Performance-optimised computing – Lecture 6.

Inter-process and Network Communication

Dr. Bakay Árpád – Ericcson

Recap: File operations in Unix, Linux, Posix

- Basic, low level operations using „file descriptor" (which is an int)
  - int fd = **open**(<pathname>, <flags>, [<create_mode>]);
  - **read**(fd, <buffer>, <count>) / **write**(fd, <buffer>, <count>)
  - **close**(fd)
  - fd is process-specific (unique within a process)
- „POSIX"/C11 Standard file ops (also available on Windows)
  - fopen(), fread()/fwrite(), fclose() + fflush(), fprintf()/fscanf(), fgetc()/fputc(), fdopen()
  - Differences:
    - **Posix compliant**
    - **struct FILE*** reresents an open file
      - It is possible to convert either way
    - buffered, formatted input/output: fprintf/fscanf
    - Implemented with library (libc) functions (uses above system calls)

Inter-process Communicaton in Posix

- Shared memory
  - Mutiple processes open a memory area, in RO or RW mode.
  - Direct & fast
  - Backed by a name, can also be „MAP_ANONYMOUS“ - only seen by descendant processes
  - Synchronization and consistency is up to the developers.
  - API: shm_open(<filename>)
- Pipes -> see next slide
- Unix sockets (and network sockets) -> see 2nd next slide
- Further IPC mechanisms
  - Signals - no data, just notifications without content.
  - Semaphores – used for synchronization
  - Message queues - obsolete
  - STREAMS – obsolete (system V)

Pipes

- 2 file-descriptors – one for writing and one for reading – linked together
  - What is written into the write fd, is received by the reader
  - Some amount (e.g. 64k) of buffering is also provided
- Unnamed:
  - Created by a process [with **pipe() syscall**], inheritable by descendant processes only
  - Process only receives a pair of file descriptors: one for reading and one for witing only
  - Read, written & closed like files
  - Multi-write access possible
- Named
  - Created as a „node" in the filesystem, with the **mkfifo(<pathname>)** libc function
  - Opened, written, read & closed just as ordinary files.
  - Multi-write access is also possible

Sockets – general

- Sockets are a generalization / enhancement of pipes.
  - They are bidirectional
  - They can span remote computers
  - They are slightly more complicated
- Sockets exist in one of several **„domains“;** these days only 3 are in use
  - **AF_UNIX:** generalized pipes, only for local machine, identified by a name /myunixsocket
  - **AF_INET** and **AF_INET6:** IPV4/IPV6, can span the network,
    - Identified by {protocol, src_addr, src_port, dst_addr, dst_port}
- **Socket type** selects method of communication within a domain
  - **SOCK_STREAM**: stream (of bytes), in AF_INETx, this is TCP
  - **SOCK_DGRAM:** sequence of packets, in AF_INETx, this is UDP
  - Others, rearely used: SOCK_RAW (for Inet), SOCK_SEQPACKET for Unix)

Unix Sockets - unnamed with **socketpair()**

Similar to unnamed pipes, the only major difference it is bidirectional:

int socket_fds[2]

socketpair(AF_UNIX, SOCK_STREAM, 0, socket_fds);

... *typically fork() here*

write(socket_fds[0], „hello“, 6);

... *in another process, e.g. Child:*

unsigned char buffer [10];

read(socket_fds[1], buffer, 10);

... **close(socket_fds[0]); close(socket_fds[1]);** // must close in each process

Unix Sockets - Normal/named with **socket(), bind(), send/sendto(), recv/recvfrom()**

Server (passive) or client (active) sides

int sock;

sock = socket(AF_UNIX, SOCK_DGRAM, 0); // error handling omitted

struct sockaddr_un sock_name; // a structure to hold adress family and address
name.sun_family = AF_UNIX;
strcpy(**name.sun_path**, "/my_path/my_socket"); // looks like a filename, but it can be anything . /my_path is not a directory

On endpoint 1 (typically: server):
bind(sock, &sock_name, sizeof(struct sockaddr_un)**) ;** **//** this expects a "sockaddr", "sockaddr_un" is a specialization
read(sock, buffer, sizeof(buffer)**)** and **write(sock,** „hello", 6**);**

On endpoint 2 (typically: client)

- **alternative 1 – connection-oriented**

connect(sock, &sock_name, sizeof(struct sockaddr_un)**)**
write(sock, „hello", 6**)** and **read(sock,** buffer, sizeof(buffer)**)**

- **alternative 2 – connectionless - we provide (or receive) the address with each call**

sendto(sock, „hello", 6, 0, &name, sizeof(struct sockaddr_un)) and recvfrom(...)

