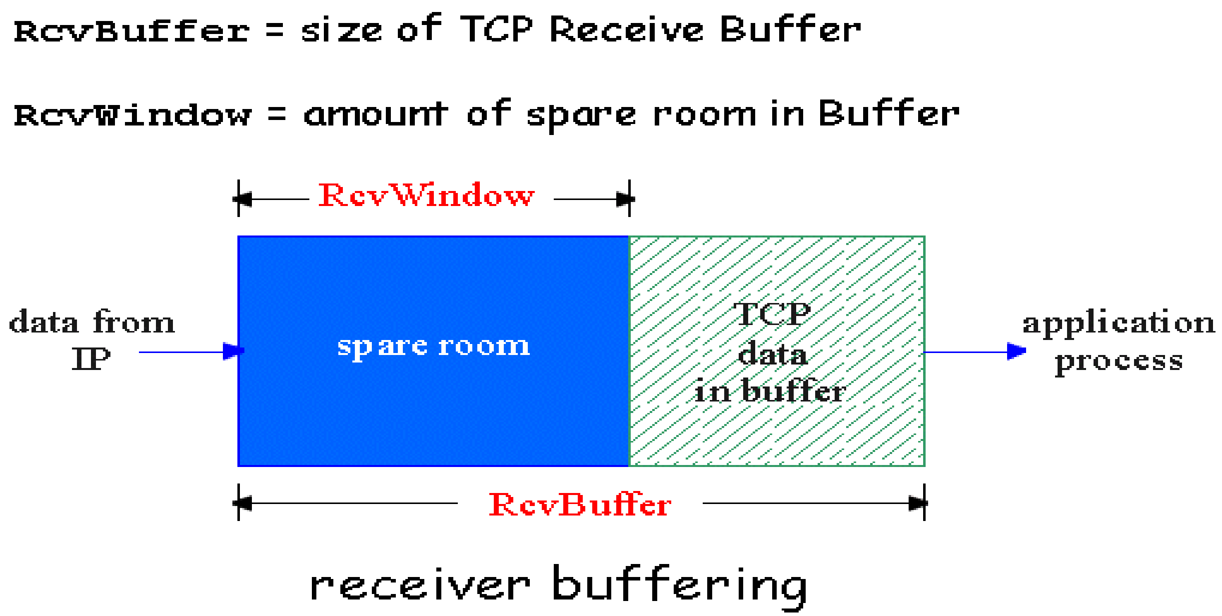
UDP:

- Wanna hear a TCP joke?
- Yes!

- Was it good?
- What?!

Flow Control = Window

- **The sender should not "drown" the recipient by sending too much, too fast.**
- Receiver informs sender about free buffer capacity.
  - *RcvWindow* in TCP segment
- The sender takes this into consideration



Traffic jam / Congestion

- Too many sources send too much data too fast for the network (routers) to handle.
  - This is different from flow control which depends on the capacity of the end systems and is controlled by the exchange of window sizes (RWIN).
- Results in..
  - lost packets (overflowing router buffer)
  - long delays (queue in router buffer)
- This can be, and often is, a big problem!

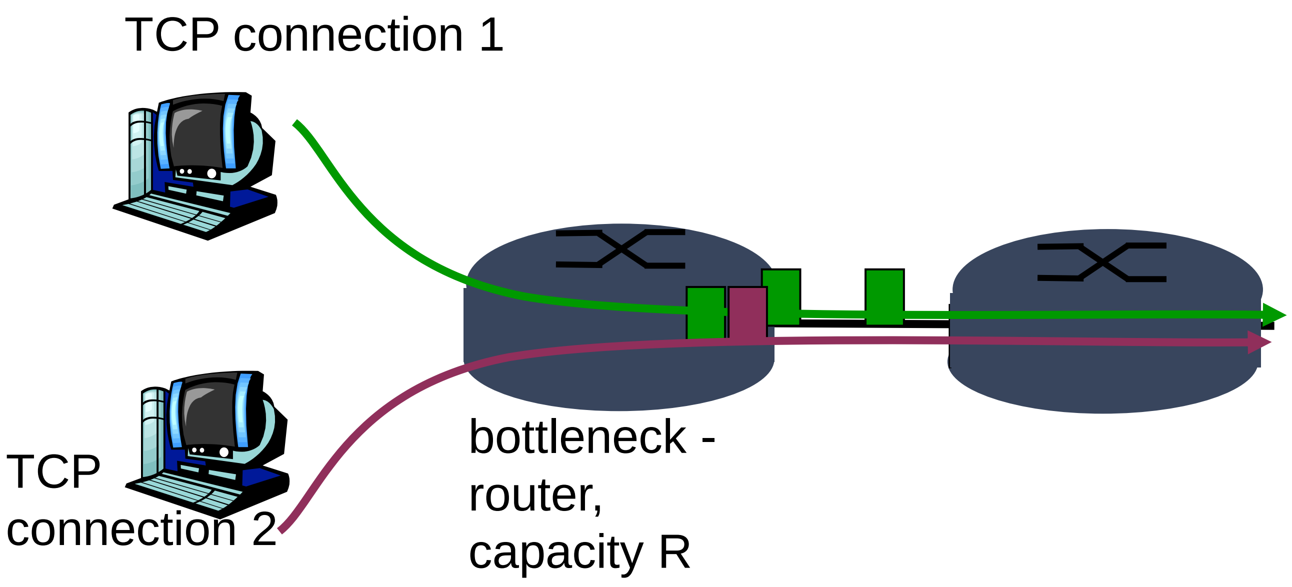
Congestion Control Principles

Høyskolen Kristiania

- End-to-end control
  - No feedback from the network
  - Endpoint even finds out if there are problems
  - Symptoms: Delay, packet loss
  - TCP uses this principle.
- Network-assisted control
  - The routers provide feedback to the ends
  - Puts bit flag in packet header (SNA, ATM ......)
  - Direct feedback (choke packet)

TCP fairness

Objective: if *K* TCP sessions share the same bottleneck link with data rate *R*, each of them should have an average data rate of *R* / *K*



TCP Fairness

Høyskolen Kristiania

Fairness and UDP

Multimedia applications "rarely" use TCP

- does not want data rate limited due to traffic jam control.

- Instead uses UDP:
  - pumps audio / video at a constant rate, tolerates packet loss

Fairness and parallel TCP connections

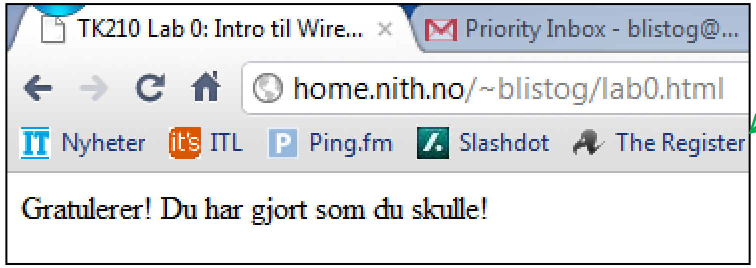
- nothing prevents the appl from opening parallel connections between two machines.
- Browsers often do this
- Example: link with rate R with 9 connections:
- new appl asks for one TCP connection, gets rate R / 10
- new appl asks for 11 TCP connections, gets rate R / 2!

End

Høyskolen Kristiania

Wireshark: Easy transfer

Høyskolen Kristiania



1) Experiment

2) Result

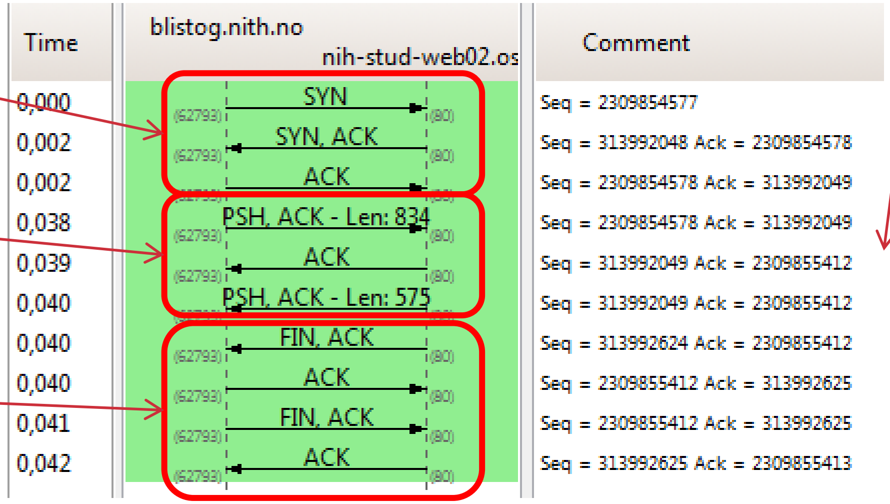
	Time	Source	Destination	Protocol	Info
5	3.012370	blistog.nith.no	nih-stud-web02.osl.	TCP	62793 > http [SYN] Seq=2309854577 Win=8192 Len=0 MSS=1460 WS=2 SACK_PERM=1
6	3.014275	nih-stud-web02.osl.	blistog.nith.no	TCP	http > 62793 [SYN, ACK] Seq=313992048 Ack=2309854578 Win=5840 Len=0 MSS=1380
7	3.014329	blistog.nith.no	nih-stud-web02.osl.	TCP	62793 > http [ACK] Seq=2309854578 Ack=313992049 Win=16560 Len=0
8	3.050158	blistog.nith.no	nih-stud-web02.osl.	HTTP	GET /~blistog/lab0.html HTTP/1.1
9	3.051843	nih-stud-web02.osl.	blistog.nith.no	TCP	http > 62793 [ACK] Seq=313992049 Ack=2309855412 Win=59 Len=0
10	3.052514	nih-stud-web02.osl.	blistog.nith.no	HTTP	HTTP/1.1 200 OK (text/html)
11	3.052515	nih-stud-web02.osl.	blistog.nith.no	TCP	http > 62793 [FIN, ACK] Seq=313992624 Ack=2309855412 Win=59 Len=0
12	3.052571	blistog.nith.no	nih-stud-web02.osl.	TCP	62793 > http [ACK] Seq=2309855412 Ack=313992625 Win=16416 Len=0
13	3.052912	blistog.nith.no	nih-stud-web02.osl.	TCP	62793 > http [FIN, ACK] Seq=2309855412 Ack=313992625 Win=16416 Len=0
14	3.054161	nih-stud-web02.osl.	blistog.nith.no	TCP	http > 62793 [ACK] Seq=313992625 Ack=2309855413 Win=59 Len=0

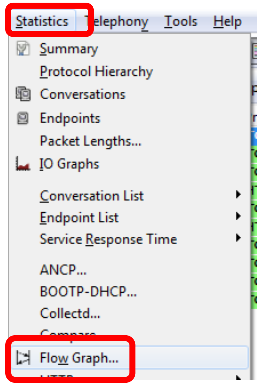
3) Flow Graph

Connection

Data transfer
GET + ACK + 200OK

Disconnection





Wireshark: SYN and SYN + ACK

Client

```
Source port: 62793 (62793)
Destination port: http (80)
[Stream index: 0]
Sequence number: 2309854577
Header length: 32 bytes
Flags: 0x02 (SYN)
  000. .... .... = Reserved: Not set
  ...0 .... .... = Nonce: Not set
  .... 0... .... = Congestion Window Reduced (CWR)
  .... .0.. .... = ECN-Echo: Not set
  .... ..0. .... = Urgent: Not set
  .... ...0 .... = Acknowledgement: Not set
  .... .... 0... = Push: Not set
  .... .... .0.. = Reset: Not set
  .... .... ..1. = Syn: Set
  .... .... ...0 = Fin: Not set
Window size: 8192
Checksum: 0x76be [correct]
Options: (12 bytes)
  Maximum segment size: 1460 bytes
  NOP
  Window scale: 2 (multiply by 4)
  NOP
  NOP
  TCP SACK Permitted Option: True
```

SYN

Server

```
Source port: http (80)
Destination port: 62793 (62793)
[Stream index: 0]
Sequence number: 313992048
Acknowledgement number: 2309854578
Header length: 28 bytes
Flags: 0x12 (SYN, ACK)
  000. .... .... = Reserved: Not set
  ...0 .... .... = Nonce: Not set
  .... 0... .... = Congestion Window Reduced (CWR)
  .... .0.. .... = ECN-Echo: Not set
  .... ..0. .... = Urgent: Not set
  .... ...1 .... = Acknowledgement: Set
  .... .... 0... = Push: Not set
  .... .... .0.. = Reset: Not set
  .... .... ..1. = Syn: Set
  .... .... ...0 = Fin: Not set
Window size: 5840
Checksum: 0x5f08 [correct]
Options: (8 bytes)
  Maximum segment size: 1380 bytes
  NOP
  Window scale: 7 (multiply by 128)
```

SYN + ACK

- Exchanges sequence number
- Agreements MSS
- Agreements / exchanges Window scaling

Exercises

Høyskolen Kristiania

- Text assignment set

- You MUST have completed practical exercises to 0x05 and 0x06!
- When we talk briefly about tools such as ping, netstat, etc., then you MUST test them in the practice lessons use text-based shell and test all tools we have been to, even if it is not specifically mentioned in the exercise text…
- Practical exercises
    - Install ncat
    - Is part of NMAP; https://nmap.org/download.html
    - Run online port scan against own machine
    - https://www.grc.com/default.htm
    - Scroll down and click on the ShieldsUp!
    - Click Proceed, and then click Common Ports
    - (Should not give any findings, and this is not the Information Security subject,
    - the purpose is to learn about port numbers…)
    - Wireshark now look at the transport layer
    - Wireshark go through the latest tasks with HTTP, FTP etc; but look at the transport layer

Høyskolen Kristiania

For optional self-study

For those who want to learn some topics in more depth to better understand, here are some extra topics related to today's teaching, it must be expected some personal work to understand these topics.

There will be no questions on the exam from these, and this is therefore not considered to be part of the syllabus.

netstat

```
C:\>netstat -n

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    127.0.0.1:19872        127.0.0.1:55169        ESTABLISHED
  TCP    127.0.0.1:27015        127.0.0.1:55155        ESTABLISHED
  TCP    127.0.0.1:52965        127.0.0.1:52967        ESTABLISHED
  TCP    127.0.0.1:52967        127.0.0.1:52965        ESTABLISHED
  TCP    127.0.0.1:55155        127.0.0.1:27015        ESTABLISHED
  TCP    127.0.0.1:55169        127.0.0.1:19872        ESTABLISHED
  TCP
  TCP
  TCP
  TCP
  TCP
  TCP
  TCP
  TCP
  TCP
  TCP
  TCP    158.36.131.51:61306    174.129.195.86:443     CLOSE_WAIT
  TCP    158.36.131.51:62305    74.125.79.118:443      ESTABLISHED
  TCP    158.36.131.51:62434    158.36.191.141:443     ESTABLISHED
```

Klient IP-adresse: portnummer

Server IP-adresse: portnummer

Forbindelsens Tilstand

Windows

`netstat` is the command that provides an overview of open ports:

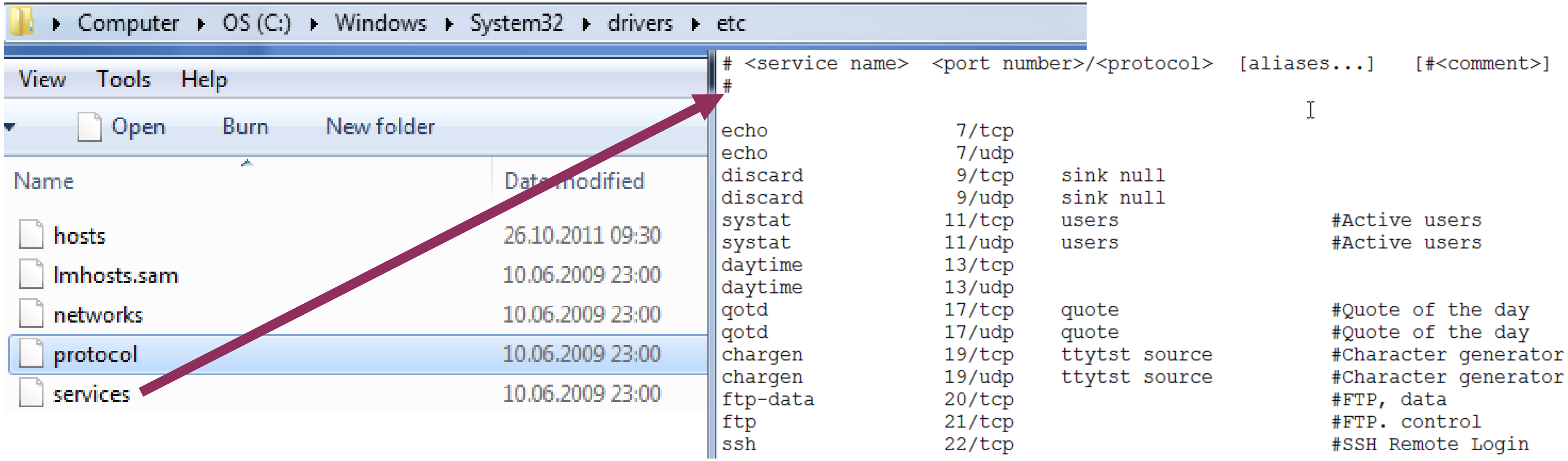
- `netstat -a` : UDP
- `netstat -s` : statistics
- `netstat -r` : routing table
- `netstat -n` : not DNS-name

