CERN School of Computing

Network Programming

*Lecture **1***

LAN Programming – The Basics

Jonas Kunze

University of Mainz – NA62

Inverted CERN School of Computing, 24-25 February 2014

CERN School of Computing

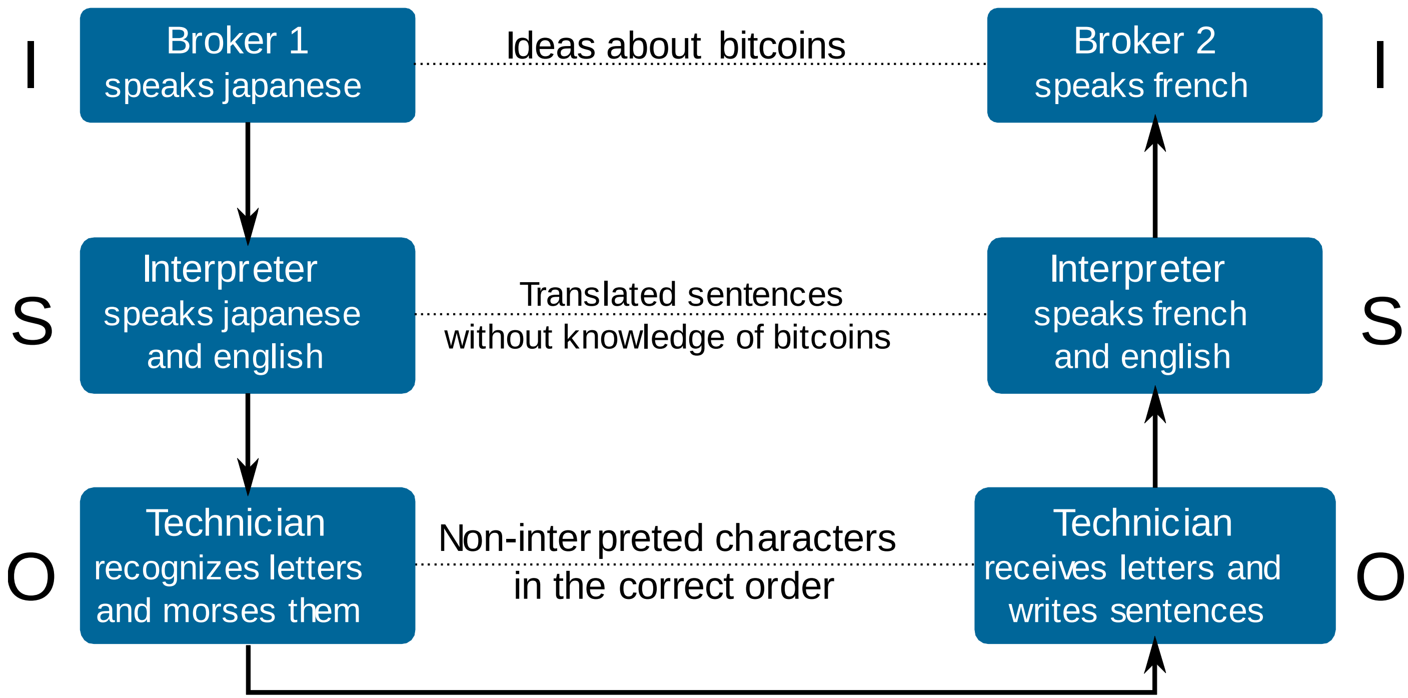
Outline

- **Recap of the TCP/IP model**
  - ISO/OSI and TCP/IP
  - User Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
- Network programming with BSD Sockets
  - Code snippets
  - Performance
- Alternatives to BSD Sockets
  - Network Protocols in User Space

CERN School of Computing

The ISO/OSI reference model

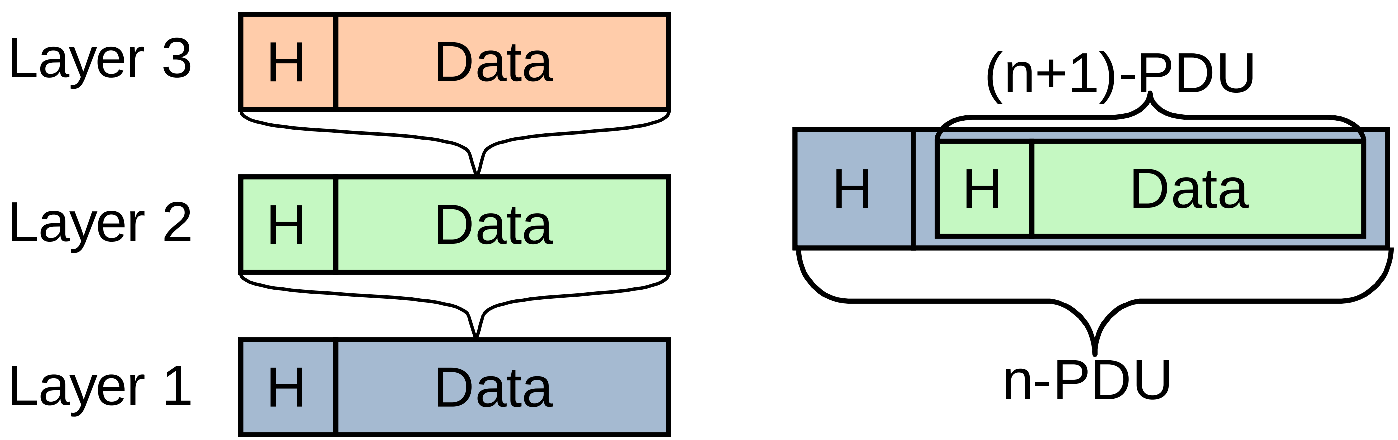
- **Communications protocols are divided into independent layers**
- **Every layer offers a service to the overlying layer**



