Networking Technologies and Applications

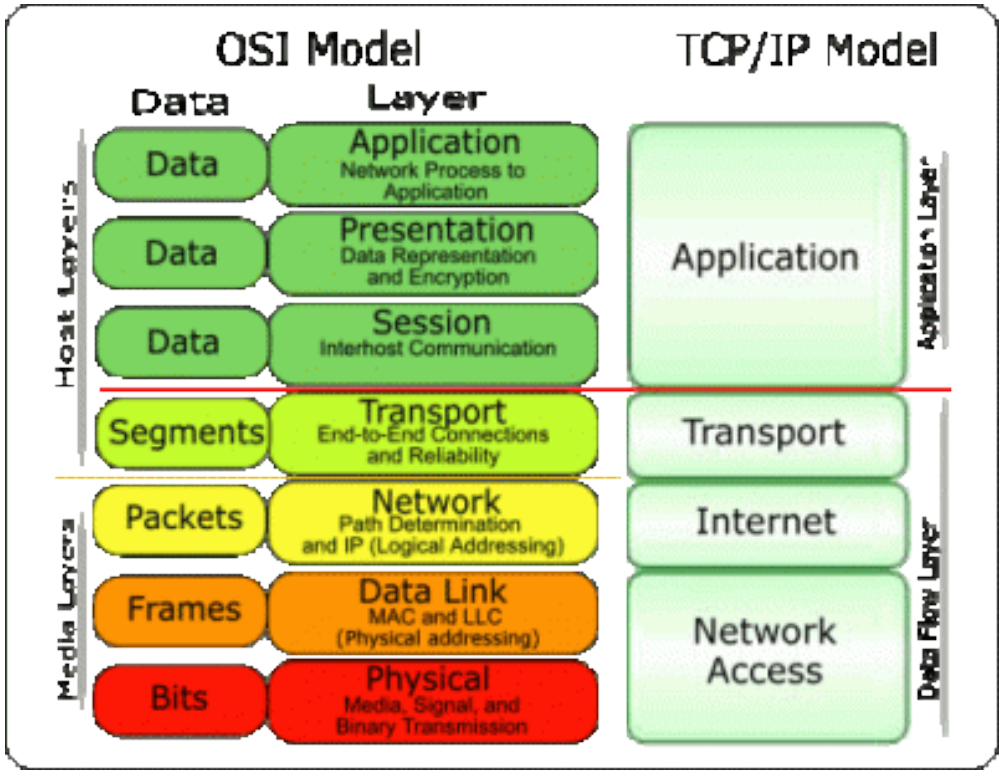
Rolland Vida
BME TMIT

November 28, 2017

Transport Protocols

- UDP – User Datagram Protocol
- TCP – Transport Control Protocol

- and many others…

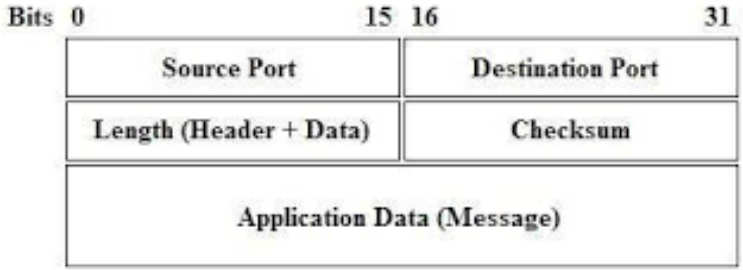


UDP

- One of the core transport protocols
  - Used by applications to send data (datagrams) between two end-hosts on the IP network
- **Connectionless transmission** – no preset channel, data path
  - No handshaking dialog between sender and receiver, unreliable transmission
  - Only data integrity is verified (checksum), not the delivery of the datagram
- No guarantee of delivery or ordering
- Used for ….
  - Time-sensitive or real-time applications, where there is no possibility for waiting for retransmissions
  - Applications that are based on simple and fast message exchanges
    - DHCP, DNS, RTP, RTCP, etc.