+ many more to see which process o.l. (–B on Win7 etc)

```
~->netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address               Foreign Address             State
tcp        0      0 localhost.localdomain:smtp  localhost.localdomain:36098 TIME_WAIT
tcp        0      0 nih-stud-web02.osl.ba:44573 nih-mysql01.osl.basef:mysql TIME_WAIT
tcp        0      0 nih-stud-web02.osl.ba:44572 nih-mysql01.osl.basef:mysql TIME_WAIT
tcp        0      0 nih-stud-web02.osl.base:ssh nith-vpn-nat03.osl.ba:29459 ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node Path
unix  14     [ ]         DGRAM                    9016   /dev/log
unix  2      [ ]         DGRAM                    1855   @/org/kernel/udev/udevd
```

OSX

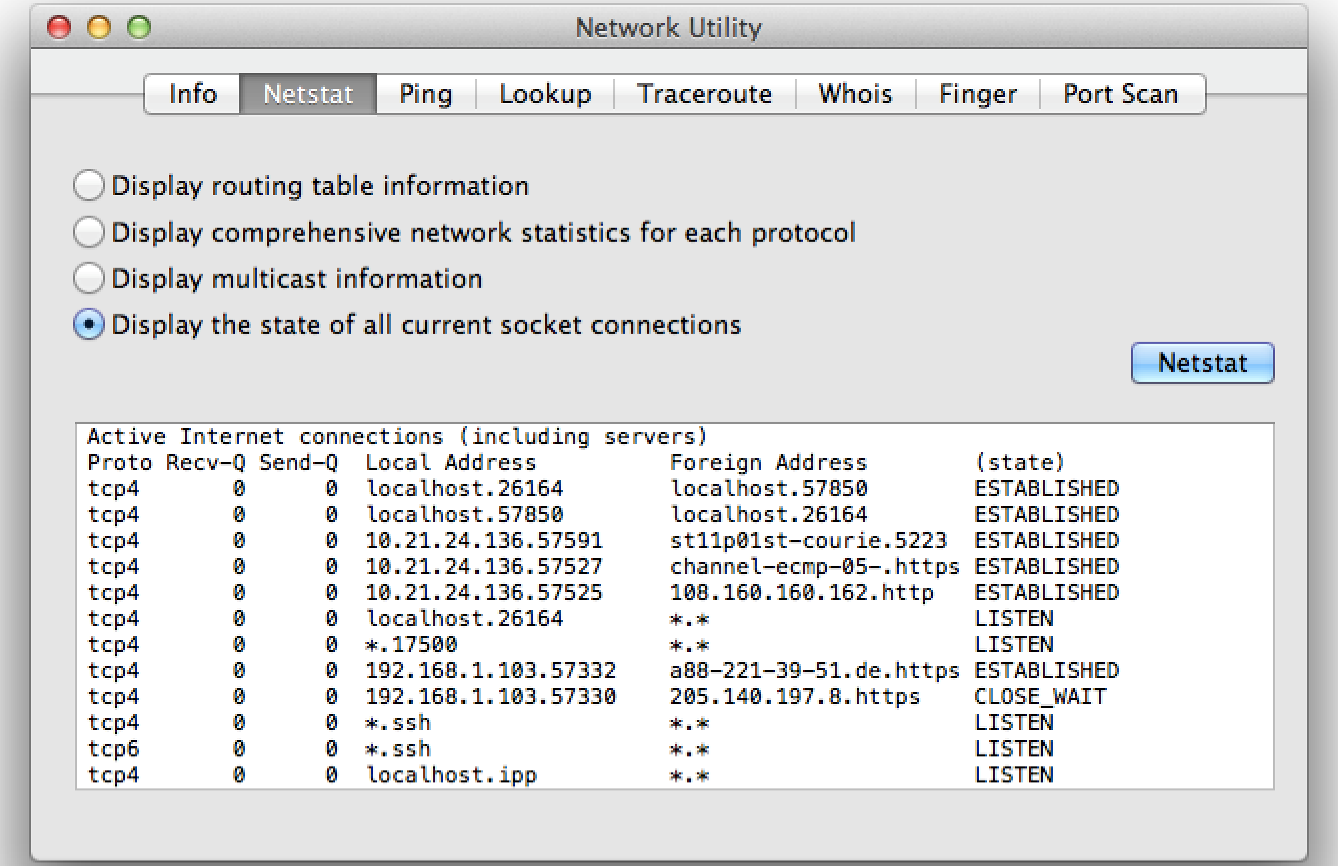
Overview: Services list



- Both in Windows (see above) and in OSX (/ etc) there is a list (services) with which ports are registered with which protocols / services
- This one decides what is stated as the protocol name of the netstat

OSX: Network Utility

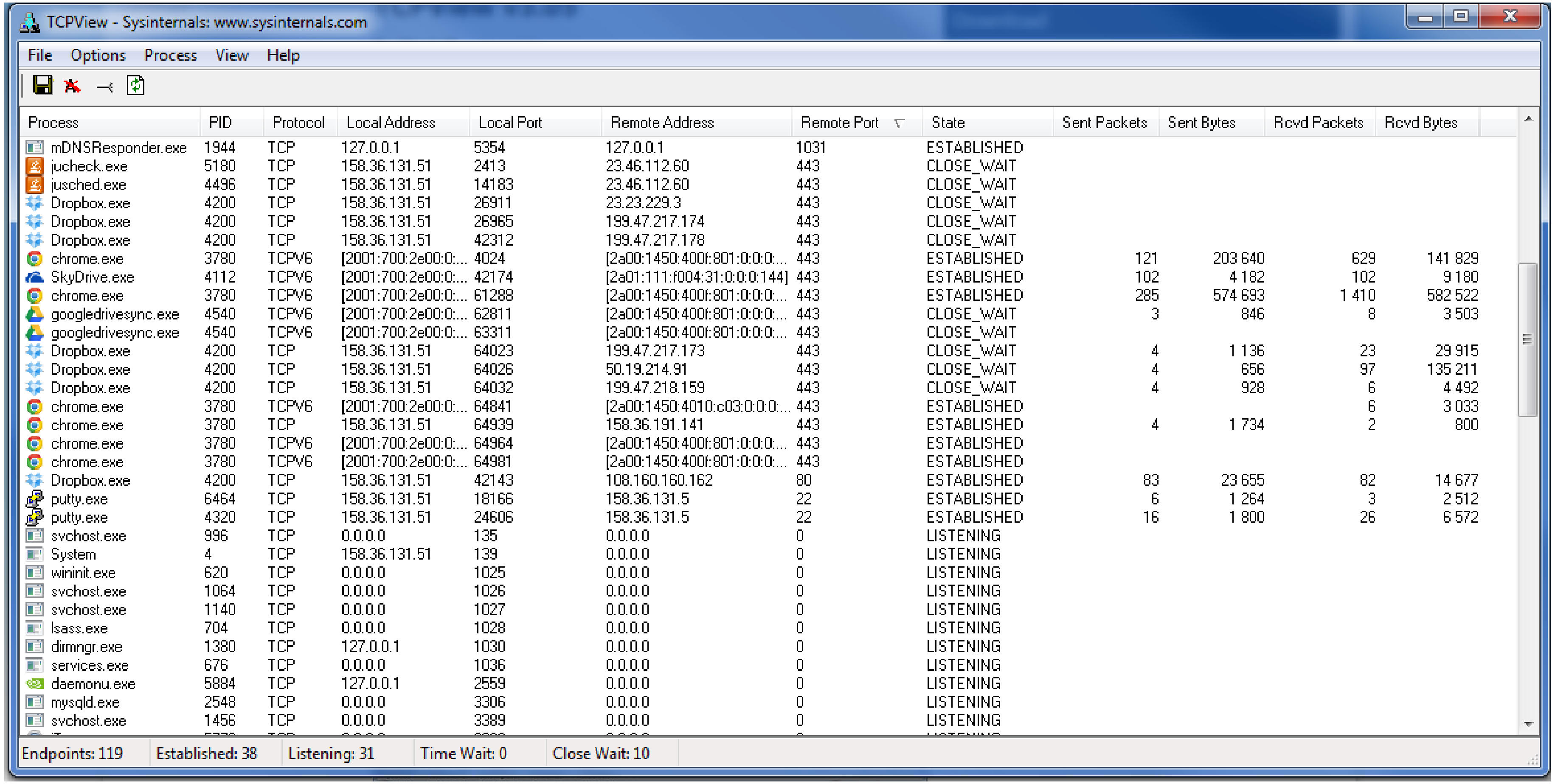
- OSX (under Applications / Utilities) has a GUI against the standard tools



Win: TCPView

Høyskolen Kristiania

- From MS you can download TCPView
- GUI that shows "most" things you can find with netstat



Process	PID	Protocol	Local Address	Local Port	Remote Address	Remote Port	State	Sent Packets	Sent Bytes	Rcvd Packets	Rcvd Bytes
mDNSResponder.exe	1944	TCP	127.0.0.1	5354	127.0.0.1	1031	ESTABLISHED				
jucheck.exe	5180	TCP	158.36.131.51	2413	23.46.112.60	443	CLOSE_WAIT				
jusched.exe	4496	TCP	158.36.131.51	14183	23.46.112.60	443	CLOSE_WAIT				
Dropbox.exe	4200	TCP	158.36.131.51	26911	23.23.229.3	443	CLOSE_WAIT				
Dropbox.exe	4200	TCP	158.36.131.51	26965	199.47.217.174	443	CLOSE_WAIT				
Dropbox.exe	4200	TCP	158.36.131.51	42312	199.47.217.178	443	CLOSE_WAIT				
chrome.exe	3780	TCPV6	[2001:700:2e00:0:...	4024	[2a00:1450:400f:801:0:0:0:...	443	ESTABLISHED	121	203 640	629	141 829
SkyDrive.exe	4112	TCPV6	[2001:700:2e00:0:...	42174	[2a01:111:f004:31:0:0:0:144]	443	ESTABLISHED	102	4 182	102	9 180
chrome.exe	3780	TCPV6	[2001:700:2e00:0:...	61288	[2a00:1450:400f:801:0:0:0:...	443	ESTABLISHED	285	574 693	1 410	582 522
googledrivesync.exe	4540	TCPV6	[2001:700:2e00:0:...	62811	[2a00:1450:400f:801:0:0:0:...	443	CLOSE_WAIT	3	846	8	3 503
googledrivesync.exe	4540	TCPV6	[2001:700:2e00:0:...	63311	[2a00:1450:400f:801:0:0:0:...	443	CLOSE_WAIT				
Dropbox.exe	4200	TCP	158.36.131.51	64023	199.47.217.173	443	CLOSE_WAIT	4	1 136	23	29 915
Dropbox.exe	4200	TCP	158.36.131.51	64026	50.19.214.91	443	CLOSE_WAIT	4	656	97	135 211
Dropbox.exe	4200	TCP	158.36.131.51	64032	199.47.218.159	443	CLOSE_WAIT	4	928	6	4 492
chrome.exe	3780	TCPV6	[2001:700:2e00:0:...	64841	[2a00:1450:4010:c03:0:0:0:...	443	ESTABLISHED			6	3 033
chrome.exe	3780	TCP	158.36.131.51	64939	158.36.191.141	443	ESTABLISHED	4	1 734	2	800
chrome.exe	3780	TCPV6	[2001:700:2e00:0:...	64964	[2a00:1450:400f:801:0:0:0:...	443	ESTABLISHED				
chrome.exe	3780	TCPV6	[2001:700:2e00:0:...	64981	[2a00:1450:400f:801:0:0:0:...	443	ESTABLISHED				
Dropbox.exe	4200	TCP	158.36.131.51	42143	108.160.160.162	80	ESTABLISHED	83	23 655	82	14 677
putty.exe	6464	TCP	158.36.131.51	18166	158.36.131.5	22	ESTABLISHED	6	1 264	3	2 512
putty.exe	4320	TCP	158.36.131.51	24606	158.36.131.5	22	ESTABLISHED	16	1 800	26	6 572
svchost.exe	996	TCP	0.0.0.0	135	0.0.0.0	0	LISTENING				
System	4	TCP	158.36.131.51	139	0.0.0.0	0	LISTENING				
wininit.exe	620	TCP	0.0.0.0	1025	0.0.0.0	0	LISTENING				
svchost.exe	1064	TCP	0.0.0.0	1026	0.0.0.0	0	LISTENING				
svchost.exe	1140	TCP	0.0.0.0	1027	0.0.0.0	0	LISTENING				
lsass.exe	704	TCP	0.0.0.0	1028	0.0.0.0	0	LISTENING				
dirmngr.exe	1380	TCP	127.0.0.1	1030	0.0.0.0	0	LISTENING				
services.exe	676	TCP	0.0.0.0	1036	0.0.0.0	0	LISTENING				
daemonu.exe	5884	TCP	127.0.0.1	2559	0.0.0.0	0	LISTENING				
mysqld.exe	2548	TCP	0.0.0.0	3306	0.0.0.0	0	LISTENING				
svchost.exe	1456	TCP	0.0.0.0	3389	0.0.0.0	0	LISTENING				

Endpoints: 119 Established: 38 Listening: 31 Time Wait: 0 Close Wait: 10

Gates and security (firewalls)

- Access to a running application is via a port number (cf. experiences with HTTP in exercises)
- Port numbers can thus tell us quite a bit about which programs are running on a machine

- Common port scanning tools are: **nmap**
- Port scanning will be able to tell you (a lot) about the OS and what services it is running
- Most software firewalls are there to filter out such "unwanted" requests.

```
~->nmap 158.36.131.51

Starting Nmap 5.00 ( http://nmap.org ) at 2011-11-02 14:29 CET
Interesting ports on blistog.nith.no (158.36.131.51):
Not shown: 994 filtered ports
PORT     STATE SERVICE
80/tcp   open  http
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
443/tcp  open  https
445/tcp  open  microsoft-ds
3389/tcp open  ms-term-serv

Nmap done: 1 IP address (1 host up) scanned in 4.75 seconds
```

UDP checksum

- Sender
  - Perceives the segment as composed of 16 bits words
  - Summarizes all of the words
  - Takes 1’s complement of the sum (flips)
  - Puts checksum into the header of the segment.
- Receiver
  - Summarizes all 16-bit words in the received segment, including the check sum.
  - If sum= 1111 1111 1111 1111 => everything OK
- At best, this only gives an indication of whether an error has occurred during the transfer

Ex: Internet checksum

Note: Mean in the most significant position is added to the LSb (Least Significant Bit)!

Ex: Two 16 bit parts of the total package are added together

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

HOW TO ACHIEVE RELIABLE TRANSMISSION THROUGH AN UNRELIABLE CHANNEL?