AF_INET: SOCK_STREAM (a.k.a. TCP) and SOCK_DGRAM (a.k.a. UDP)

- „Transmission Control Protocol" **1974**, „User Datagram Protocol" **1980**
- Defined by IETF RFC-s (ietf.org/rfc/rfc-index-latest.txt) : TCP: 761 -> 9232 UDP: 768
- Both are „transport layer" protocols over IP „network protocol"

- UDP is simple, connectionless, no guarrantees, light load on Kernel
  - UDP Header is minimalistic: ports, length, optional checksum 8 bytes in IPV4
- TCP requires connection setup, guaranteed delivery in correct order, heavy load on Kernel.
  - Connection setup and teardown with **handshake**
  - Acknowledges receipt of every data, **retransmissions** happen if ack is not received
    - Dynamically adjusted window size for unacked data
  - Header (20-60 bytes) has sequence counters in both directions, flags, 20

Programming AF_INET DGRAM Sockets

Only the address assembly is changed relative to AF_UNIX

sockaddr_in is another specialization of the sockaddr struct, with AF_INET family (used as type dicriminator) plus a 32 bit address and a 16 bit port

sock = socket(AF_INET, SOCK_DGRAM, 0); // this tells the kernel to clreate ans Internet socket, with UDP

On client:

struct sockaddr_in sock_name;
sock_name.sin_family = AF_INET; // address type must match sockt type
sock_name.sin_port = htons(5000);
struct hostent *hp = gethostbyname(„server.mydomain.hu");
bcopy(hp->h_addr, **&sock_name.sin_addr**, hp->h_length);

} *Ugly, obsolete API to create socket addresses*

connect(*sock, sock_name, sizeof sockaddr_in) … write() / read() or sendto() - same AF_UNIX on previous slide*

On server:
struct sockaddr_in sock_name;
sock_name.sin_family = AF_INET; // address type must match sockt type
sock_name.sin_port = htons(5000);
sock_name.sin_addr.s_addr = INADDR_ANY; // servers typically accept clients on any of the machine's IP adresses