CERN School of Computing

Interplay between OSI layers

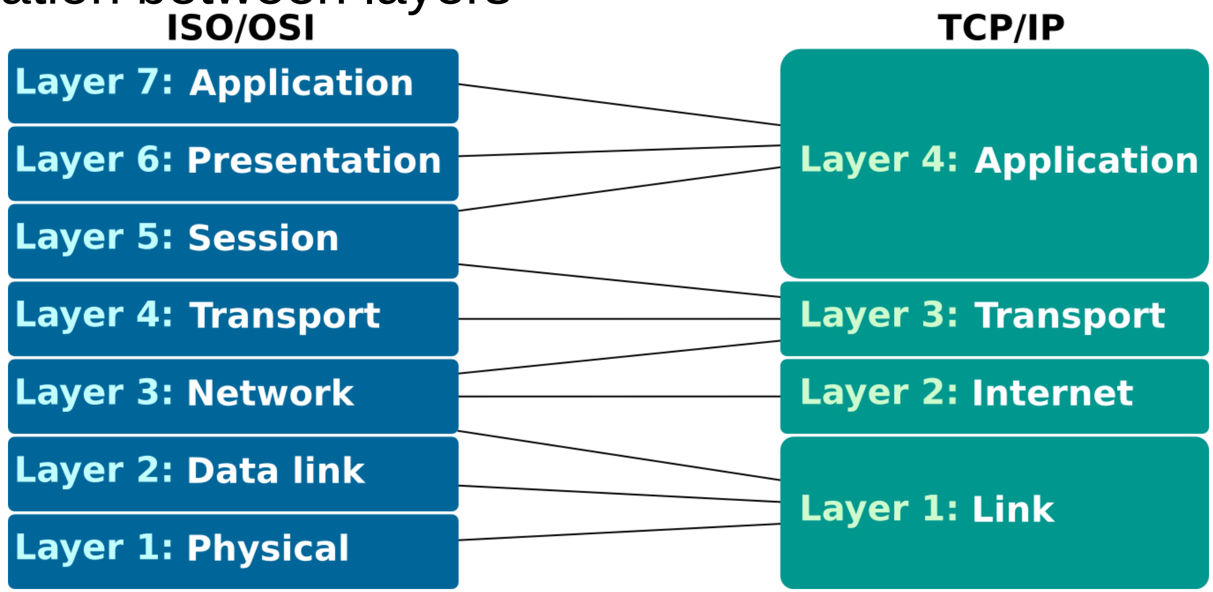
- **Every layer encapsulates the message into a Protocol Data Unit (PDU)**
  - PDUs typically consist of a Header and a Data section
- **Communication partners exchange PDUs by using the next lower layer**



- **Receiver unpacks PDUs in reverse order (like a stack)**

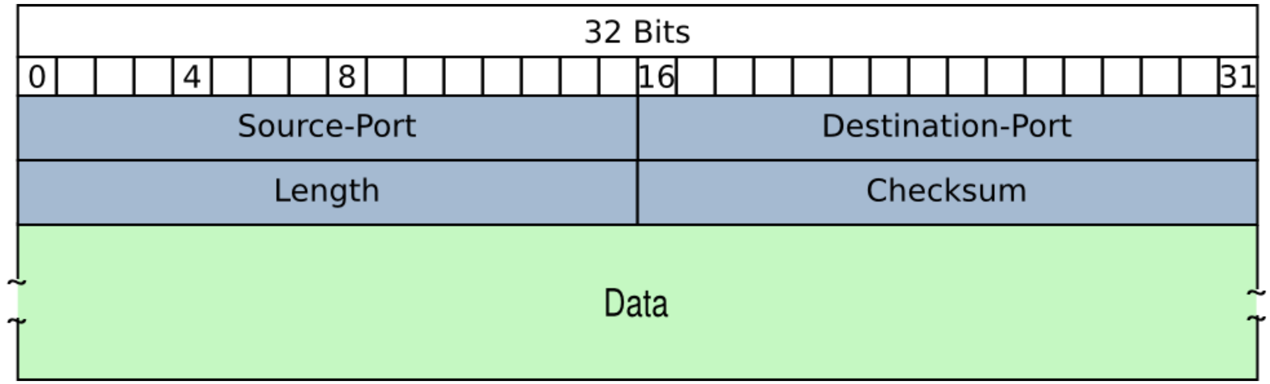
The TCP/IP model

- **The ISO/OSI model is just a theoretical model with almost no implementation**
- **The most common communications protocols are part of the Internet Protocol Suite (TCP/IP model)**
  - Some ISO/OSI layers are merged
  - No strict separation between layers



User Datagram Protocol (UDP)

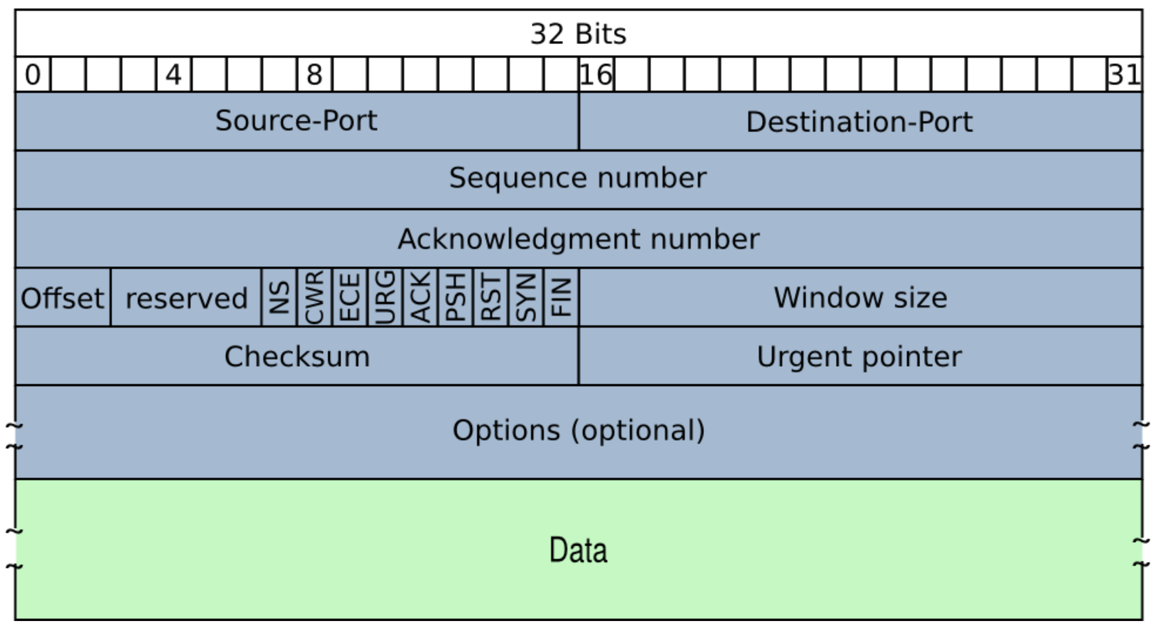
- UDP is connectionless and unreliable like IP