Principles behind…

T ransmission

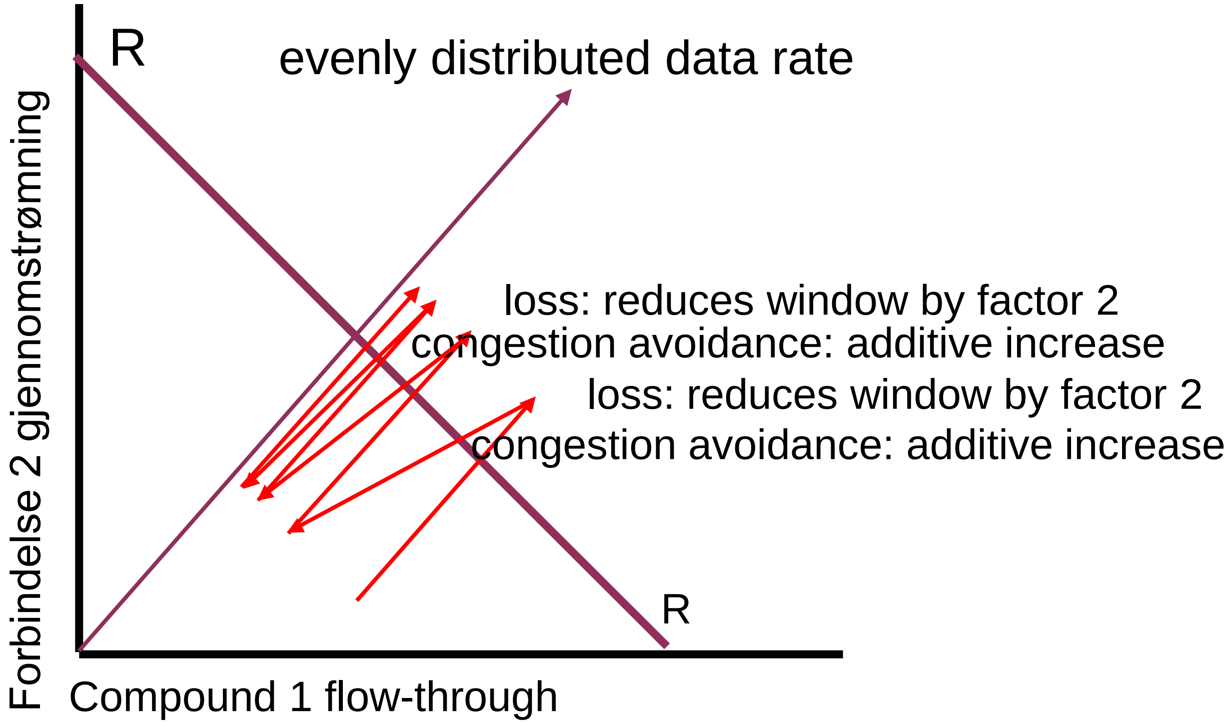
C ontrol

P rotocol

Why is TCP fair?

Two competing sessions :

- Additive increase gives a slope of 1, which increases gradually
- multiplicative reduction reduces flow proportionally



TCP: Versions

- The biggest difference between different varieties is exactly how they handle saturation
- Tahoe ("vanilla"), Reno, New Reno, Vegas, BIC / CUBIC (Linux 2.6->), CTCP (Windows Vista / 7 ->, ..)
- Everyone tries to get the bit rate back up faster after packet loss
- Without outperforming "vanilla" TCP
- New versions should be Tahoe-friendly!

TCP: Nagle's algorithm

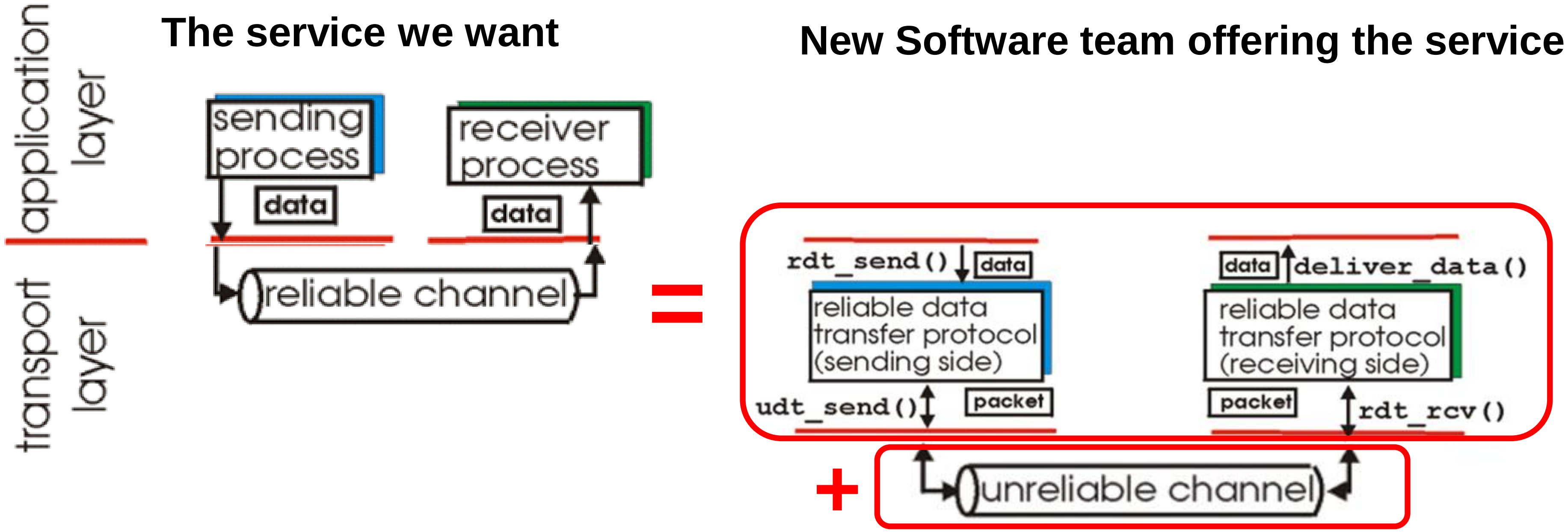
Høyskolen Kristiania

- TCP + IPv4 adds 20 bytes of headers each
- If we only have to transfer **one** letter, this means a large "overhead".
- Only 1/41 = 2.4% of the package is data
- With ACK from server: only 1.2% of the bandwidth is used for something useful (even less if we include the link layer header.
- Nagle's algorithm
  - Stores data going to the same server until ACK on the previous packet is received, or the amount of data becomes> = 1 MSS (maximum segment size)
  - Problematic in real-time applications (eg online games) due to "delayed ACK" from the server side
  - Can / must be solved in the programming of the application

Reliable transmission

- UDP sikrer mux/demux
- UDP gjør det bare mulig for mottager å oppdage at bit-feil i en pakke kan ha oppstått
  - In the event of an error, it is common for the OS to drop the package
  - No error handling etc.
- TCP must provide a reliable connection.
  - hen we need to take into account the various sources of error (noise / bit error, loss,…) and how to deal with them

Reliability is dealing with errors



- If the channel is reliable, we only need mux / demux
- What we want is to offer the service reliable transmission over an unreliable channel
- Then you will want to create a protocol and software (methods) that will take care of this.
- Cost: Greater complexity

Problem & Solution

- Problem:
    - Want reliable data transfer over networks made up of unreliable media.

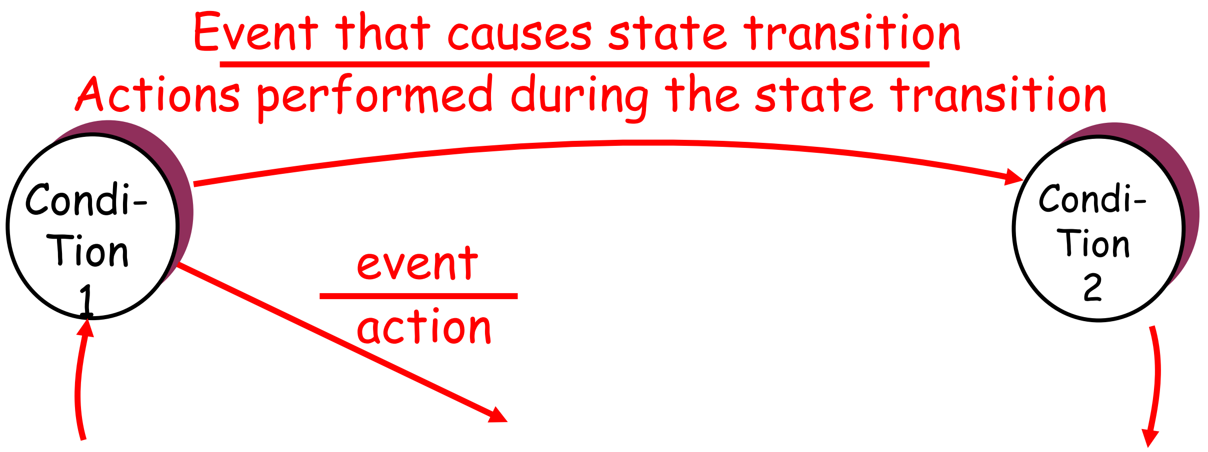
- Solution strategy:
    1. Starts with the ideal state (media perfect)
    2. Introduces the problems associated with real media (noise and loss) one by one
    3. Step by step constructs a protocol that handles the issues.

Different steps of reliability

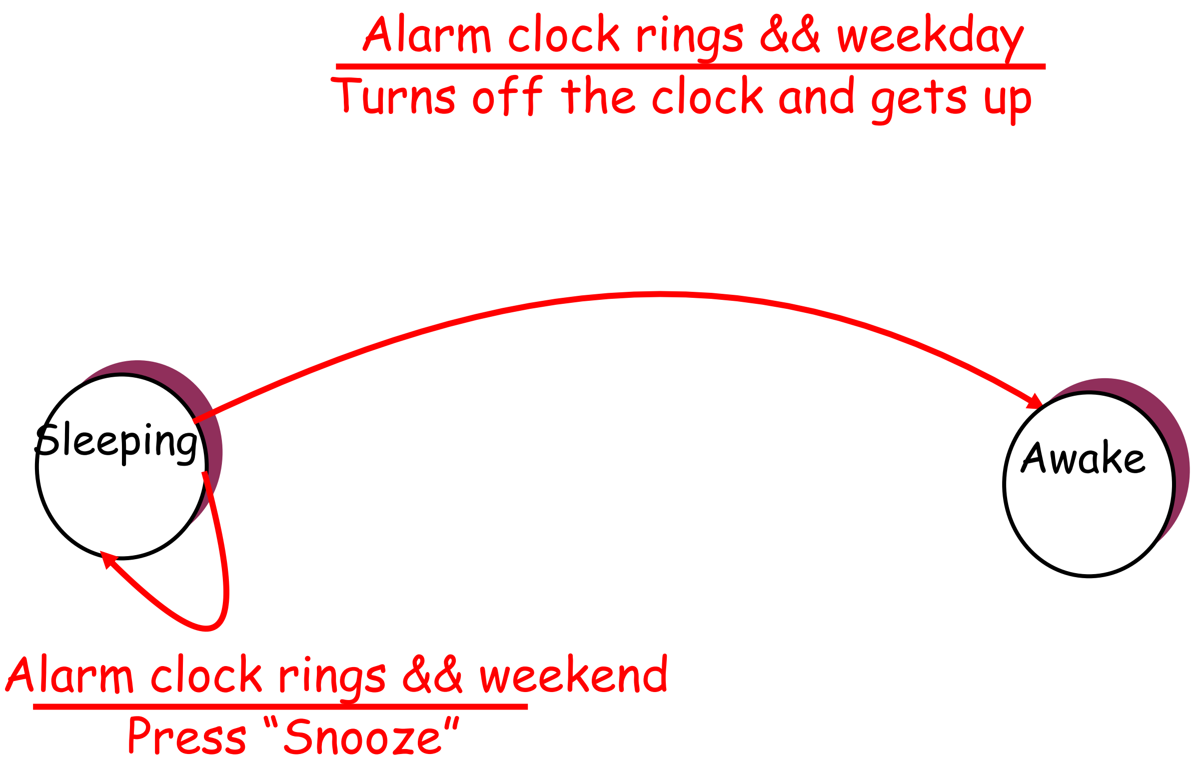
Høyskolen Kristiania

- We are now creating a "play protocol", which we call RDT (Reliable Data Transport).
- We will look at different levels of RDT and build these up gradually
- Discusses only data transport in one direction
  - Same as full duplex, but easier to explain
  - Control information goes in both directions
- Uses FSM (Finite State Machines) to specify the behavior of the sender and recipient

state: when in this "state" the next state is unambiguously determined by the next event

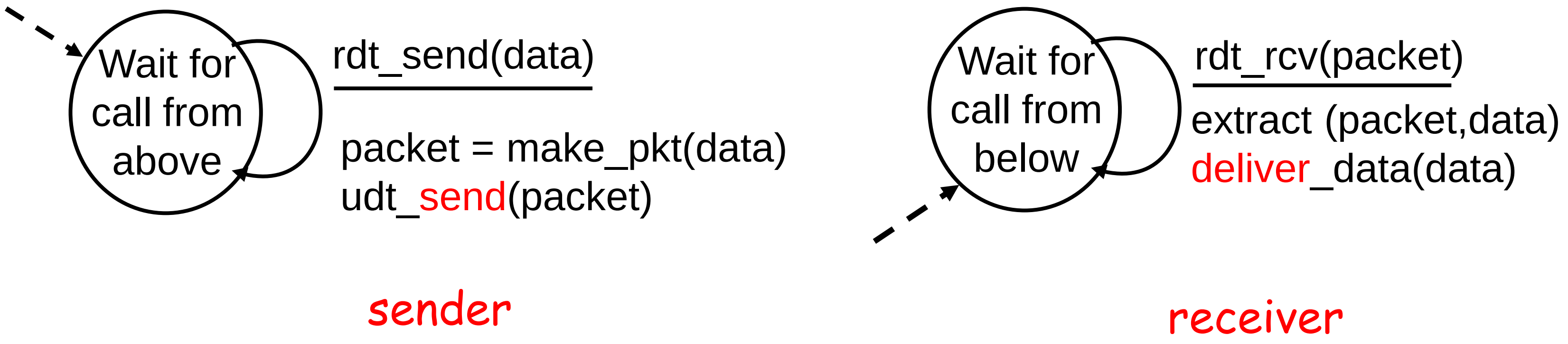


FSM: "Practical" Example



V. 1.0: Reliable transmission / channel

- The chanel completely reliable
  - No bit errors, no loss of packages
- Separate FSM for sender and receiver
  - Check not required



v. 2.0: Channel with bit error

Høyskolen Kristiania

A) Detect any errors:

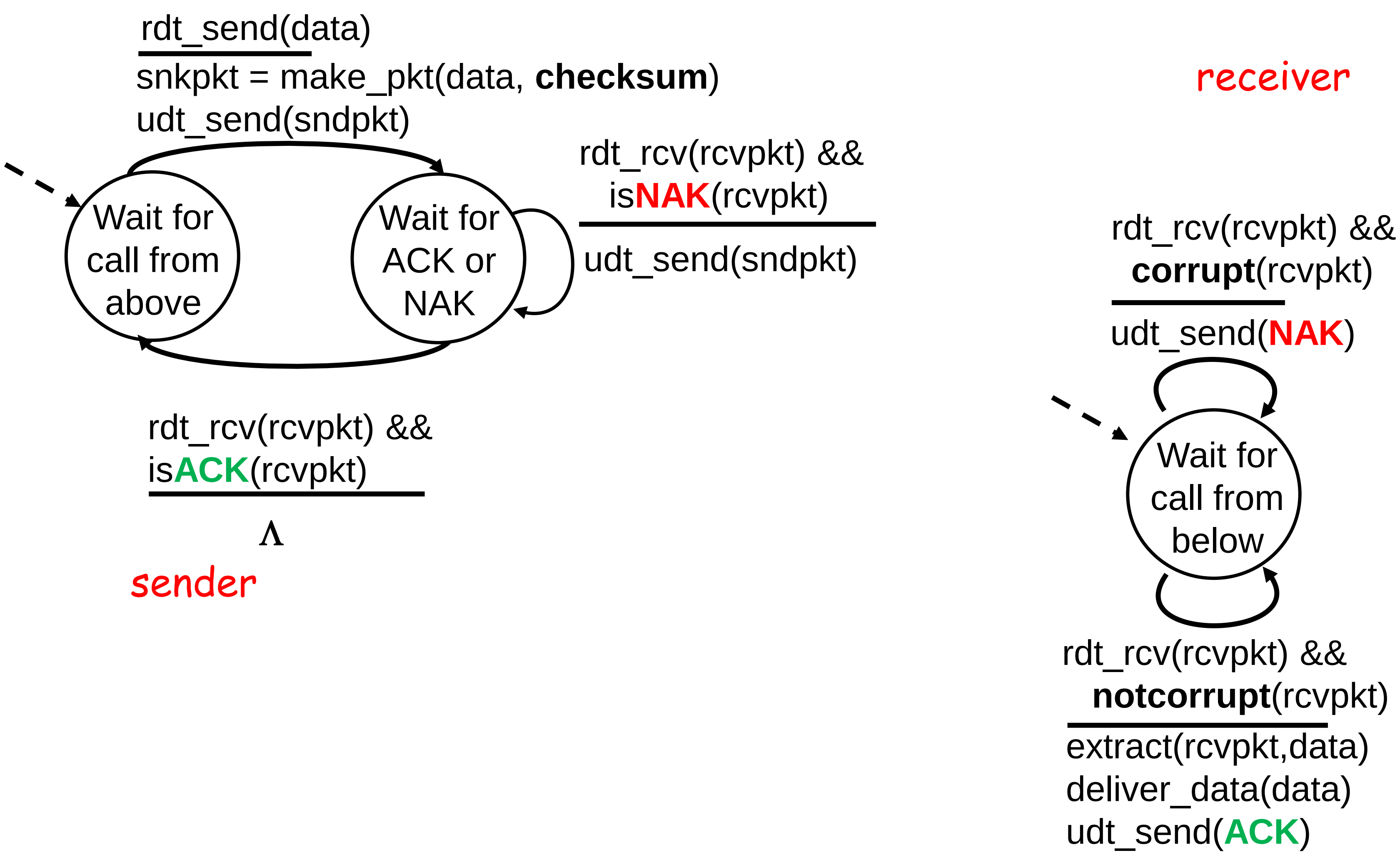
- Send checksum together with the data package.
  - Errors can be detected by the recipient, but are not corrected