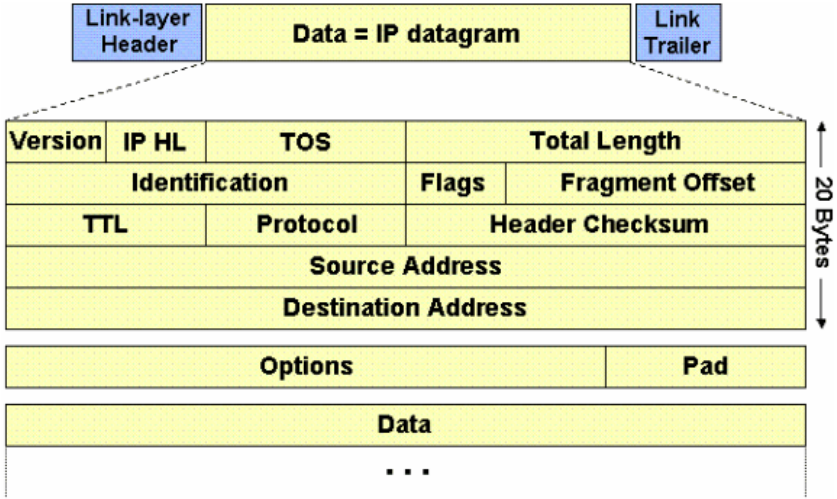
UDP datagram structure

- **Source port** (16 bits)
  - Identifies the sender application
- **Destination port** (16 bits)
  - Identifies the receiver application
- **UDP length** (16 bits)
  - Length of the entire datagram (header + data) in bytes
    - Minimum value: 8 (no data)
    - Theoretical maximum size data: 65507 bytes
- **Checksum** (16 bits)
  - Calculated for the header and data together
  - If checksum wrong, packet silently discarded
    - No error message



Fragmentation of UDP datagrams

- No fragmentation allowed for UDP
  - Not sure that all fragments will arrive
  - The application has to make sure that the correct datagram size is used



Flags on 3 bits

Value	Bit 0 Reserved	Bit 1 DF	Bit 2 MF
0	0	May	Last
1	0	Do not	More

Fragmentation example

Original IP Datagram

Sequence	Identifier	Total Length	DF May / Don't	MF Last / More	Fragment Offset
0	345	5140	0	0	0

IP Fragments (Ethernet)

Sequence	Identifier	Total Length	DF May / Don't	MF Last / More	Fragment Offset
0-0	345	1500	0	1	0
0-1	345	1500	0	1	185
0-2	345	1500	0	1	370
0-3	345	700	0	0	555

Fragment offset in units of 8 bytes

- 185 x 8 + 20 (IP header) = 1500 bytes

TCP – Transmission Control Protocol

- Provides a reliable end-to-end connection between two applications
  - Connection-oriented data stream service
  - Flow control algorithm
- Before starting data transmission, the TCP connection has to be built
- Cannot be used for broadcasting and multicasting

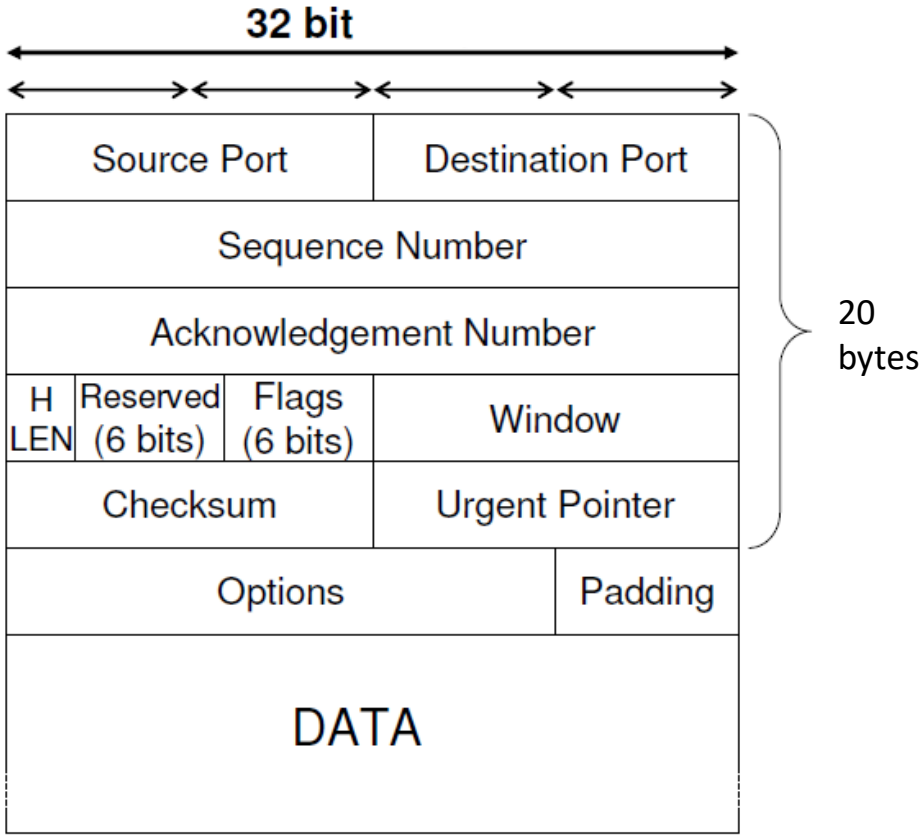
- **TCP segment** encapsulated into an IP packet
- **TCP socket** – combination of the IP address and TCP port number