} *Similar*

bind*(sock, sock_name, sizeof sockaddr_in) …. read()/write() like with AF_UNIX*

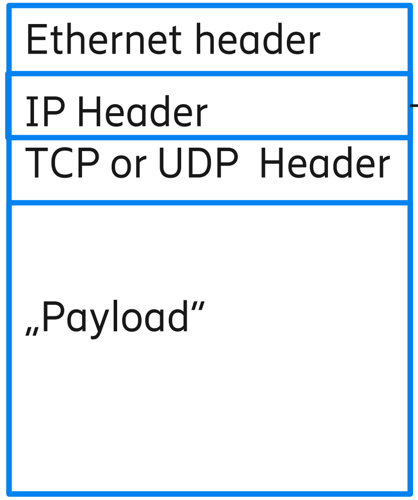
Programming TCP

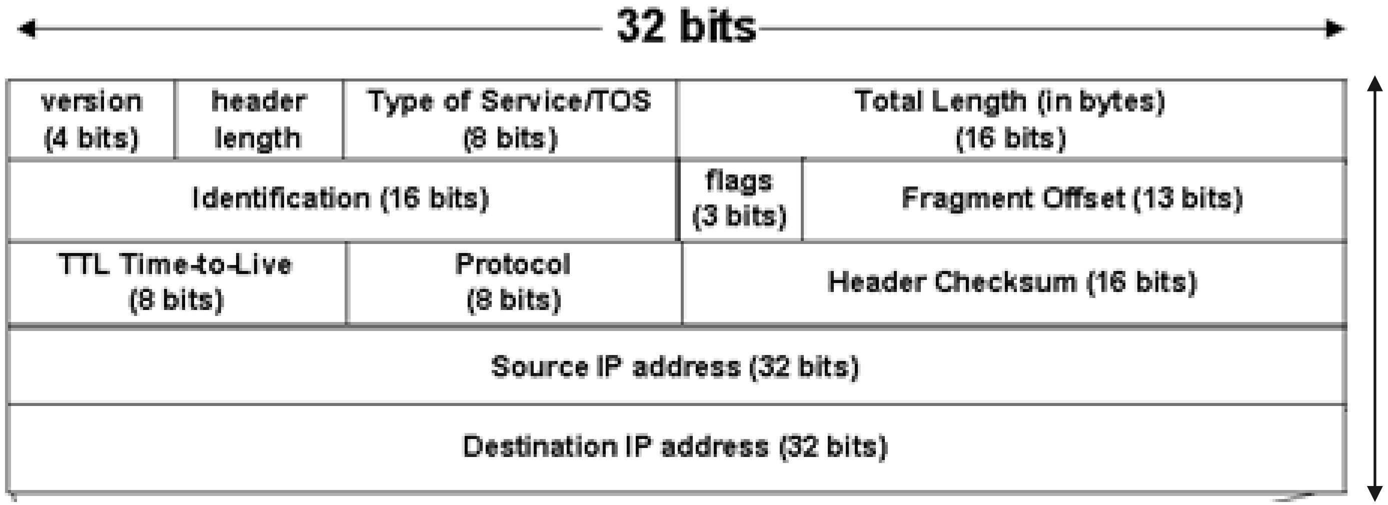
- Socket created with AF_INET, SOCK_STREAM
- bind() is again mandatory for the server (just as with UNIX and UDP sockets)
- **listen()** is a new, **TCP-specific step for the server code**. It waits for client connections, and returns **a new socket** when a connection arrives.
  - The new socket is already connected, server can send and receive data on it (same as UDP)
  - **listen()** may be called again while connected sock is being used
    - The connected socket is often served by a forked process or thread, to allow for multiple simultaneous connection to a service.

```
int server_sock = socket(AF_INET, SOCK_STREAM, 0);   // socket is only used for listening, no reads/writes!
bind(sock, sock_name, sizeof sockaddr_in) ;
int connected_sock = listen( server_sock, 5);     // 5 is the backlog, we accept 5 connection
if(connected_sock >= 0)  read(connected_sock, ....)  .....
```

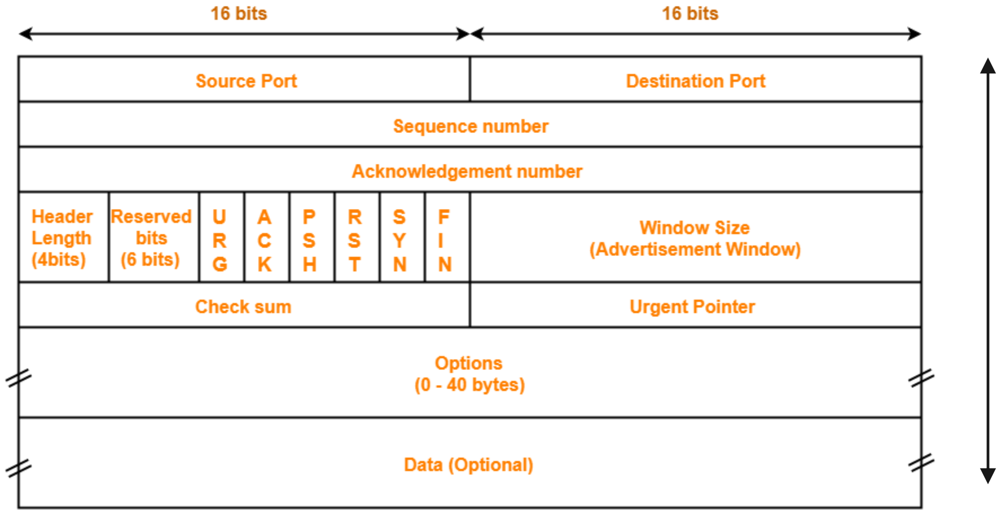
IP, TCP, UDP Headers

Encapsulated in a packet





IP Hdr
20 bytes



TCP Header

TCP Hdr
44 bytes

UDP Header

UDP Hdr
8 bytes

Ethernet header a „physical" layer, not shown here

TCP/UDP Protocols in practice

- **TCP is more popular**, and has evolved significantly from the early Internet years
  - Much-much **higher bandwidth** (from 300 bps up to 100Gbps - and beyond)
  - Various **application-level protocols**: HTTP, SMTP, FTP, Telnet/SSH, DB connections, etc.
  - **Security layers**: TLS (or SSL) between transport and application layers.