B) Give feedback:

- Recipient sends notification of error to sender
  - **ACK** (acknowledge) sent when the package is OK.
  - **NAK** (negative ACK) sent when the package is faulty.
  - **The sender sends the package again at NAK.**

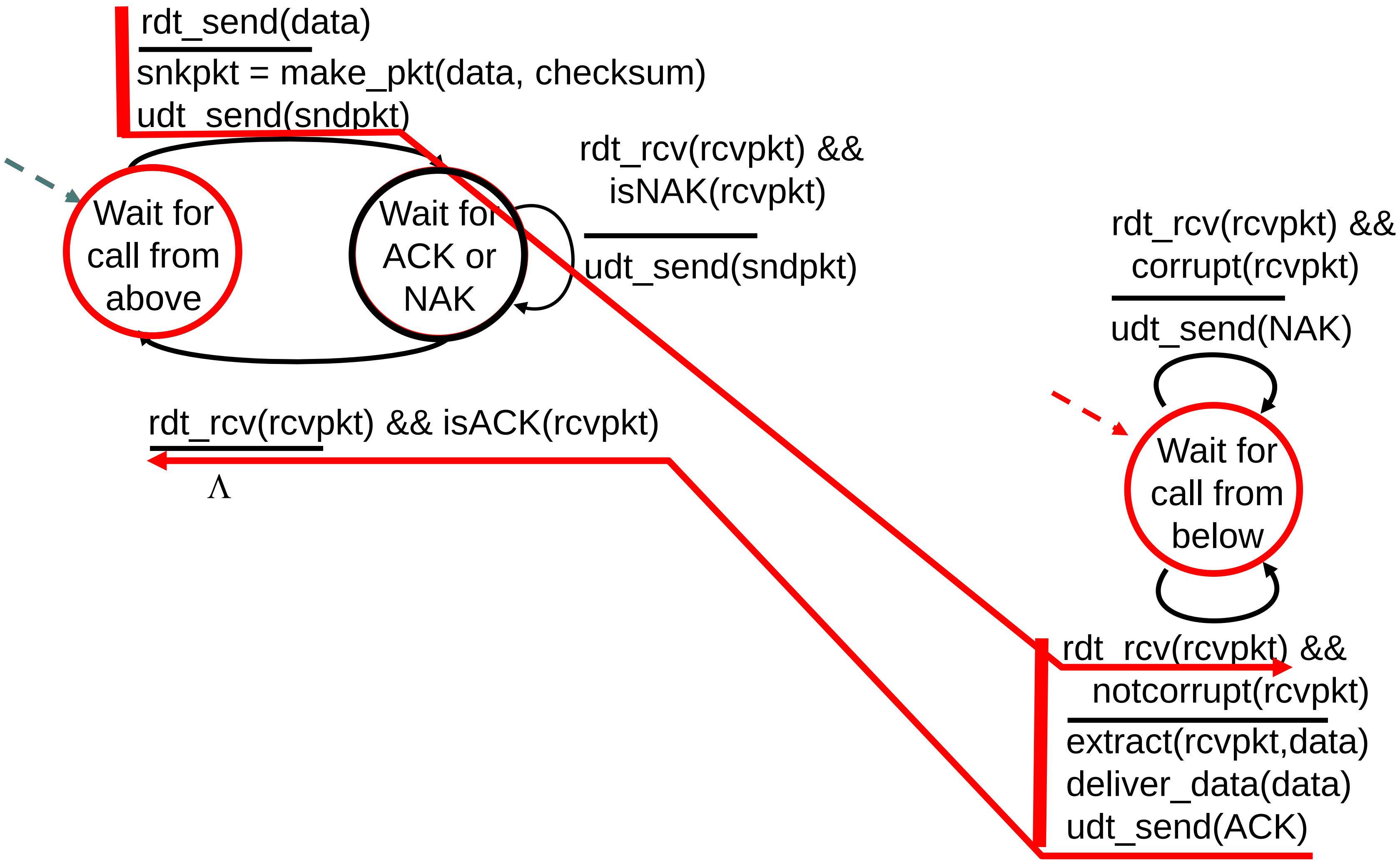
- Three new mechanisms:
  1. Error detection
  2. Control message (receipt) from recipient to sender
  3. Omatt transmission in case of error message

RDT 2.0: FSM model

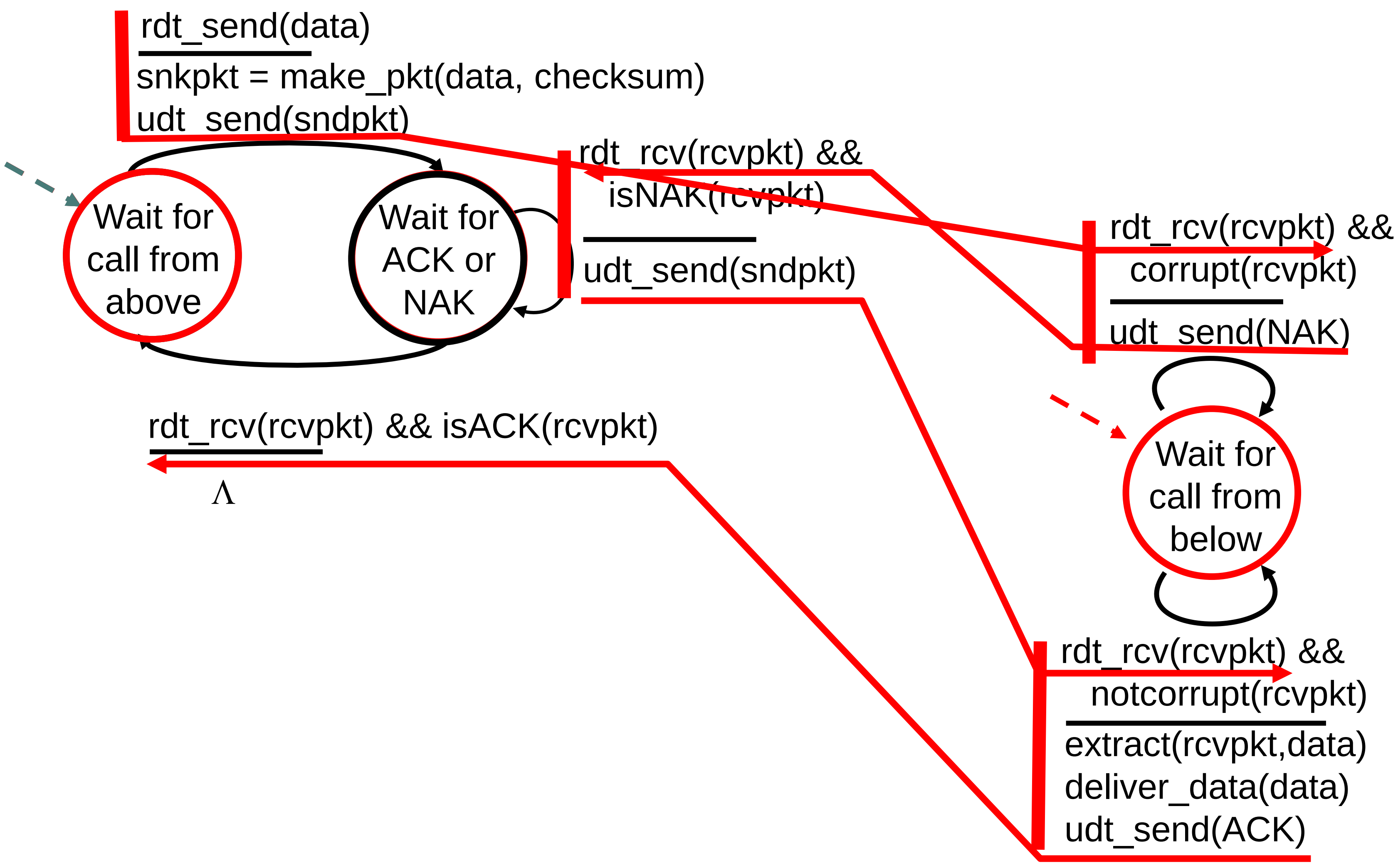


RDT 2.0:

FSM no transmission error



v. 2.0: FSM bit error in transmission ..



version 2.0: Fatal ERROR !!

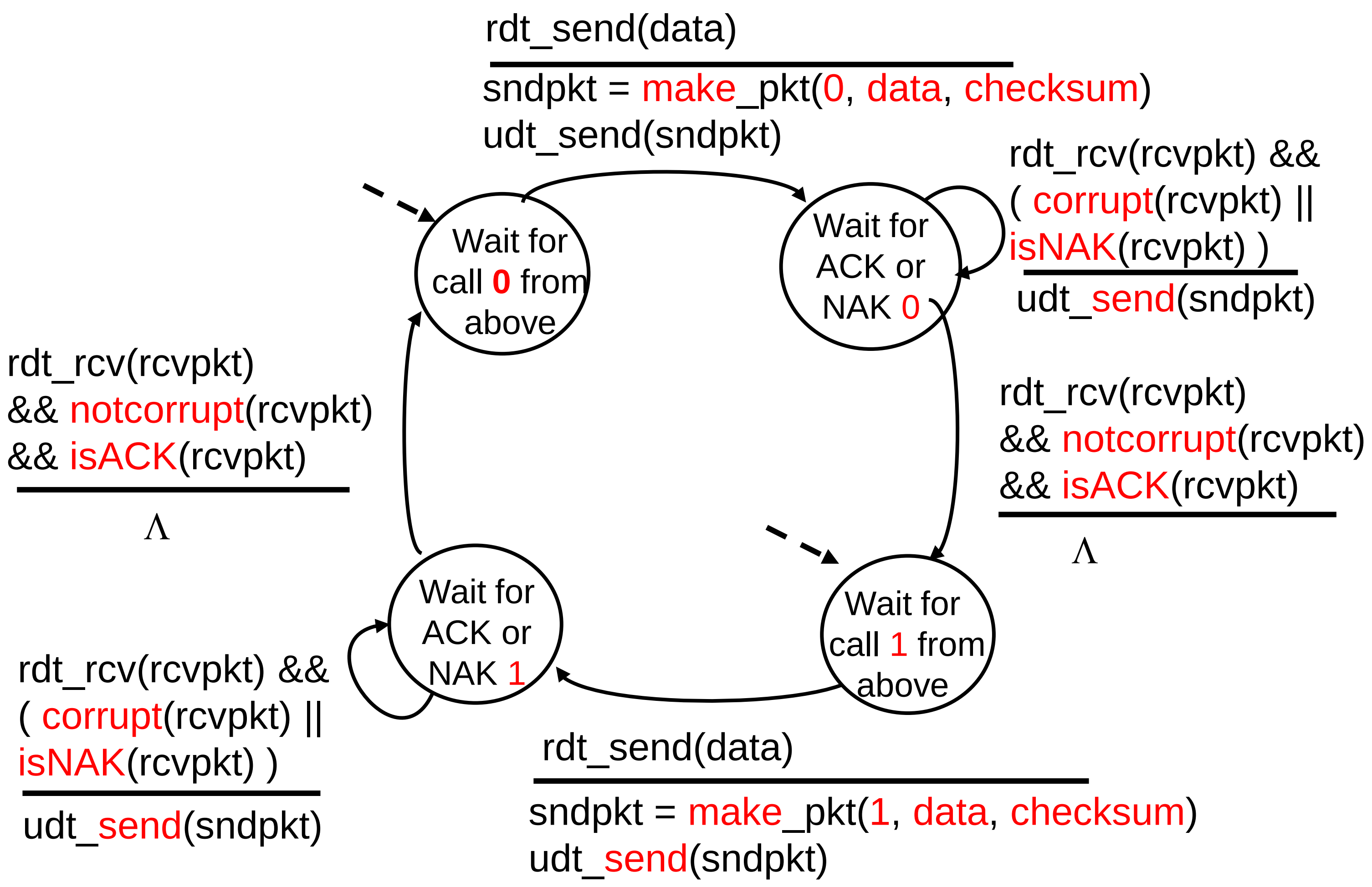
Høyskolen Kristiania

- The Problem occurs if the control message fails
  - The sender can then not know what happened to the package! (It certainly came out, but was it wrong or not?)
    - No purpose in re-sending a package that is OK.
    - Cannot re-send the package for fear of duplicate (Two identical packages will end up with incorrect data for the application)
- Solution
  - The sender sets the sequence number on the packet.
  - For a stop / wait protocol, only 1 bit sequence number is needed.
  - This means that we have to **double the number of conditions** for both sender and receiver

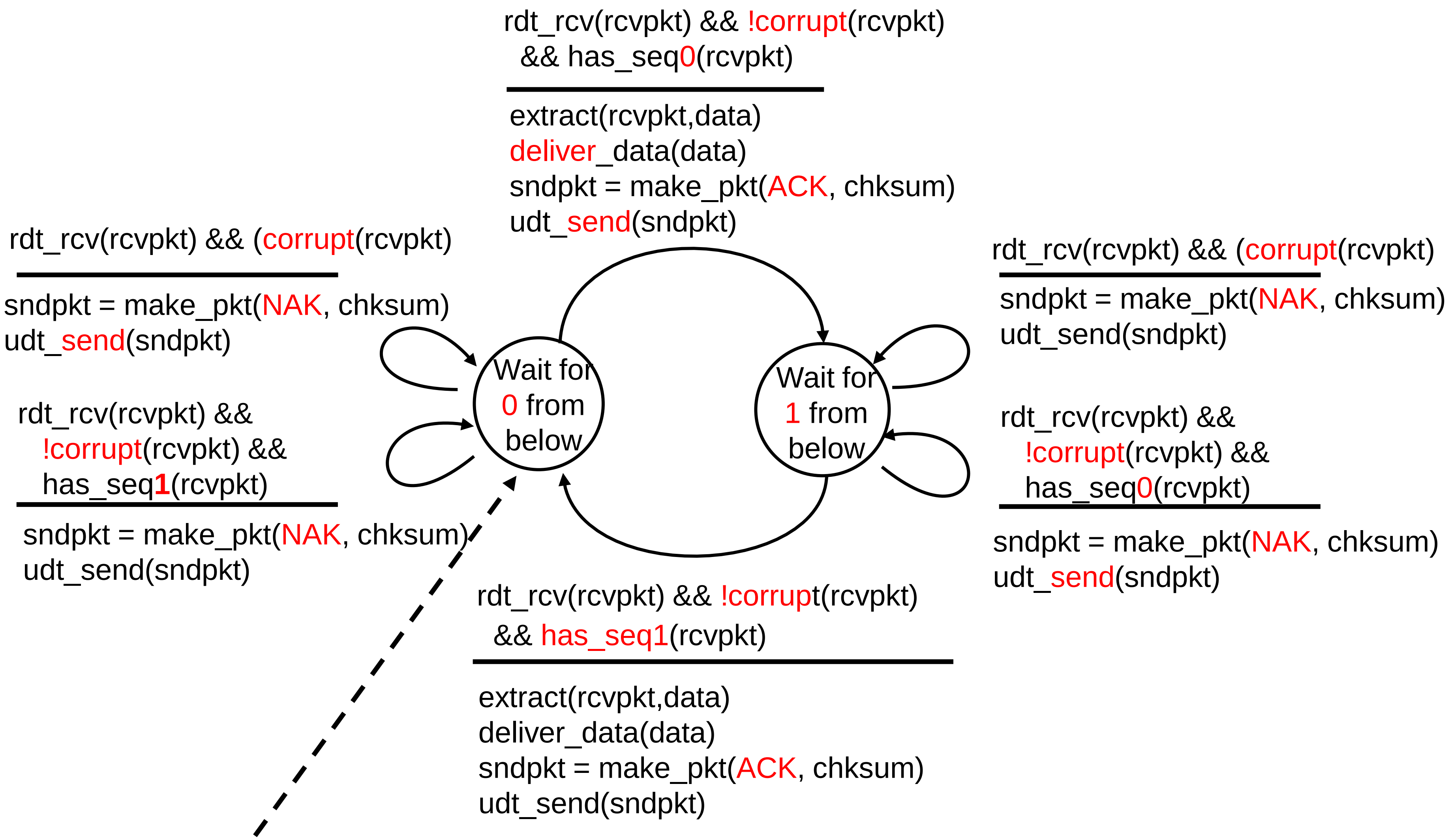
Version 2.1: FSM **sender**

Høyskolen Kristiania

- Can handle broken ACK / NAK



2.1: FSM receiver



2.1: Analysis

Sender:

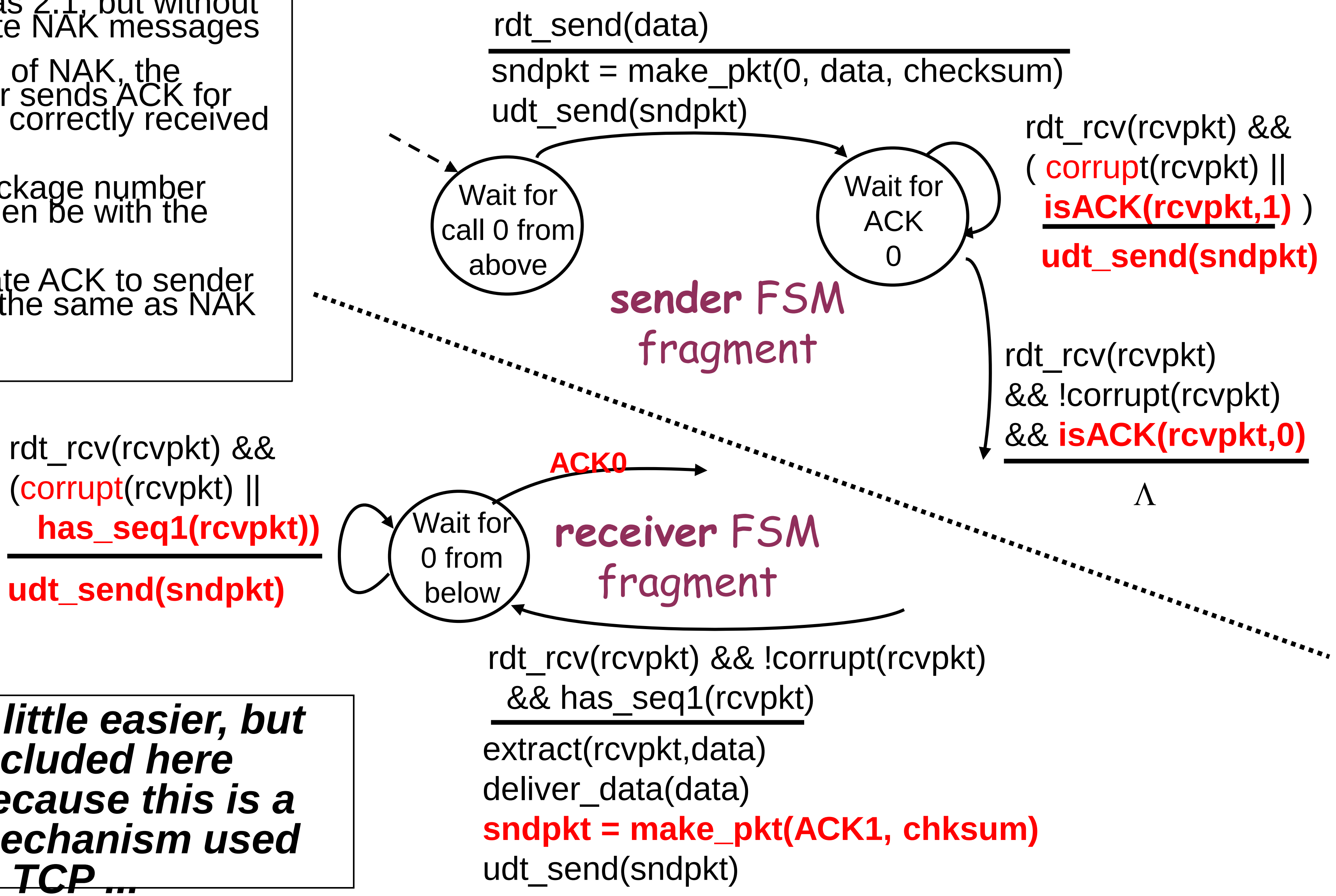
- Adding package number (0, 1)
- Must check if ACK / NAK package is corrupt
- Must remember number of last sent package
- This gives complexity; 2 states
- Must have two different states to "remember" the package number of the "current" package

Receiver:

- Must check if package is duplicate
- The state indicates whether 0 or 1 is the expected sequence number
- Note! The recipient cannot know if the last sent ACK / NAK has been received by the sender

version 2.2: NAK-free protocol

- Same as 2.1, but without separate NAK messages
- Instead of NAK, the receiver sends ACK for the last correctly received packet
- The package number must then be with the ACK
- Duplicate ACK to sender entails the same as NAK in 2.1

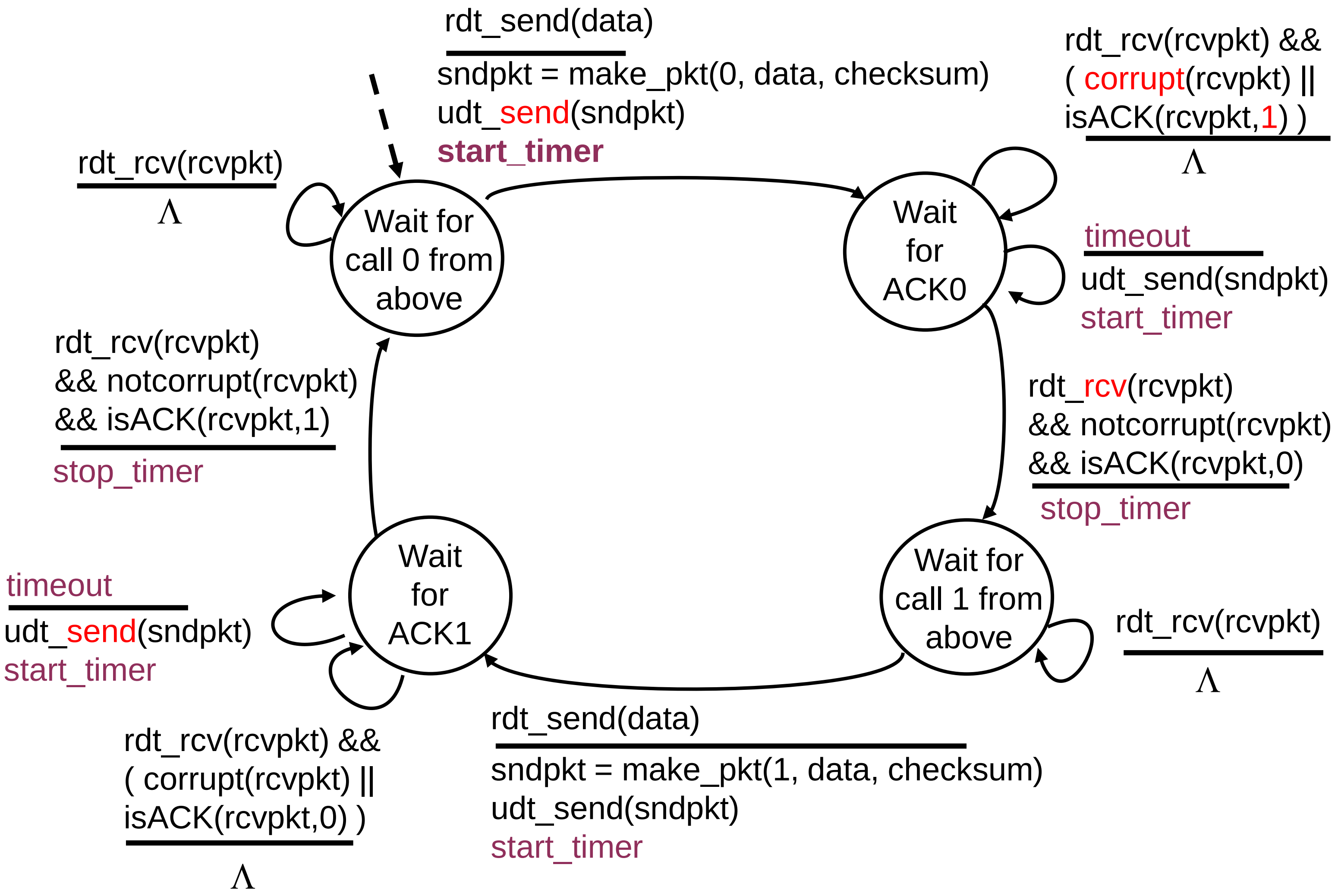


- ***A little easier, but included here because this is a mechanism used in TCP ...***

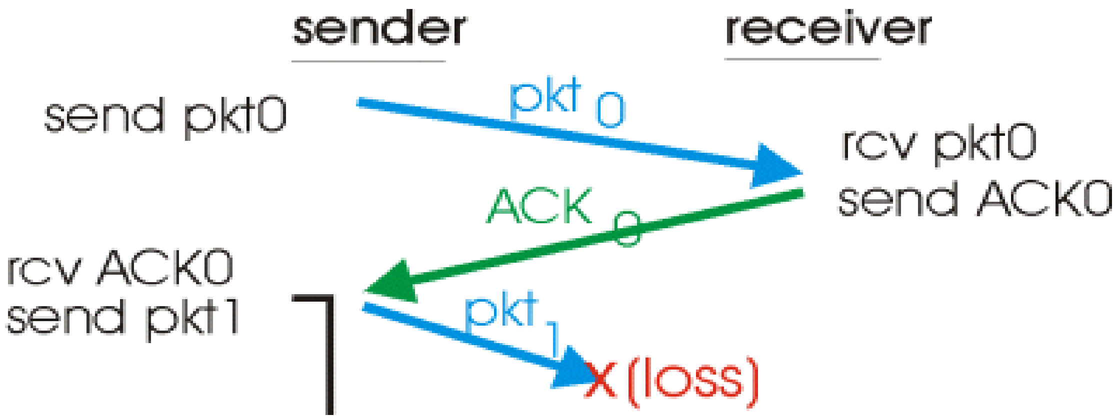
RDT 3.0: Channel with both errors and losses

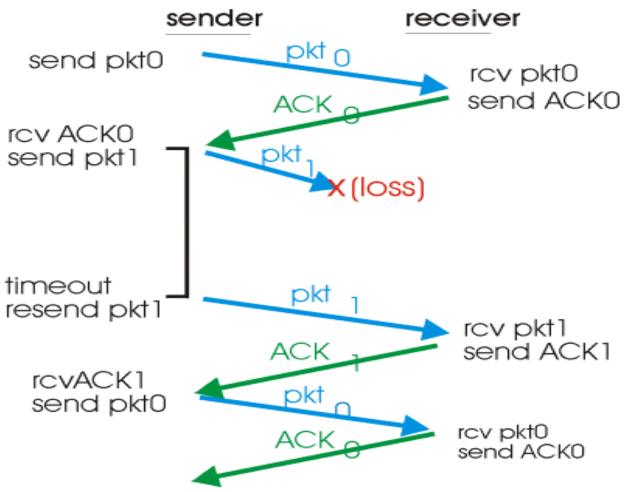
- The transport channel may also lose packages.
  - Both data and ACK packages
- Solution:
  Sender waiting for a "reasonable" period of time and resending package if no receipt appears.
- If packet or ACK is only delayed
  - Resending is then a duplicate, but the sequence number will solve this
  - The receiver must specify the sequence number on the ACK
- This solution requires a countdown timer at the sender;
- as a recipient, we can still use version 2.2