TCP acknowledgments

- Full duplex, bi-directional data connection
- No selective ACK
  - ACK means that all the bytes until now (but not including the sent packet number) were received correctly
- No negative ACK

TCP header

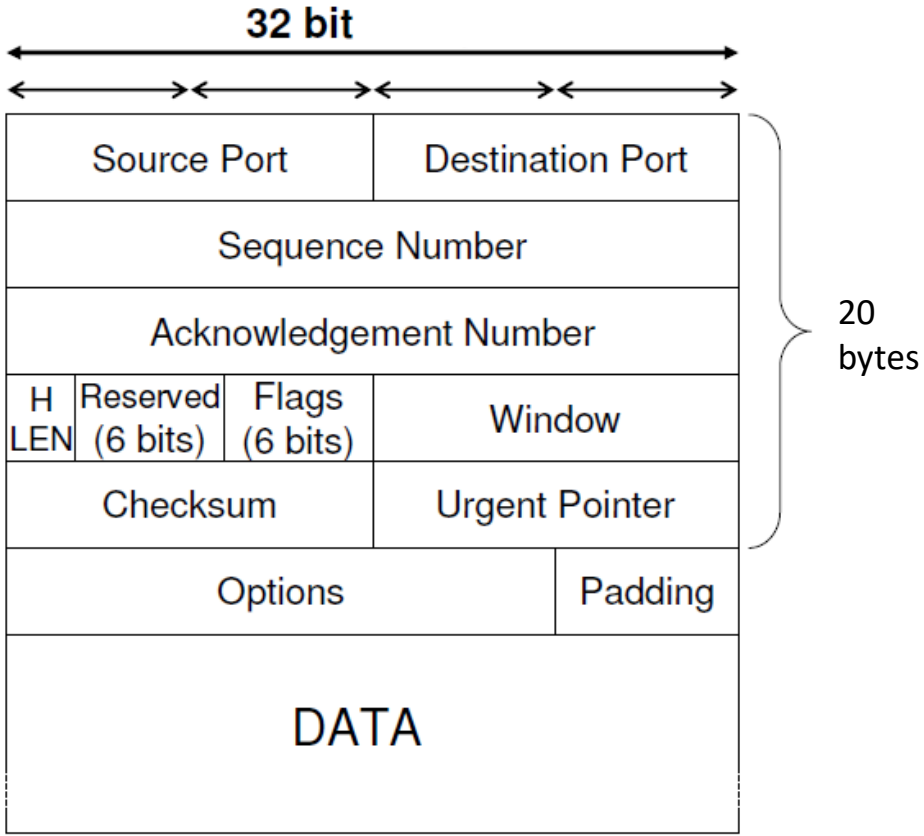
- Sequence number
  - The number of the packet in the stream
- Ack number
  - The sequence number the reciever expects to receive next
- HLEN – Header length
- Reserved
  - For future use