- **UDP** is used where **transmission time is more important** and data loss is tolerable
  - Video and audio streaming
  - DNS, NTP, DHCP, BOOTP
  - Gaming
- **Multicast** is possible, e.g. IPTV

UDP Usage - Quick overview

- UDP is used where
  - Data losses are not critical
  - Data rate is „naturally limited“, e.g. media streams
  - Data with packetized nature.
  - Frequent, „single packet“ sessions, e.g. DNS
    - UDP has much lower kernel footprint.
  - Simplicity, e.g. TFTP at network boot
  - Only UDP is suitable for multicast communication

- Error correction for UDP is possible with non-standard methods
  - Forward EC: send redundant info in stream
  - Backward EC: clients can request missing packets
- Auxiliary protocols for quality monitoring, e.g. RTCP for RTP

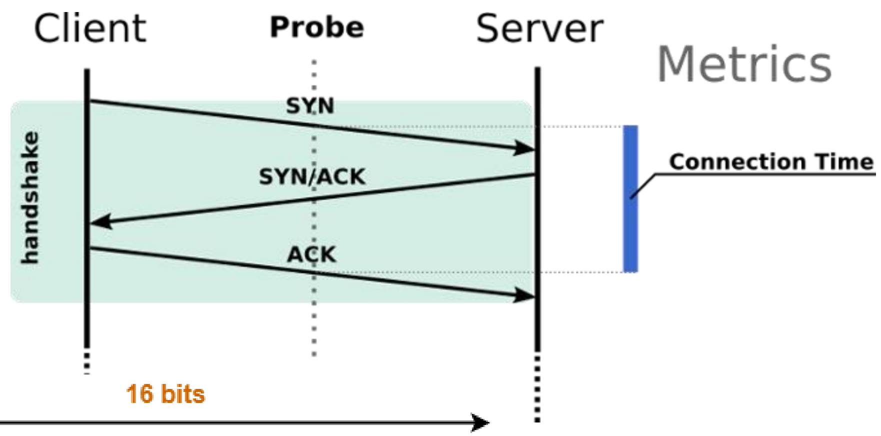
TCP Protocal – Quick Overview

- Multiple „lifecycle phases” for each connection
  - Connection setup
  - Data transfer (uni- or biderectional, often conversation-like).
  - Termination

- Messages in all phases are acknowledged!
  - ACK may come with data in a message (i.e. ACK flag in header + non-zero payload)
  - Each packet contains 2 „sequence counters”, for sent and received bytes (based on the status viewed by the sender)
    - ACK is sent back when the received sequence counter is advanced.