RDT 3.0: FSM sender

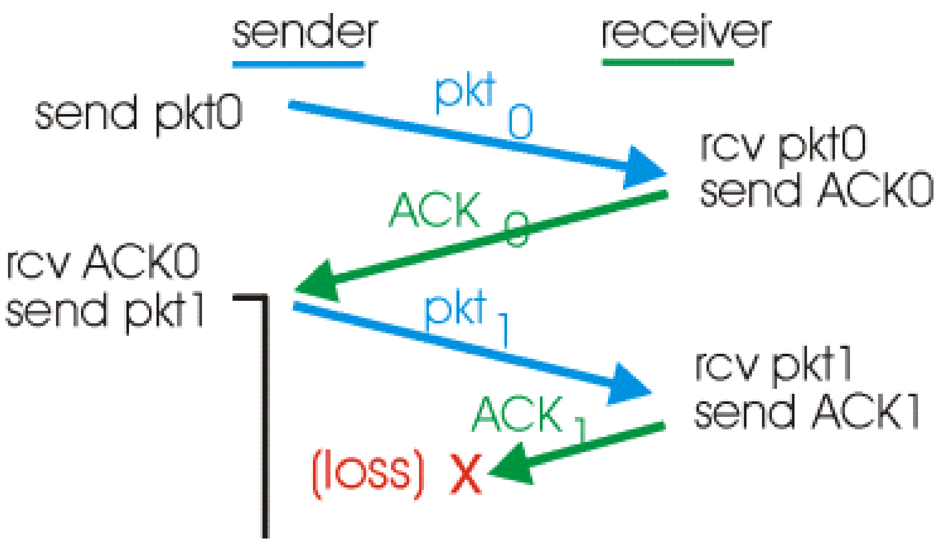


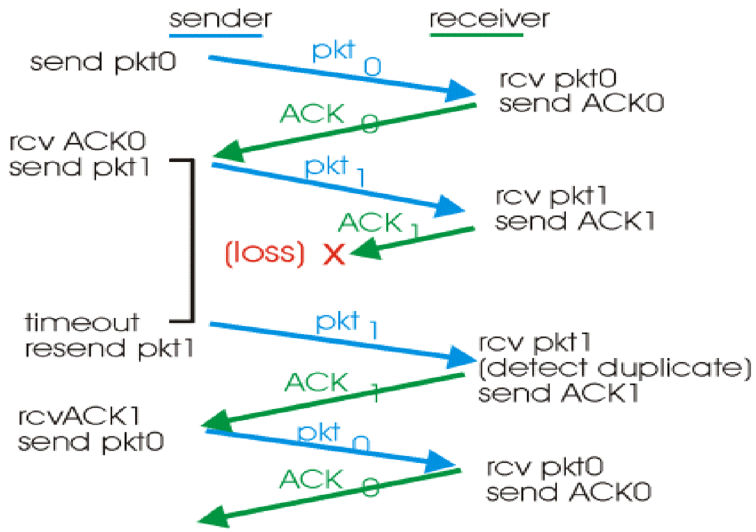
version 3.0: Lost package





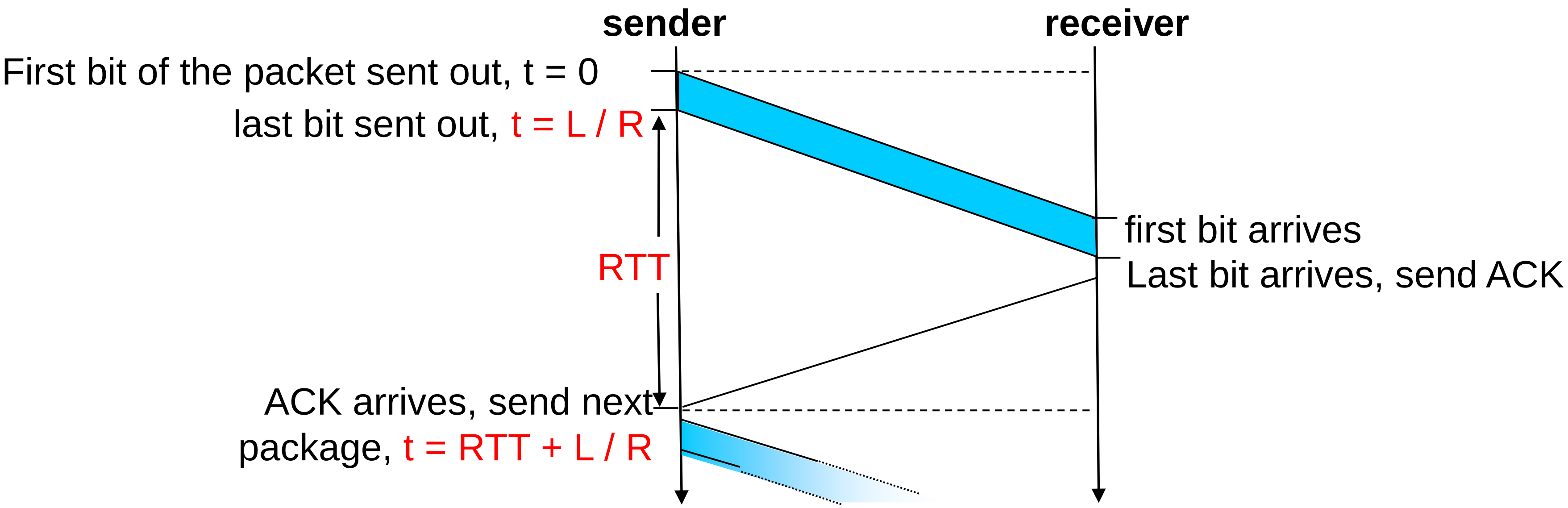
version 3.0: Lost ACK





version 3.0: Stop&Wait Performance

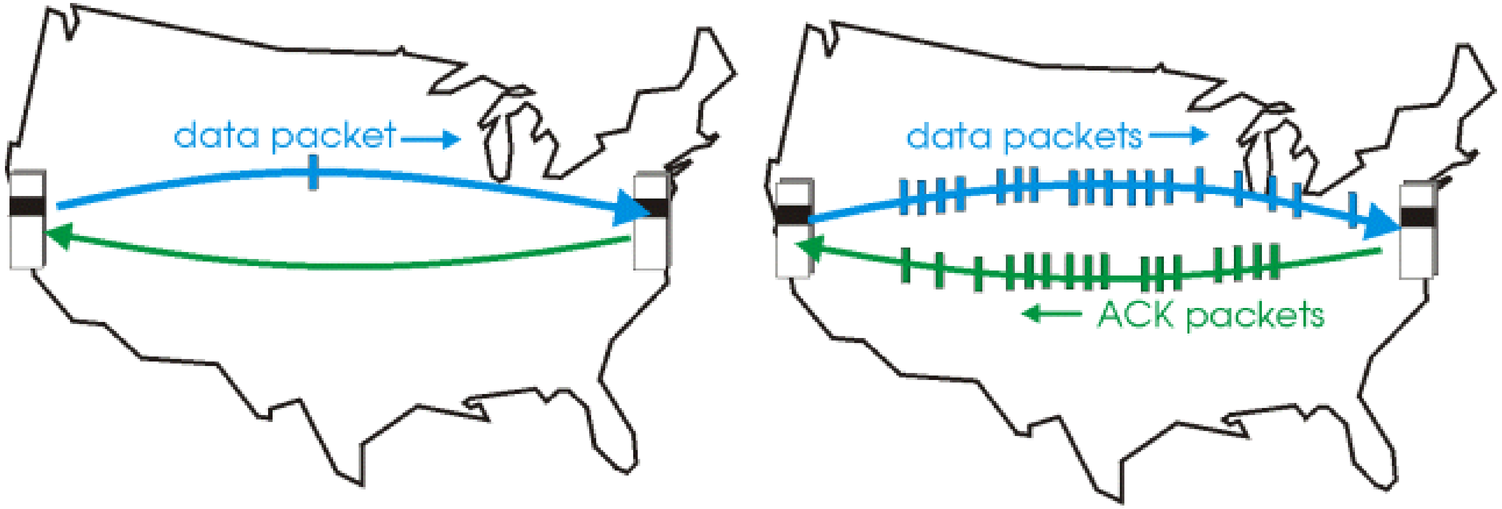
- version 3.0 works so far, but performance is *lusen*
- Example:
  - 1 KB package is shipped from Los Angeles to New York
  - Transmission time of 4500 km is 15 milliseconds
  - Bandwidth of 1 Gb / s takes 8 microseconds to get the entire packet out on the channel
  - Assumes the same length of time on the ACK
- Has then spent 30.016 milliseconds transferring a packet
- The channel utilization will be: 0.14 per mille (1/6667)



$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

Protocols with pipelining

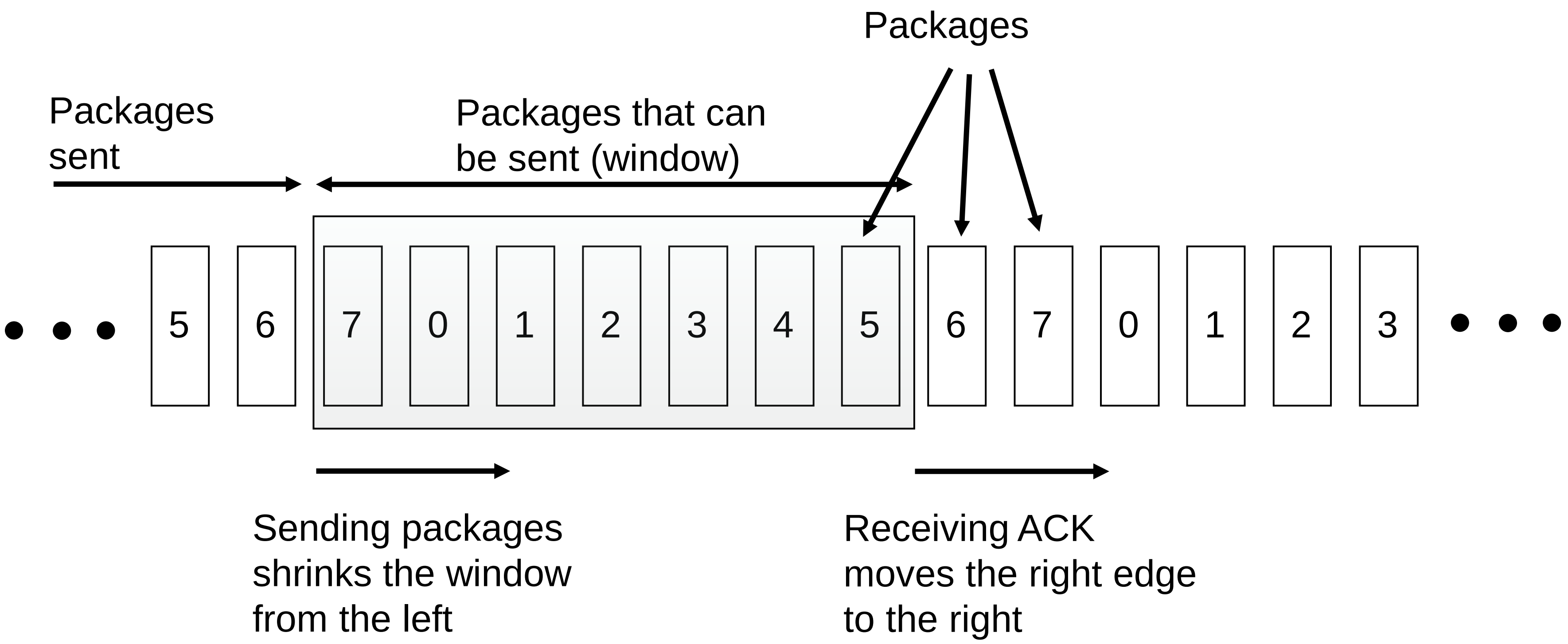
- Use of pipeline allows many packages "in the air" at the same time, ie «better bandwidth»
  - Sequence numbers must then be large enough
  - The sender / recipient must establish packet buffers



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

Sliding window

Example with three bit sequence number



SR (Selective Repeat) Principle

- Recipient gives ACK on each and every received package
  - Must have buffer to sort received packets
  - ACK also on unsolicited packages.

- Sender sends only about packages without received ACK
  - Must have timer for each package

Reliability: Summary

RDT	Situasjon	Problem	Løsning
1.0	Reliable channel	None	
2.0	Bit error on the line	Data becomes dull	•Error detection (checksum) •Receipt messages (ACK, NAK) •Resending in case of reported error
2.1	Bit error	•Error in receipt messages •Duplicate	•(Stop-Wait) •Sequence number of data packets (0.1)
2.2	Bitfeil	”Kompleksitet”	•Fjerner NAK, sender heller ACK med sekvensnummer for siste **korrekt** mottatte pakke
3.0	+ Loss of packages	Stop-Wait + packs away in the net	•Timer

- To achieve performance, we use pipeling
- This presupposes that we keep track of which packages are "in the air" with regard to receipts
- Requires buffers for unrecognized packets that may need to be resent
- TCP uses both window and selective retransmission…

T ransmission

C ontrol

P rotocol

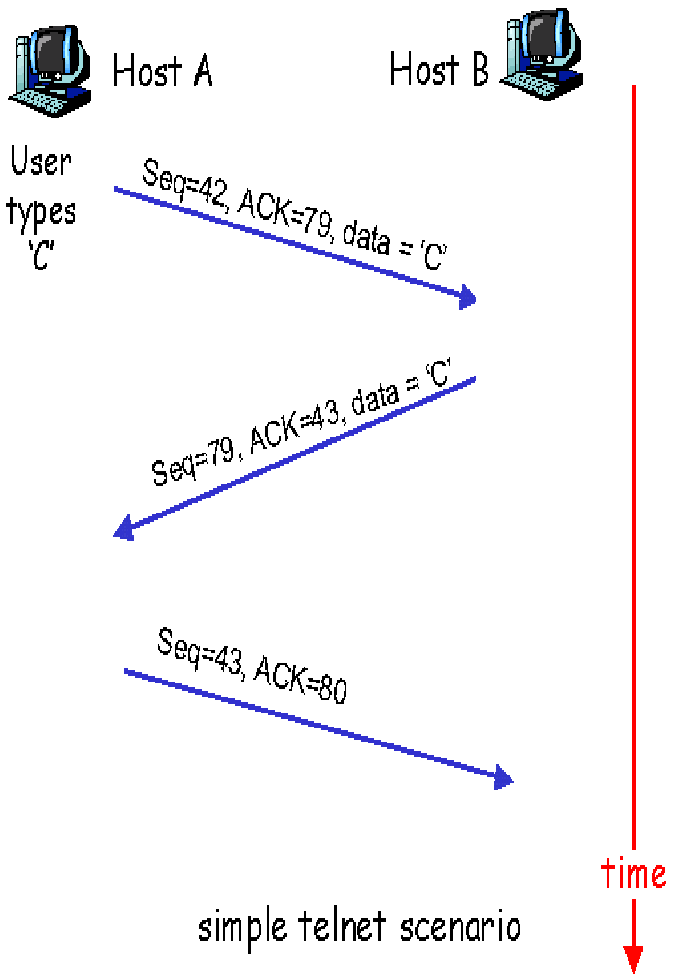
IN PRACTICE

Høyskolen Kristiania

Sequence number and ACK

Høyskolen Kristiania

- TCP is byte-oriented.
- Sequence number
  - Byte flow number for the first byte in the segment
- ACK-number
  - Sequence number of the next byte expected from the other side
  - Cumulative ACK
- Segments out of order
  - Not covered by the TCP specification, but must be handled in the implementation



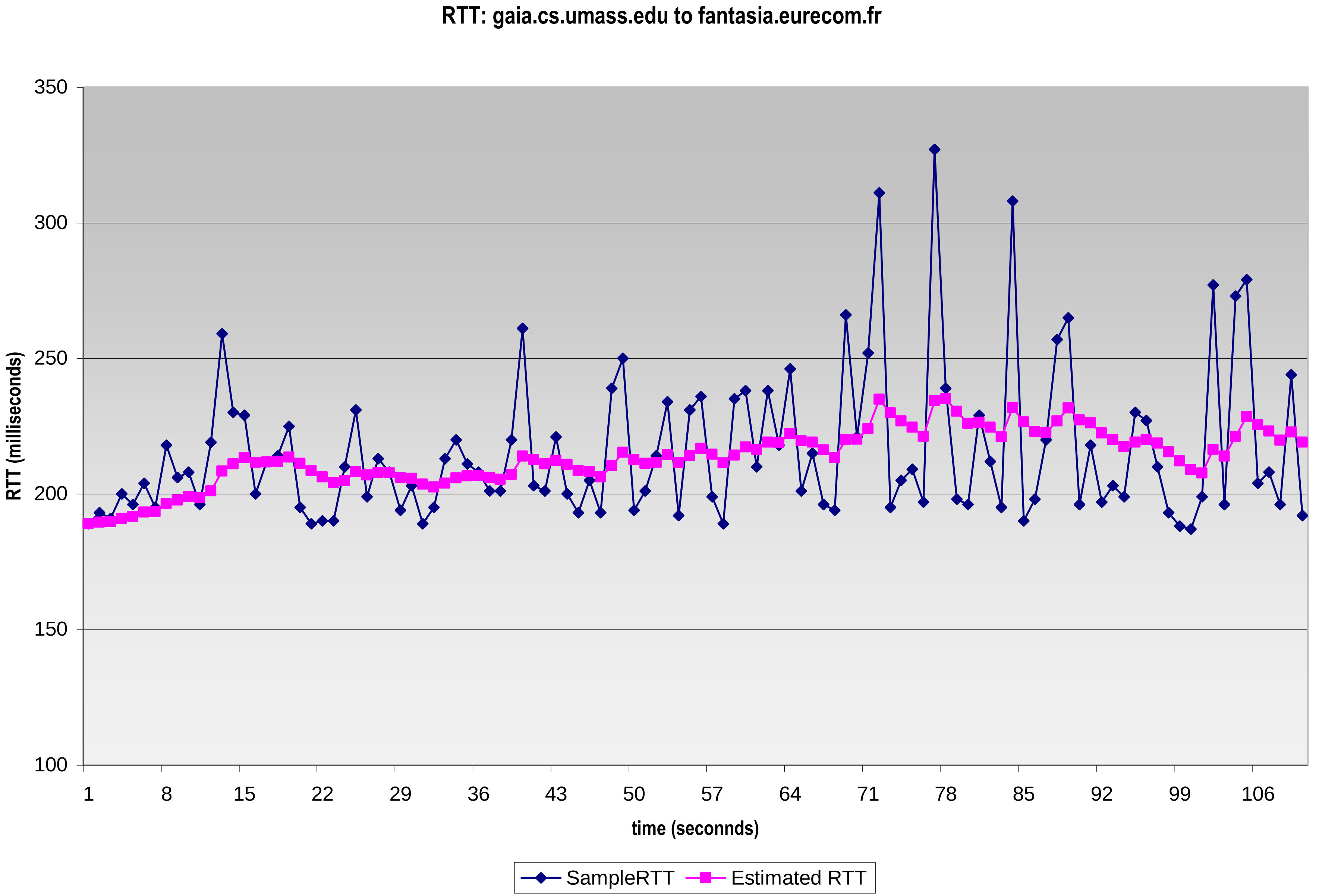
TCP ACK generation [RFC 1122, RFC 2581]

Event at Recipient	TCP Receiver Handling
Arrival of segment in the correct order with expected seq #. All data already ACKet	Delayed ACK. Wait up to 0.5 s on the next segment. If it does not arrive, send ACK. Send immediately cumulative ACK (works as ACK for both)
Arrival of segment in the correct order with expected seq #. A previous segment has arrived, but not ACKet	Send duplicate ACK immediately, which indicates seq # for the next expected byte
Arrival of segment out of order with higher seq # than expected (Gap detected)	Send immediate ACK (assuming the segment
Arrival of segments that partially or completely fill the gap	starts at the "bottom of the gap")