- Source-Port: The port of the process sending the datagram
- Destination-Port: The port number the datagram should be forwarded to
- Length: The length of the whole PDU in Bytes (8 < length < 65535)
- Checksum: Calculated with the whole PDU and data from the IP header

Transmission Control Protocol (TCP)

- **Much more powerful and complex communication service than UDP**
- **Important application layer protocols based on TCP**
  - World Wide Web (HTTP)
  - Email (SMTP)



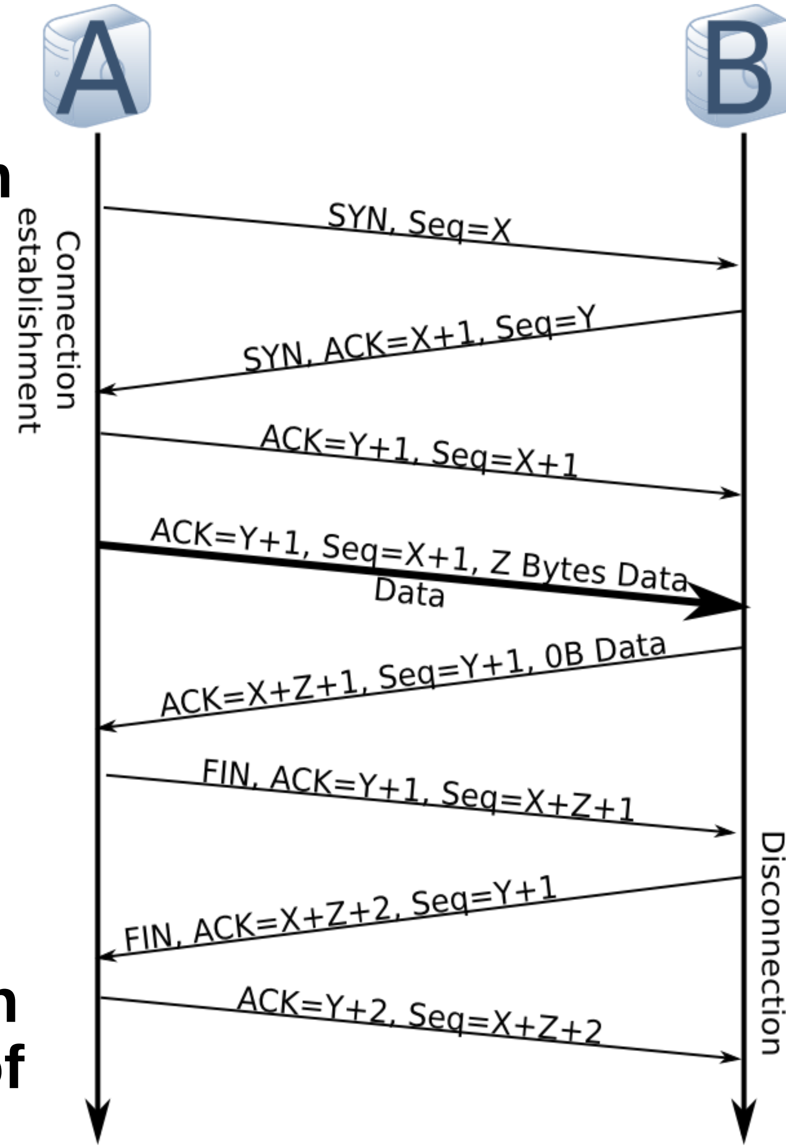
CERN School of Computing

Transmission Control Protocol (TCP)

- **TCP is reliable:**
  - Error-free: fragments are **retransmitted** in case they did not arrive at the destination (timeout)
  - **preserving order** without **duplicates**
- TCP is **connection oriented**
  - Connection establishment necessary before data can be sent
  - Connection defined by IP and **port number** (like UDP) of source and destination
  - Connections are always **point-to-point** and full-duplex
- It implements **flow control** and **congestion avoidance**
- Data is transmitted as an **unstructured byte stream**

TCP data flow

- A sends frame with SYN and random Sequence number X
- B acknowledges with ACK=X+1 and random Sequence number Y
- A acknowledges the reception
- A sends Z bytes
- B increases the sequence by Z to acknowledge the data reception
- Disconnection works like connection establishment but with FIN instead of SYN



CERN School of Computing

Flow Control and Congestion Avoidance

- **Frames are only rarely dropped because of transmission errors (e.g. bit flip)**
  - Connections are typically either working without transmission errors or not at all

- Main reason for dropped frames are overloads of the **receiver** or the **network**