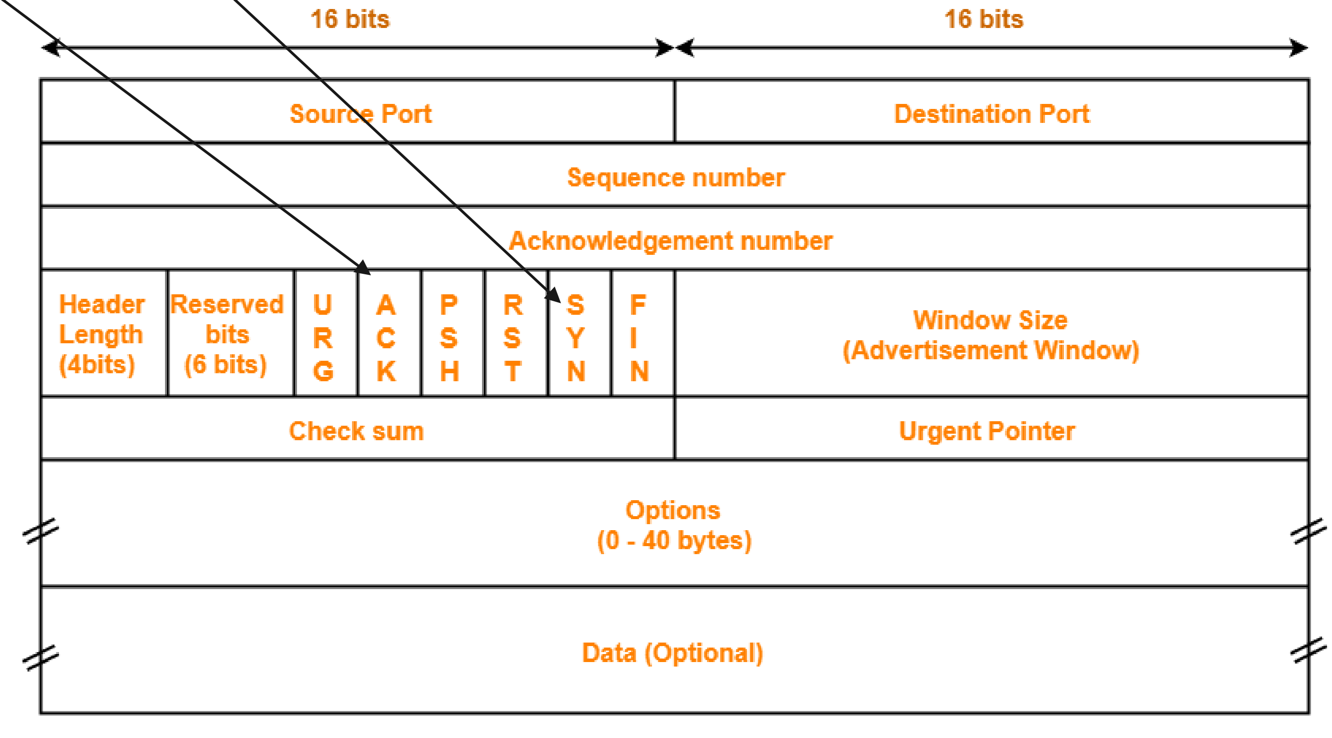
TCP Protocal details - Connection setup

- 3 Messages with 2 „SYN“ packets
  - Client -> Server: SYN
  - Server -> Client: ACK + SYN
  - Client -> Server: ACK (+ optional data transfer, e.g. HTTP request)



Flags are 1 bits each

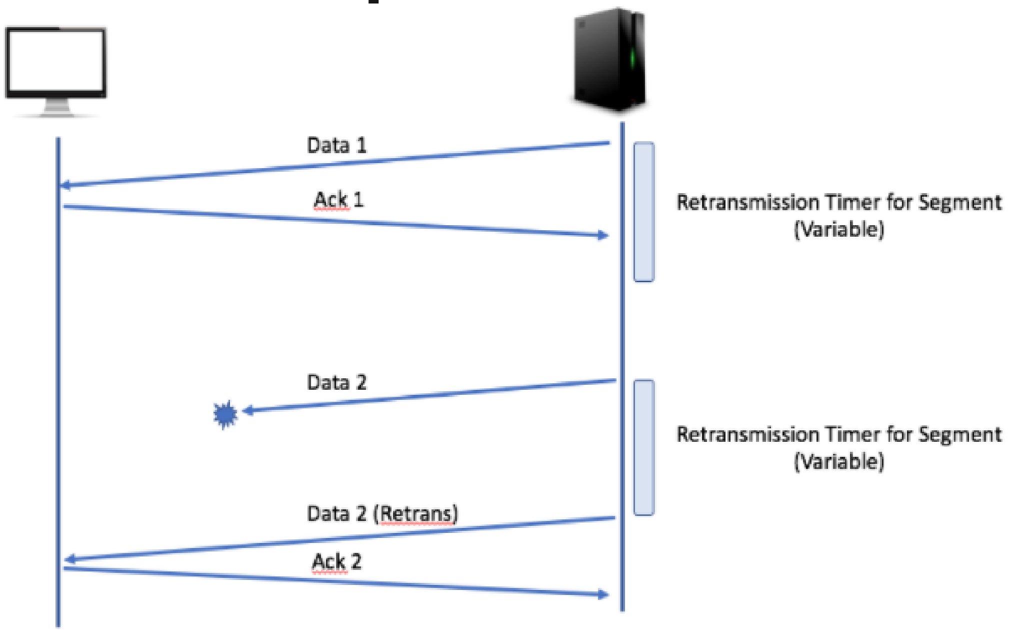
- SYN is only set during setup
- ACK is used in all phases
- FIN and RST used at termination (see later)



TCP Header

TCP Protocal details - Data transfer phase

- Each party can send packets with data payload
  - When payload is present, sender sequence is typically advanced



- Receiver sends packets with ACK + the received seq. count (may also contain payload in the reverse direction).