RTT (Round Trip Time) og timeout

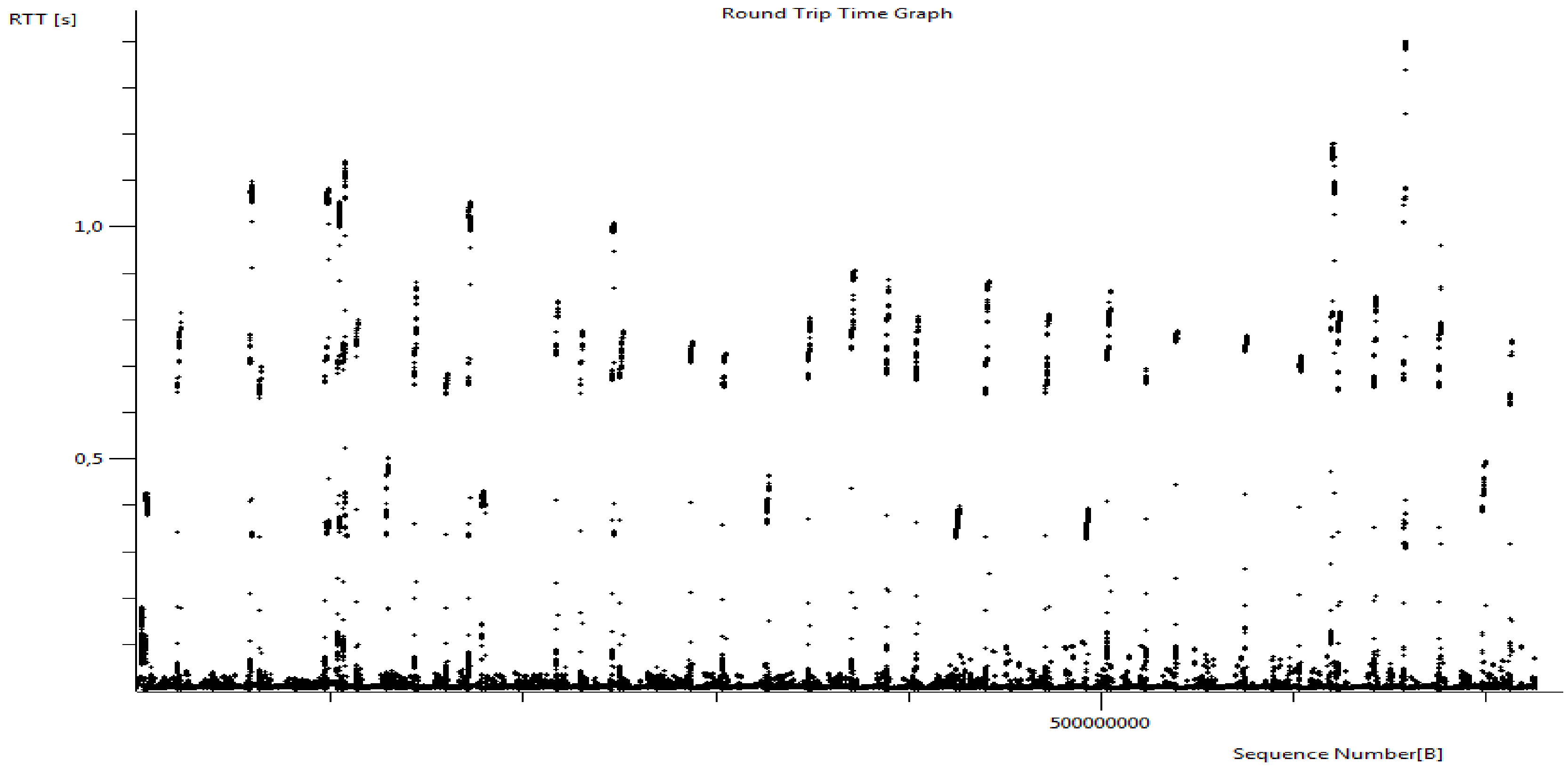
- How big should one choose the timeout value?
  - Longer than RTT
  - RTT varies!
  - Too short a timeout value results in unnecessary retransmission
  - Too long a timeout value gives too bad a reaction to segment loss
- Measures the time from sent segment to received ACK
  - Does not include transmit and cumulative ACK in the measurements
  - Must take into account that RTT varies
  - Most interested in the latest measurements

Example RTT estimation:



Wireshark: RTT

- Uploads 1 GB to home.nith.no
- RTT varies between well below 1/10 ms and 1.4 s (approx. Factor 10,000)



Calculation of timeout

Høyskolen Kristiania

- Calculation of equalized RTT.

```
EstimatedRTT = (1-x)*EstimatedRTT + x*SampleRTT
```

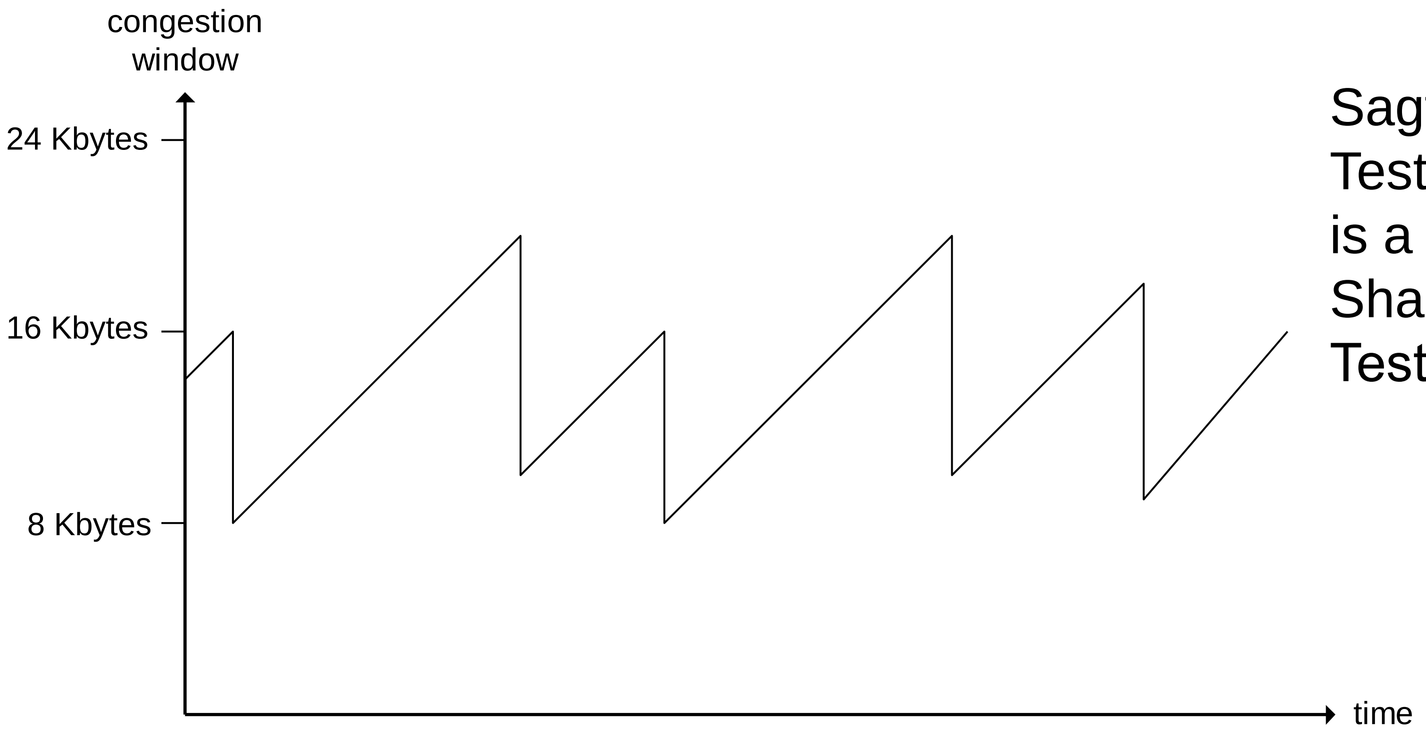
  - Exponentially weighted, moving average
  - Recent measurements weigh heaviest
  - Typical values for x: 1/8 1/10.
- Calculation of timeout
  - *EstimatedRTT plus a safety margin*
  - *The greater the variation in EstimatedRTT, the greater the safety margin*

```
Timeout = EstimatedRTT + 4*Deviation
```

```
Deviation = (1-x)*Deviation + x*|SampleRTT-EstimatedRTT|
```

TCP traffic cork control (AIMD)

- Method: Carefully increase the emission rate (window size), look for available bandwidth, until packet loss occurs.
- Additive increase: increase CongWin by 1 MSS (Max Segment Size) each RTT until packet loss occurs
- multiplicative lowering: cut CongWin in half after packet loss.



Sagtann behavior:
Tests the bandwidth until there is a packet loss
Sharply reduces emissions
Tests again…

TCP AIMD

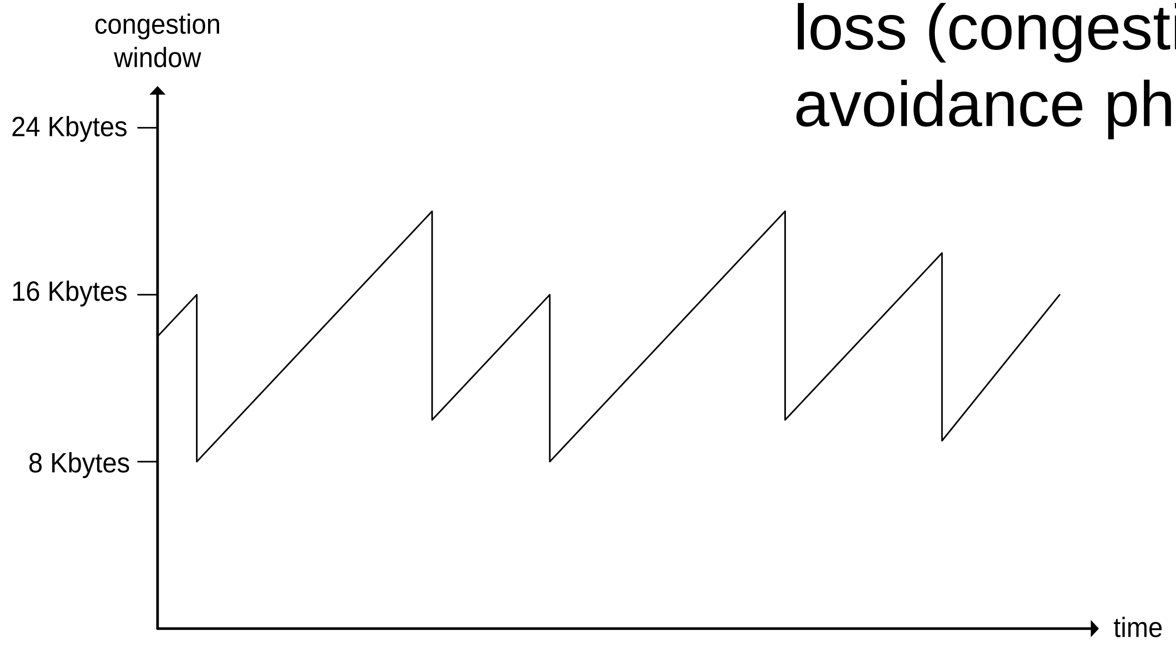
Høyskolen Kristiania

multiplicative reduction (MD):

half **CongWin** after segment loss (minimum one MSS)

additive increase (AI):

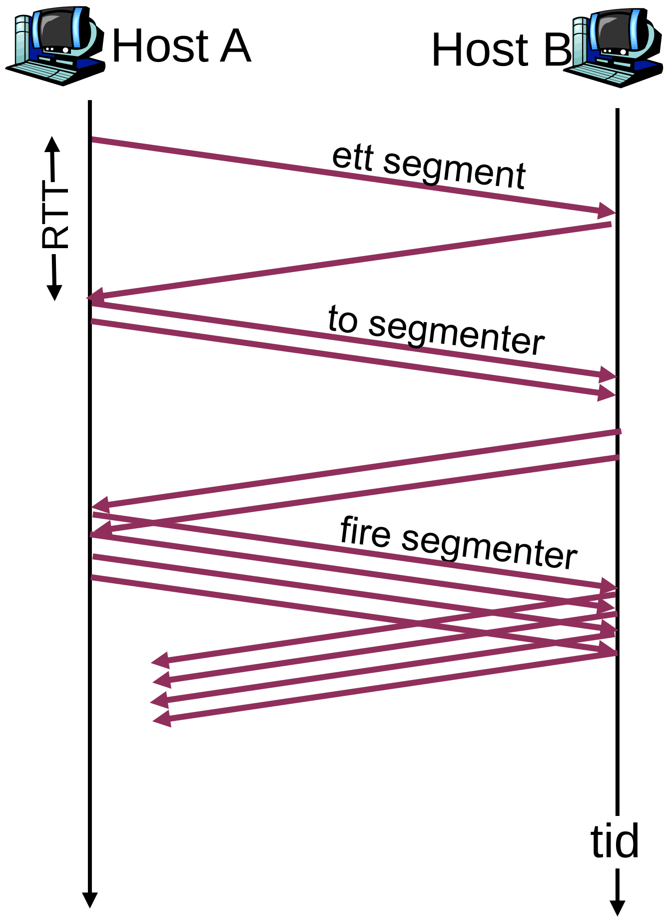
CongWin ncreased by one MSS for each RTT in the absence of segment loss (congestion avoidance phase)



Long-lasting TCP connection

TCP Slow-start

- At the start of the connection, the rate increases exponentially until you experience segment loss:
  - CongWin doubles each RTT
  - done by increasing CongWin for each ACK received
- Summarized: initial rate is low but increasing exponentially



Reaction to segment loss

Høyskolen Kristiania

- After three duplicate receipts:
  - **`CongWin`** is half
  - the window then grows linearly.
- <u>But</u> after the timeout:
  - **`CongWin`** is set to one MSS
  - the window then grows exponentially up to half the value before timeout, and then grows linearly

Philosophy:

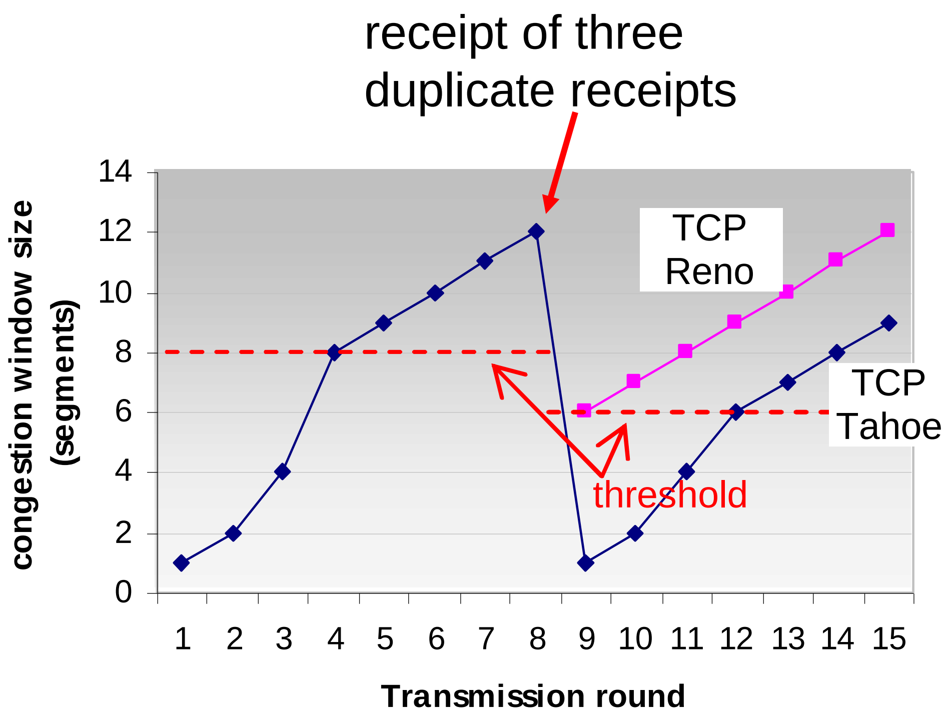
- 3 duplicate ACKs indicate that the network can actually deliver a number of segments as these come through
- Timeout before 3 duplicate ACKs is more alarming, then it seems that very little comes through

Response to segment loss (continued)

Høyskolen Kristiania

Q: Når skal eksponentiell økning endres til lineær?