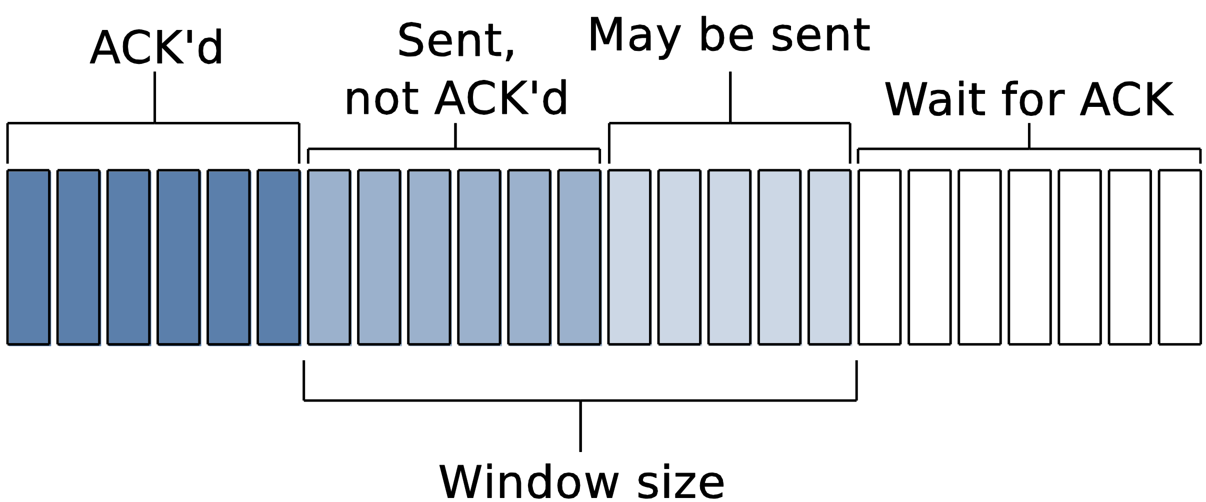
TCP implements two mechanisms to avoid overloading:

- **Flow control:** Avoids overloading of the receiver

- **Congestion avoidance:** Reduces the sending rate in case that fragments are dropped by the network

TCP's Flow Control: Sliding Window

- Each node has a receiving and sending buffer
- In each segment a node specifies how many bytes it can receive
  - Receiver window size: Number of free bytes in the receiving buffer
- If a node has sent as many **unacknowledged** bytes as the window size is large it will stop sending and wait for the next acknowledgment



- **With each acknowledgment the window slides to the right**

TCP's Congestion Avoidance

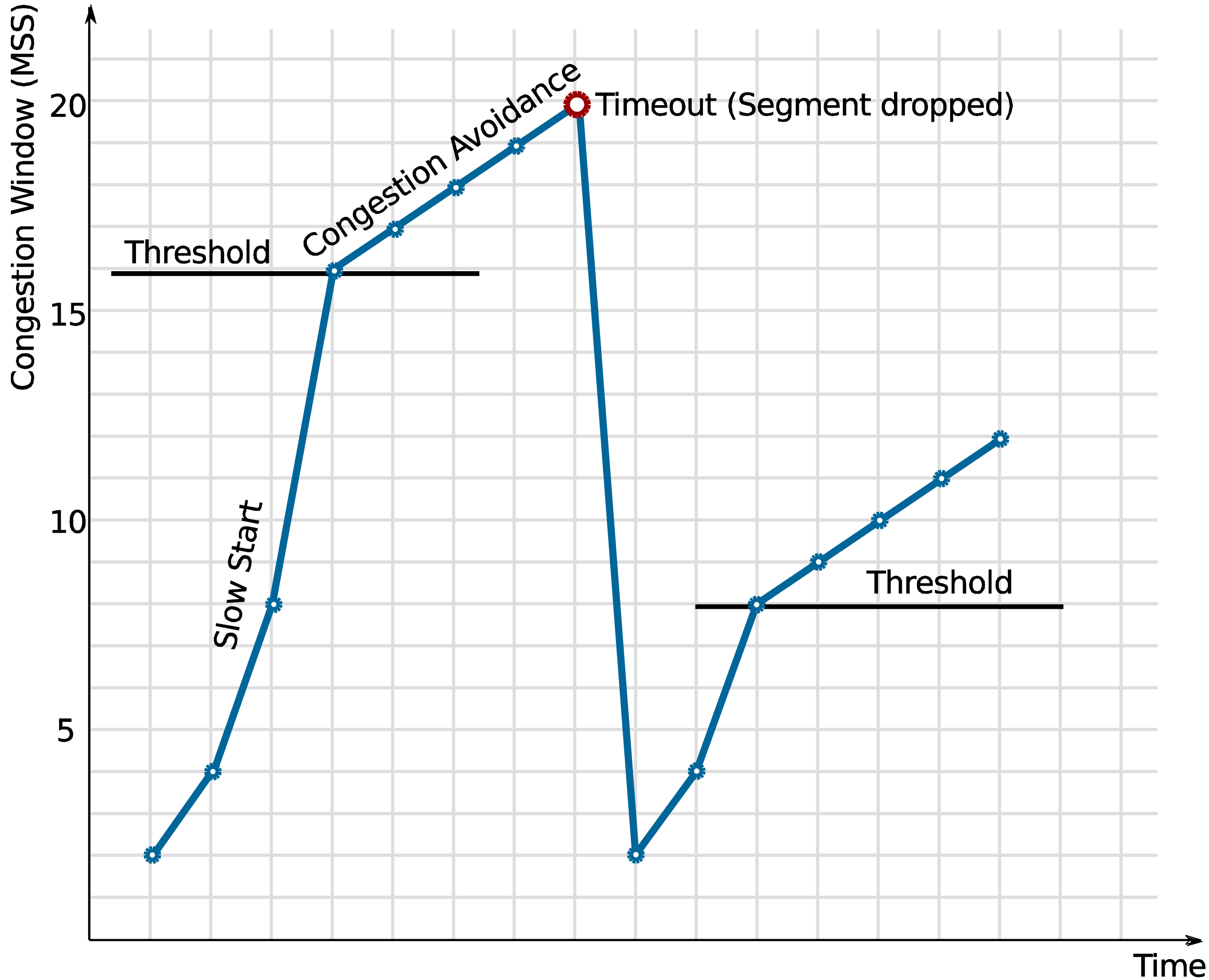
- **Congestion window**: Specifies the maximum number of bytes that may be sent without acknowledgment **depending on the network capacity**