- If some packets are not acknowledged within a timeout, the un-ack-ed payload is **retransmitted**.
  - Timeout is dynamically tuned, based on delay of earlier responses

- Each sender has a „window" for the max count of non-acknowledged bytes sent
  - Window is also dynamically adjusted: gradually increased, but reduced if many losses are experienced,

TCP Performance Issue Sources

- Packet losses may be due to:
  - Bad, „noisy" network links
  - Over-capacity use of vertices (network links) or nodes (routers and switches)
    - Network nodes do not hesitate to throw away packets - no guarranteed delivery on IP level

- Receiver-side congestion
  - If consumer is slow, kernel will accumulate recieved data only up to a limit.
    - Window in ACK will be set to `0`, producer must wait

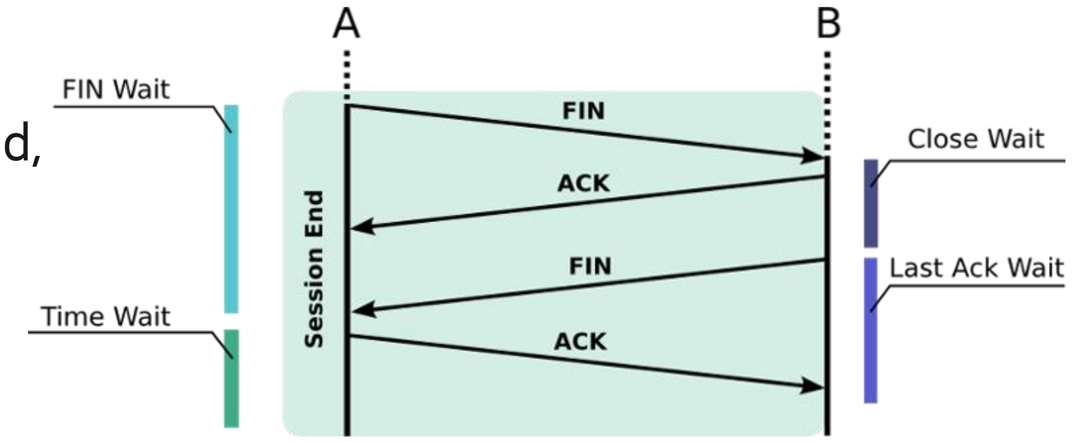
TCP Protocal details - protocol optimization for high bandwidth

- Window scaling
  - Originally, window size was interpreted in bytes, up to 64k. Nowadays it is interpreted in up to 16kbyte units, allowing window of up to 1GBytes
  - Set at the SYN phase
- Selective Acknowlegements
  - Receiver can specify gaps in the received data, which allows the sender to replace the missing packets only.
  - Very useful with large windows.
- Further options defined for security, window scaling, etc

TCP Protocal details - 2 ways for Termination

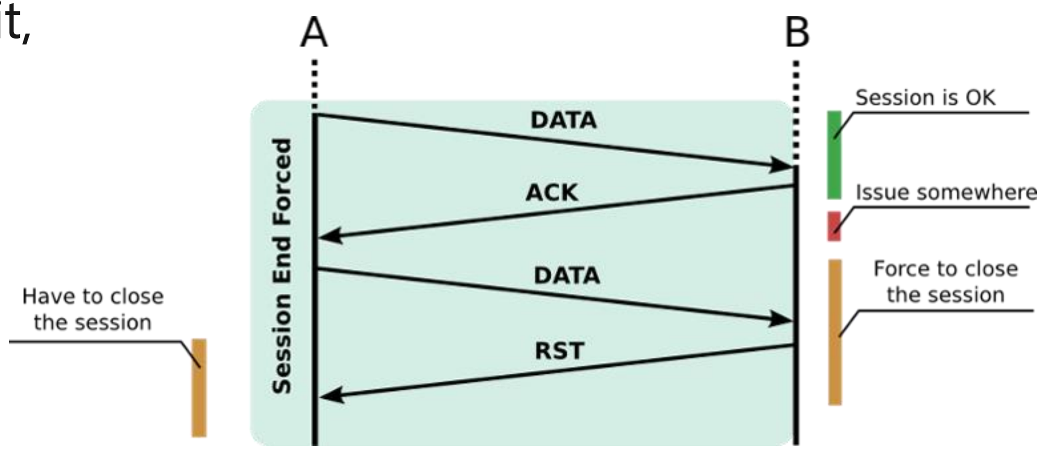
- **FIN + FIN**: When a party has nothing more to send, message FIN flag is sent.