A: When CongWin becomes half of its value before timeout.



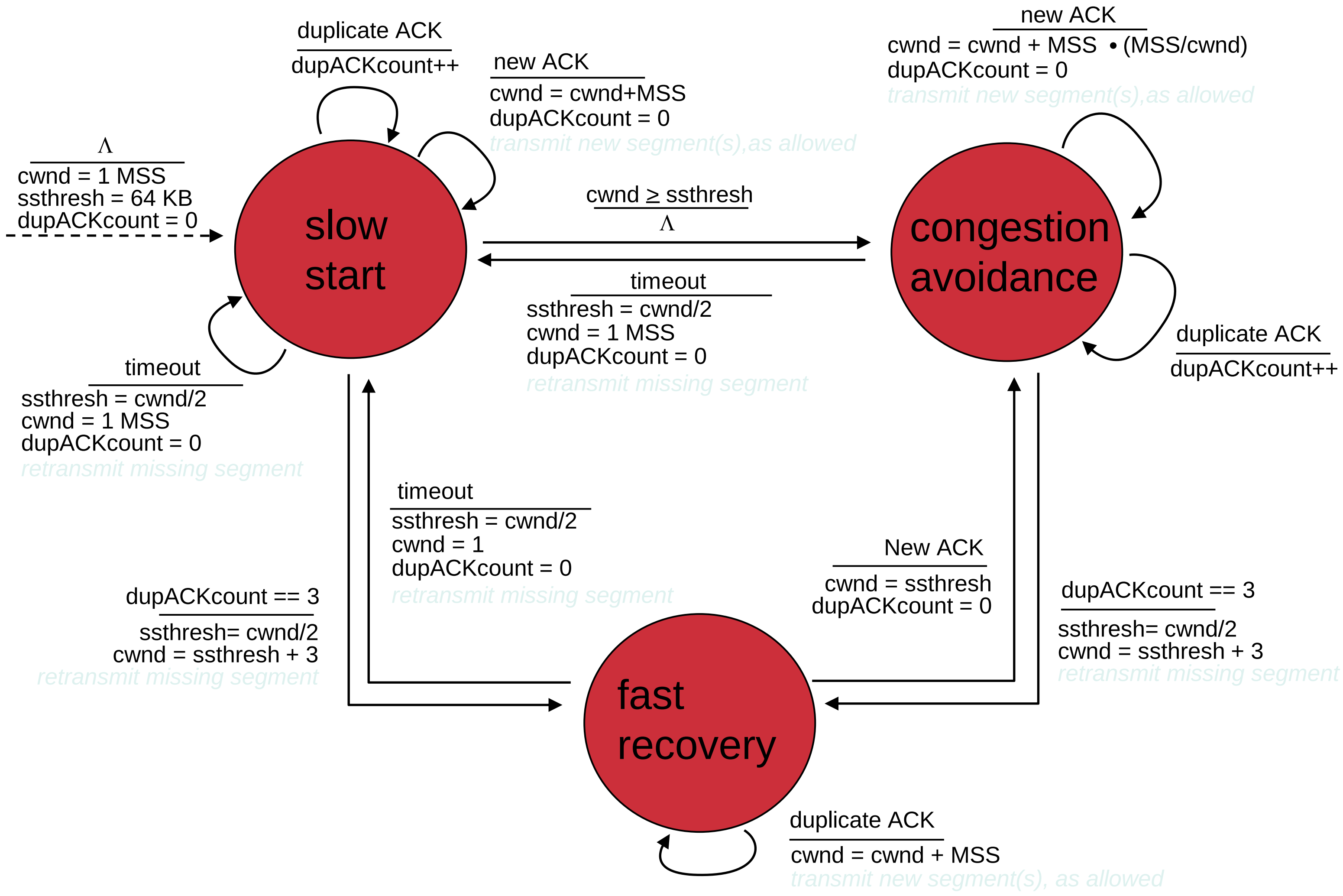
Implementation:

- Variable threshold
- In the event of a segment loss, the threshold is set at half of what CongWin was before the segment loss

Summary: TCP saturation control

- When CongWin is lower than Threshold, the transmitter is in the slow-start phase; window increases exponentially.
- When CongWin is larger than Threshold, the transmitter is in the congestion-avoidance phase; window increases linearly.
- When transmitter receives a triple duplicate ACK, Threshold is set to CongWin / 2 and CongWin is set to Threshold.
- When the sender is timed out, Threshold is set to **CongWin / 2** and **CongWin to one MSS.**

The state machine…

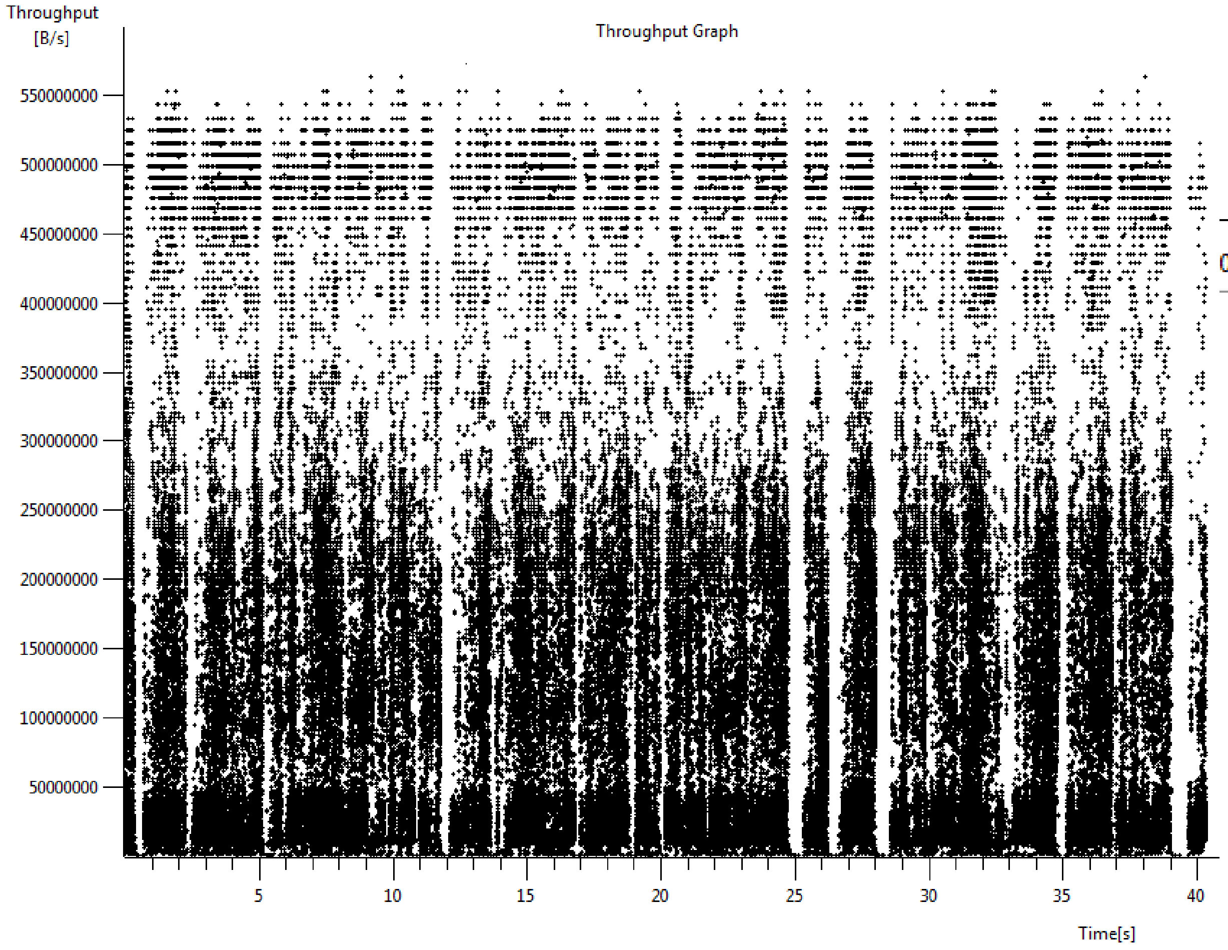


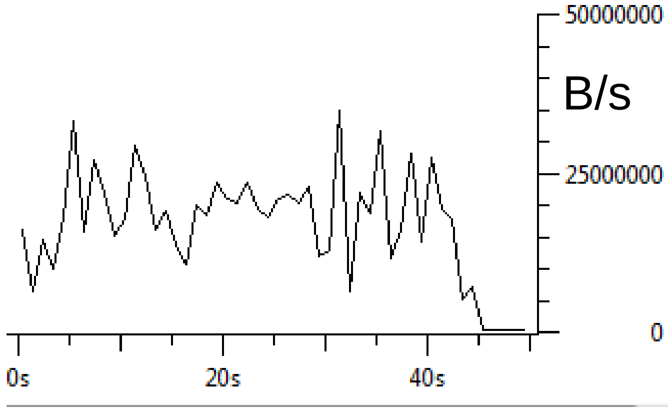
Effective bit rate (example)

- Uses a server in Tromsø (www.uit.no)
- RTT = approx. 22 ms (measured with pathping)
- In classic TCP ("vanilla"), the maximum window size is 64 KiB
- Max (theoretical) flow then becomes (65535 · 8 b) / (0.022 s) = 23.8 $Mbps$, but that is only if there is never a segment loss…
- Automatic Windows scaling will increase this in many modern systems, but this assumes that the server also supports scaling
- RFC 1323 specifies how to switch to Jumbo windows and achieve even greater maximum throughput

Wireshark: Flow

- Uploads 1 GB to home.nith.no





Future of TCP 1: TCP over "long, thick pipes"

- Example: 1500 Byte segment, 100ms RTT, We would like 10 Gbps throughput
- This then requires a minimum window size of W = 83,333 segments on the line
- Flow rate in relation to loss rate (L):

$$\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

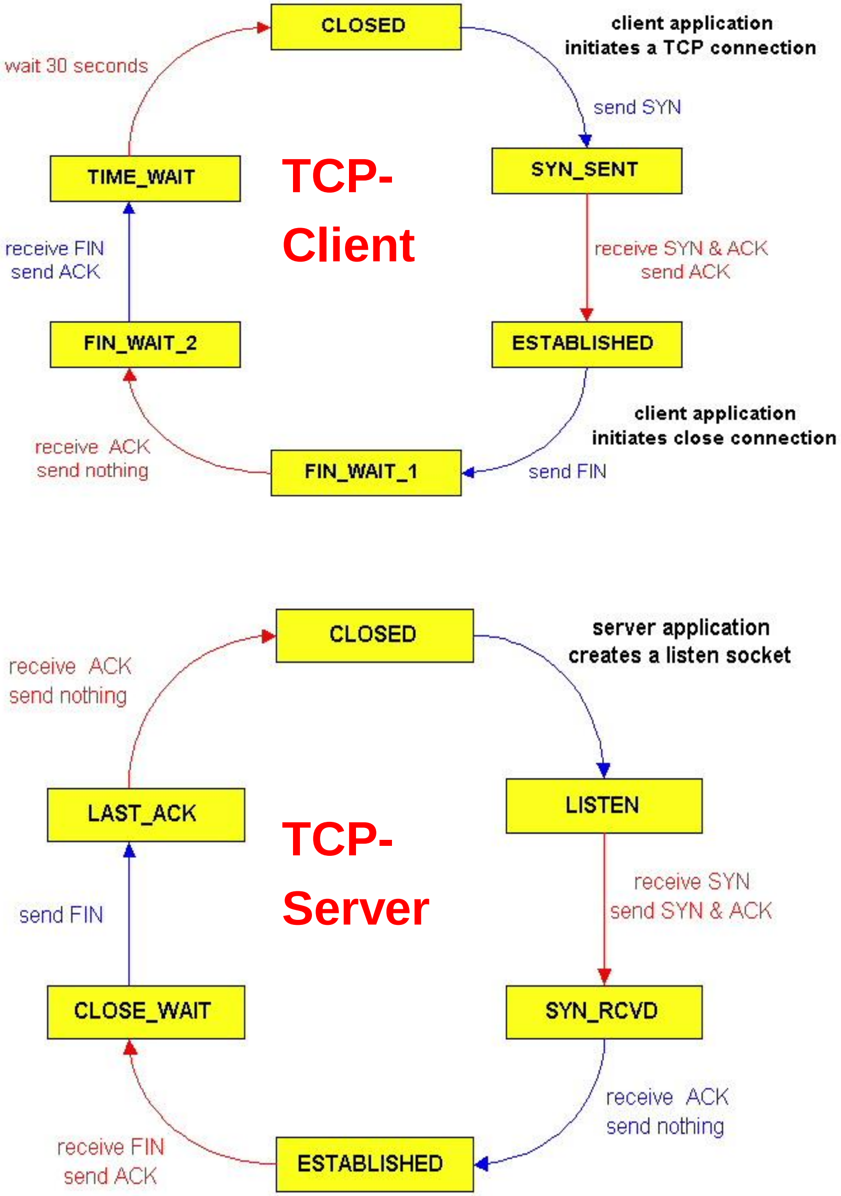
- ➜ $L = 2 \cdot 10^{-10}$ = *only about* ***two out of ten billion packages can be lost.***
- *TCP is also problematic in wireless networks, where packet loss is much more common than in wired.*
- **New versions of TCP are needed!**

TCP's future 2: interactivity over «thin tubes»

- Can use UDP, but must then add custom reliability mechanisms a la TCP into the application itself
- QUIC (Google, Chrome)
- Complicates
- The focus in R&D so far has been on "thick tubes" and high latency, not on interactivity
- Petland (2009) shows that in the game Anarchy Online, old (New Reno) works better than some of the newer TCP implementations!
- In addition, TCP is needed, which detects and takes into account real-time and thin tubes.

TCP: Life Cycle

Høyskolen Kristiania




```
                              +---------+ ---------\      active OPEN
                              |  CLOSED |            \    -----------
                              +---------+<---------\   \   create TCB
                                |     ^              \   \  snd SYN
                   passive OPEN |     |   CLOSE        \   \
                   ------------ |     | ----------       \   \
                    create TCB  |     | delete TCB         \   \
                                V     |                      \   \
                              +---------+            CLOSE    |    \
                              |  LISTEN |          ---------- |     |
                              +---------+          delete TCB |     |
                   rcv SYN      |     |     SEND              |     |
                  -----------   |     |    -------            |     V
 +---------+      snd SYN,ACK  /       \   snd SYN          +---------+
 |         |<-----------------           ------------------>|         |
 |   SYN   |                    rcv SYN                     |   SYN   |
 |   RCVD  |<-----------------------------------------------|   SENT  |
 |         |                    snd ACK                     |         |
 |         |------------------           -------------------|         |
 +---------+   rcv ACK of SYN  \       /  rcv SYN,ACK       +---------+
   |           --------------   |     |   -----------
   |                  x         |     |     snd ACK
   |                            V     V
   |  CLOSE                   +---------+
   | -------                  |  ESTAB  |
   | snd FIN                  +---------+
   |                   CLOSE    |     |    rcv FIN
   V                  -------   |     |    -------
 +---------+          snd FIN  /       \   snd ACK          +---------+
 |  FIN    |<-----------------           ------------------>|  CLOSE  |
 | WAIT-1  |------------------                              |   WAIT  |
 +---------+          rcv FIN  \                            +---------+
   | rcv ACK of FIN   -------   |                            CLOSE  |
   | --------------   snd ACK   |                           ------- |
   V        x                   V                           snd FIN V
 +---------+                  +---------+                   +---------+
 |FINWAIT-2|                  | CLOSING |                   | LAST-ACK|
 +---------+                  +---------+                   +---------+
   |                rcv ACK of FIN |                 rcv ACK of FIN |
   |  rcv FIN       -------------- |    Timeout=2MSL -------------- |
   |  -------              x       V    ------------        x       V
    \ snd ACK                 +---------+delete TCB         +---------+
     ------------------------>|TIME WAIT|------------------>| CLOSED  |
                              +---------+                   +---------+

                      TCP Connection State Diagram
                               Figure 6.
```