- **Max bytes that may be sent = min(sliding win, congestion win)**

The congestion avoidance algorithm:

- Initialize the congestion window to typically 2 x MSS (slow start)
- Send until one of the two windows are filled
- If a segment is acknowledged: Increase the congestion window
  - Doubled until threshold reached, then linearly
- If acknowledgment timed out (frame dropped by network):
  - Set threshold to half the current congestion window and go back to slow start

TCP's Congestion Avoidance



Sending Buffer

- **When an application sends data chunks to the TCP stack two different approaches can be applied:**

1. Low latency
   - Data chunks sent directly as they are
   - Disadvantage: Many small IP packets will be transmitted (low efficiency)
2. High throughput
   - Buffer data and send larger segments
   - Higher latency but more efficient

Nagle’s Algorithm

- **An algorithm to reach the high throughput approach:**
  - Send first chunk of data arriving at the TCP stack directly
  - Fill sending buffer with new incoming data without sending
  - If the buffer reaches the MSS : Send a new frame clearing the buffer
  - If all sent segments are acknowledged: Send a new frame clearing the buffer

- **Nagle’s algorithm is used in almost all TCP implementations**
  - Can be deactivated to reduce latency (e.g. for X11 applications)

CERN School of Computing

Switch off Nagle's Algorithm

- **This is only rarely necessary!**
- **Within your program:**

```
int flag = 0;
setsockopt(     socket,            /* socket affected */
                IPPROTO_TCP,       /* set option at TCP level */
                TCP_NODELAY,       /* name of option */
                (char ) &flag,     /* the actual value */
                sizeof(int));      /* length of option value */
```

- **System wide:**

```
echo 1 > /proc/sys/net/ipv4/tcp_low_latency
```

CERN School of Computing

TCP vs UDP

- **TCP: A lot of bookkeeping and additional data transmission for acknowledgments**
- **UDP: Just sends the data as it is**

But...

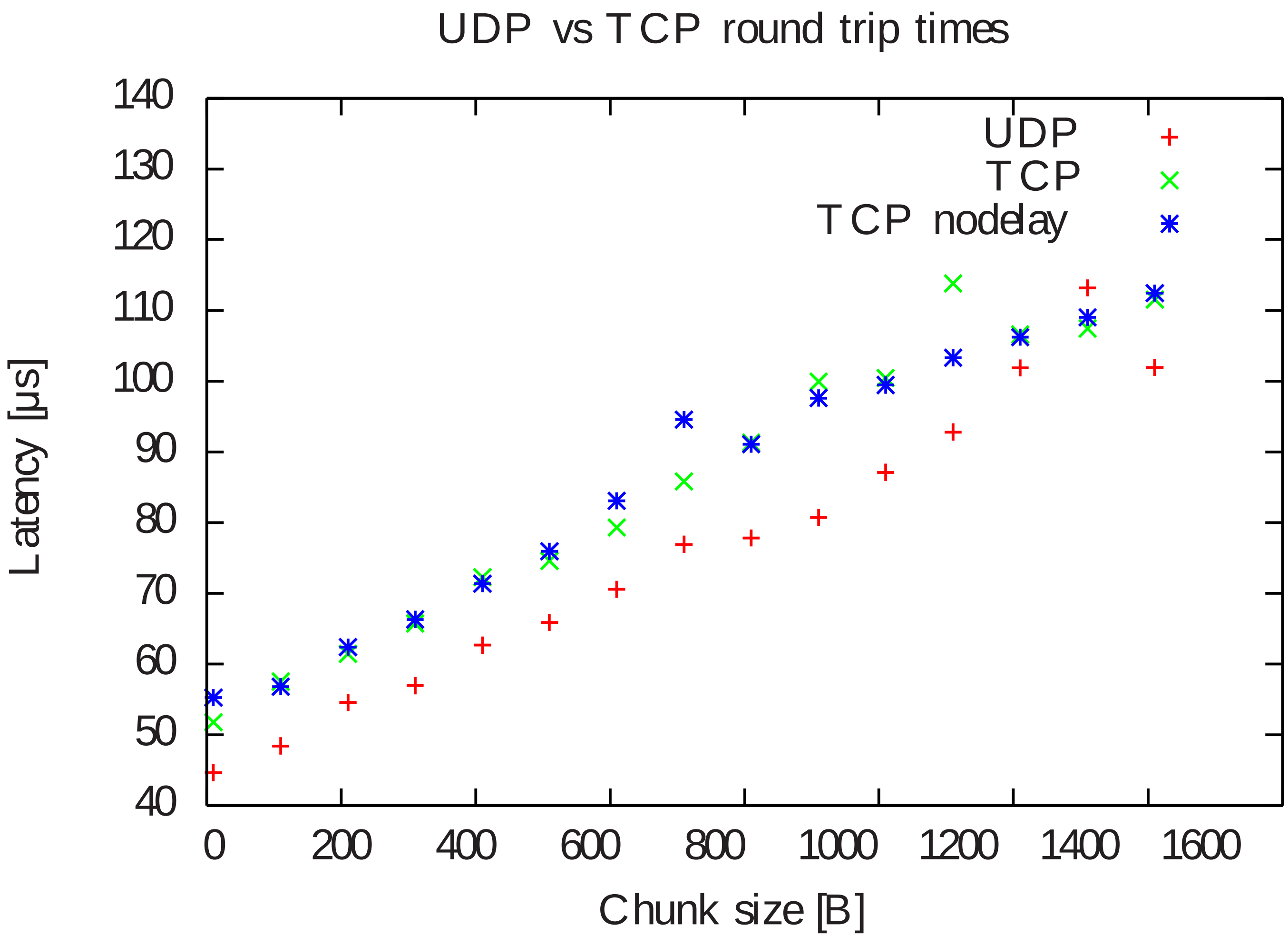
- **TCP: Flow control, congestion avoidance, Nagle's algorithm**