- Parties close their sending side independently



- **Reset (RST)** is used when a party will no longer wait, or other reason requires an „abort"

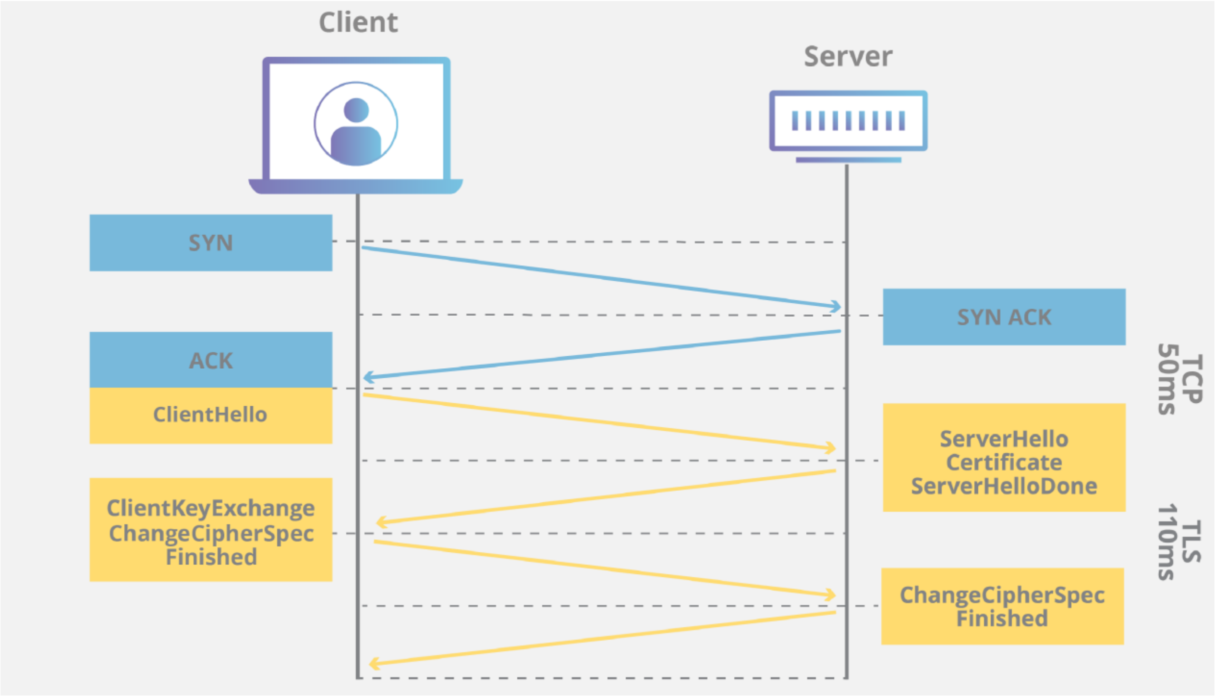
- TCP session is kept for a while in TIME_WAIT state to avoid stream collisions/interferences



TLS Protocol

- An additonal „sublayer“ within TCP
- TLS handshake after TCP SYN
  - Authentication (through certificates)
  - Cypher selection
  - Symmetric key exchange (K)
- All TLS payload bytes are encrypted with K
  - But TCP headers are not encrypted!

- Performance improvement: TLS Session reuse
  - Allows for a shorter, 2-step handshake
  - Works only between identical parties

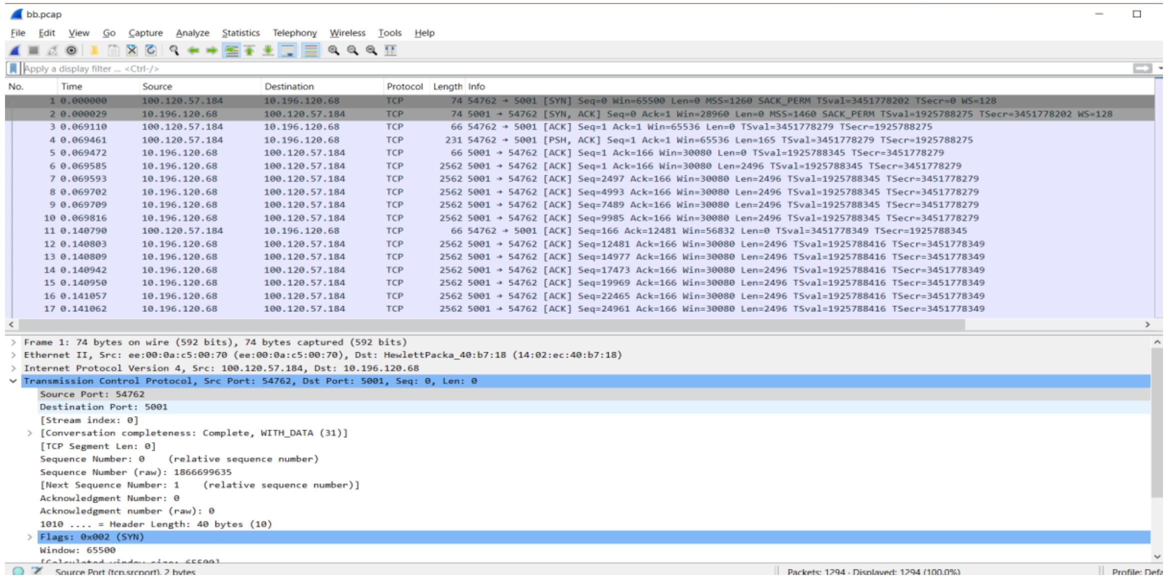


Src: [What happens in a TLS handshake? | SSL handshake | Cloudflare]

Useful tool: tcpdump & Wireshark

- **tcpdump** command line tool to capture and visualize packets
    - `tcpdump -i eth0 tcp port 5001`
  - ‚-i eth0' is the interface
  - „tcp port 5001" is a „BPF filter" expression processed in the kernel
  - The –w option saves packets to a „pcap" file:
    - `tcpdump -i eno1 tcp port 5001` **`-w saved_capture.pcap`**
  - Rudimentary display of packet headers and data

- **Wireshark is an iproved & GUI based tool.**
  - It can read pcaps
  - Also can capture pcaps from network
  - Extensive analysis capabilities for hundreds of protocols
  - Packets are listed in a table, and can be drilled down to finest detail.



Useful Network Performance Commands Speedtest.net, iperf3

- **Standard Linux tool for measuring network bandwidth and quality**
  - Supports TCP and UDP (and SCTP -> „Stream Control Transmission Protocol“)
  - Unidirectional measurements, but may be run in parallel.
  - Tool to be started in server mode first [iperf3 -s], then in client mode [iperf3-c server] at the other end
    - Public iperf3 servers are also available (although many are often down)
  - For UDP mode, bitrate is selectable

- **Speedtest.net (ookla)**: another popular tool for quick network bandwidth testing in various directions
  - Very user friendly, runs in web browser, even on smartphones
  - May be used to measure bandwidth to any part in the world (there are hundreds of ookla servers)

TCP or UDP? - Experiment

- Experiment:
  - UDP
    - Loses packets at larger packet rates (above 5-10 k/sec)
    - Very undeterministic
    - Top speed around 500-800 Mbps / stream
  - TCP
    - No loss
    - Top speed around 10 Gbps/stream
- Why?
  - Kernel is optimized for TCP
  - Userplane sender interface is easier: no need for timing, will automatically control the datarate.

Improving Network Performance – Key takeavays

- Use high-speed links e.g. (10 or 25Gps)
  - Or multiple of these bundled together – „network bonding"
- Make sure this interface speed is available on all network devices (switches, routers) on the path
- Use TCP when high throughput is required.
- Watch out for server-side throttling

Exercise

- Find and example program to write and read files (chat GPT?)
- Convert it to AF_UNIX communication
- Convert it to AF_INET UDP communication
- Convert it to AF_INET TCP communication
- Test all tools

That's it for today!