Typical rule of thumb:

- **TCP for high throughput, reliability and/or congestion avoidance**
- **UDP for low latency and broadcasts/multicasts (not possible with TCP)**

CERN School of Computing

A Quick RTT Test



This test was performed with hpcbench: hpcbench.sourceforge.net

CERN School of Computing

Outline

- Recap of the TCP/IP model
  - ISO/OSI and TCP/IP
  - User Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
- **Network programming with BSD Sockets**
  - Code snippets
  - Performance
    - Interrupt Coalescing
    - NAPI
- Alternatives to BSD Sockets
  - Network Protocols in User Space

CERN School of Computing

BSD Sockets

- **Linux supports TCP/IP as its native network transport**
- **BSD Sockets is a library with an interface to implement network communications using any TCP/IP layer below the application layer**
- **Important functions**
  - socket() opens a new socket
  - bind() assigns socket to an address
  - listen() prepares socket for incoming connections
  - accept() creates new socket for incoming connection
  - connect() connects to a remote socket
  - send() / write() sends data
  - recv() / read() receives data

TCP Code Snippet

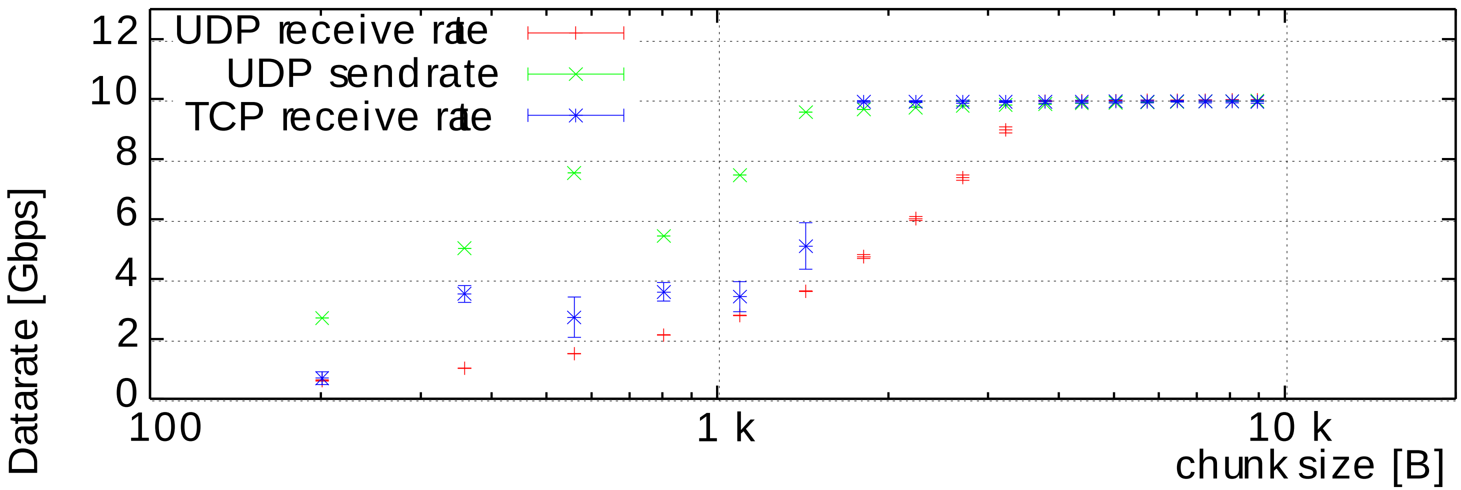
Simple TCP socket accepting connections and receiving data:

```
socket = socket(AF_INET, SOCK_STREAM, 0);
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(8080);
serv_addr.sin_addr.s_addr = INADDR_ANY;
bind(socket, (struct sockaddr *) &serv_addr, sizeof(serv_addr));
listen(socket, 5);
connectionSocket = accept(socket, (struct sockaddr *) &cli_addr, &clilen);
recv(connectionSocket, buffer, sizeof(buffer), 0);
```

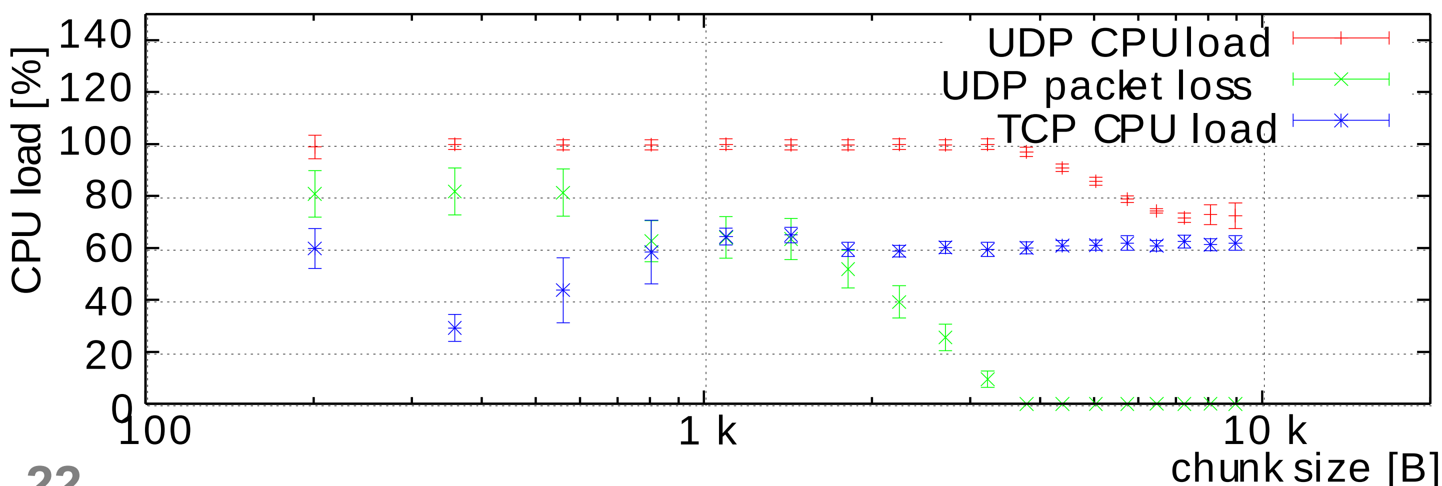
Complete examples to be found at: http://github.com/JonasKunze

TCP vs UDP: Throughput

Single threaded blocking sender and receiver, reliable network

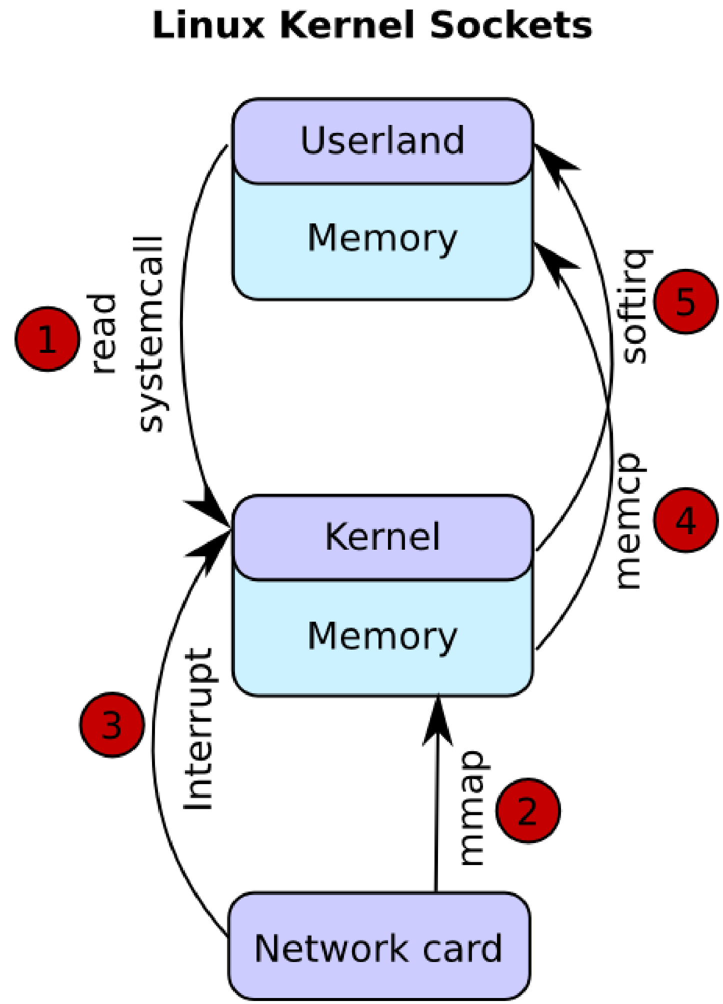


- Small frames induce high CPU load → packet loss
- TCP achieves higher throughput



Down to the Kernel

- **When data arrives at the NIC:**
  - Data **copied** to kernel space (DMA)
  - NIC sends **interrupt**
  - Kernel **copies** data to the corresponding user space buffer (socket)
  - Kernel informs user space application



CERN School of Computing

Interrupt Coalescing

- **Technique to reduce interrupt load**
- **Interrupts are held back until...**
  - … a certain number of frames have been received...
  - … or a timer times out
- **Now the kernel can process several frames at once**
  - Higher efficiency with just little increase of latency

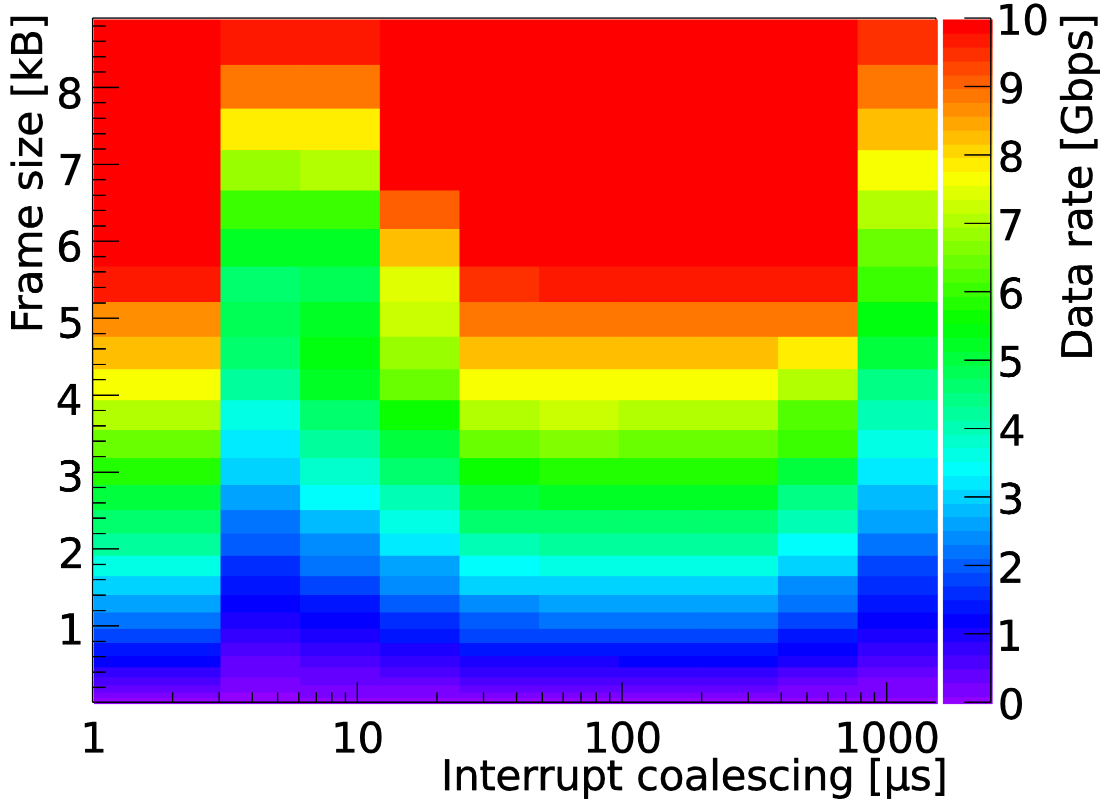
```
# print current settings
ethtool -c eth0

# change settings
ethtool -C eth0 rx-usecs 0 # 0 is adaptive mode for many drivers
ethtool -C eth0 rx-frames 12
```

CERN School of Computing

Interrupt Coalescing

- **Small values overload the CPU → Packet loss**
- **High values lead to buffer overflow → Packet loss**



First bin shows adaptive mode

CERN School of Computing

NAPI

- **An alternative to interrupts is polling:**
  - Kernel periodically checks for new data in the NIC buffer
    - High polling frequencies induce high memory loads
    - Low polling frequencies lead to high latencies and packet loss
- **NAPI: Linux uses both**
  - Interrupts per default
  - Polling in case of high data rates incoming

The kernel still needs to copy incoming data!

CERN School of Computing

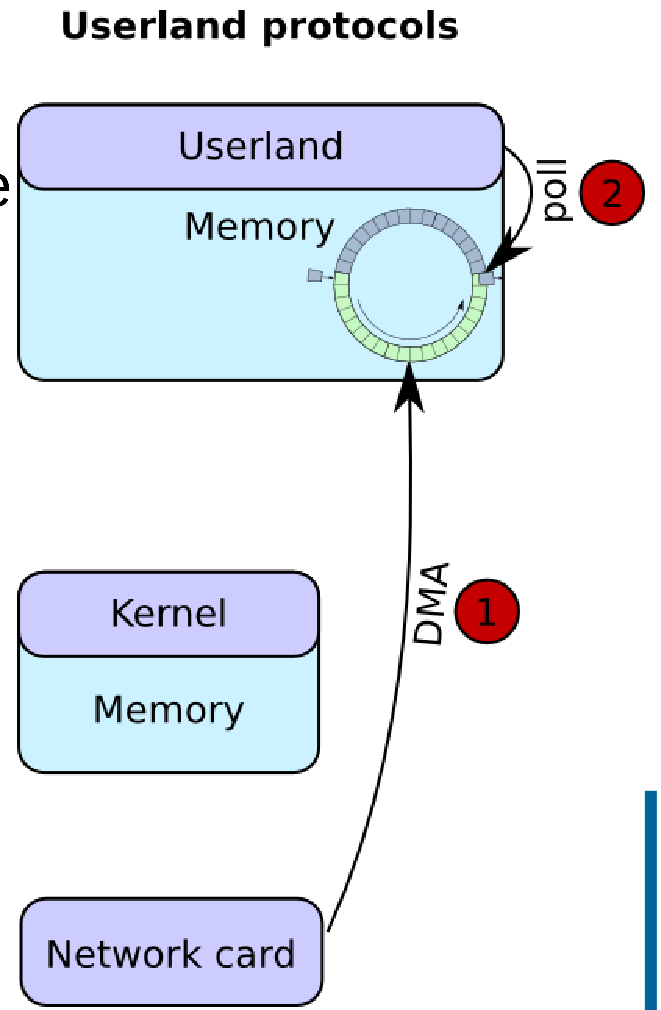
Outline

- Recap of the TCP/IP model
  - ISO/OSI and TCP/IP
  - User Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
- Network programming with BSD Sockets
  - Code snippets
  - Performance
- **Alternatives to BSD Sockets**
  - Network Protocols in User Space
    - Example: pf_ring DNA
    - Reliability on top of UDP?
    - Reliability without acknowledgment

CERN School of Computing

Network Protocols in User Space

- **Following approach can be implemented in the user space to avoid double copies**
  - NIC copies incoming data to a user space buffer (DMA)
  - The user space application polls the buffer
  - The user space application may enable interrupts for low data rates
  - The kernel is only used for the initialization
- **0% CPU used for accessing the data**



CERN School of Computing

Example: pf_ring DNA

- **Proprietary user space driver by ntop**
- **Does not implement any protocol**
  - You need to implement them: ETH, IP, UDP, TCP, ARP, IGMP...
- **Compatible with all 1 GbE and 10 GbE NICs running on PCI-E**
- **Full line rate (1-10 GbE) with any frame size**
- **Round trip time below 5 µs**
- **Hardware filtering (only Intel and Silicom NICs)**
  - Very efficient Intrusion prevention systems possible (Snort)
- **Other userspace drivers: Netmap, Intel DPDK, OpenOnload**

Reliability on top of UDP?

- **At CERN experiments most data senders are FPGAs**
  - Very fast in parallel jobs
  - Typically fully loaded by algorithms
    - Sometimes there's no space left for a fully implemented TCP/IP stack
- **I've seen many groups implementing reliable protocols on top of IP**
  - In most cases the result was TCP without flow and congestion control
- **Being compatible with TCP/UDP relieves the software developers**
  - You don't need to implement the protocol on the receiver side
  - Instead you can use standard libraries

Reliability without acknowledgment

- **Sometimes it's not even possible to store data until the acknowledgment is received**
  - You should use pure UDP in this case
- **As soon as datagrams are sent out you have to trust the network**
  - Make sure that you don't overload switches/routers/receiver nodes
  - Check every node whether frames are dropped

Switch/Router:	**Linux:**
show interfaces ...	cat /proc/net/udp

CERN School of Computing

Summary

- **TCP is more than just reliable**
  - It implements a maximum efficient data transmission
- **BSD sockets provide a nice API for simple network programming**
  - For more complex architectures networking libraries are recommended
- **Linux' network sockets are not as efficient as they could be**
  - High performance network drivers provide efficient alternatives to BSD sockets but they generate additional work for the developer team