TCP header

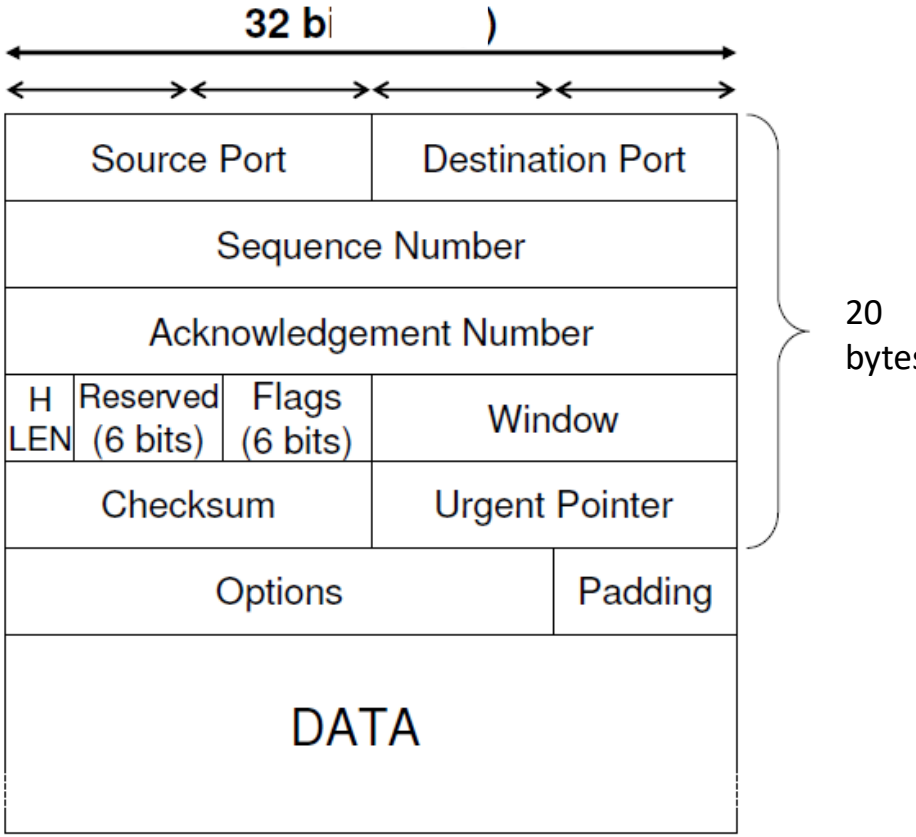
Flags

- 6 flags that regulate the behavior of the TCP segment
  - 1. Urgent (URG)
  - 2. Acknowledgement (ACK)
  - 3. Push (PSH)
  - 4. Reset connection (RST)
  - 5. Synchronous (SYN)
  - 6. Finish (FIN)



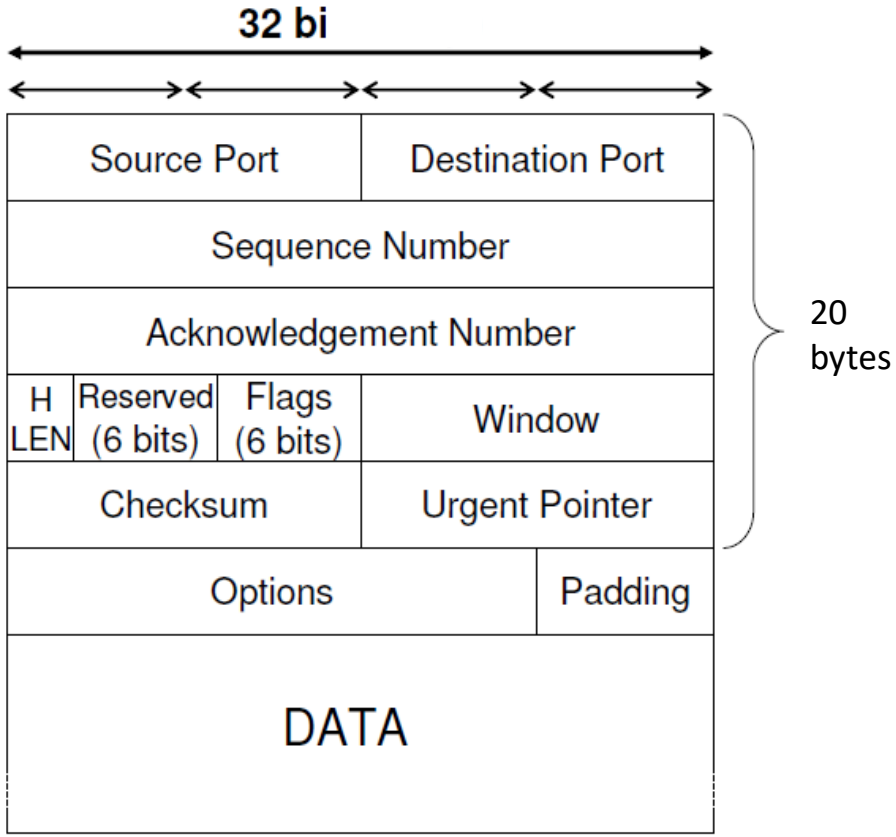
TCP header

- **Urgent flag (URG)**
  - End-points can send a notification that the data stream contains data that should be urgently handled
- **Acknowledgement flag (ACK)**
  - Used to indicate that data has been successfully received
- **Push flag (PSH)**
  - Often set at the end of a block of data, signaling the receiver to process the block of data



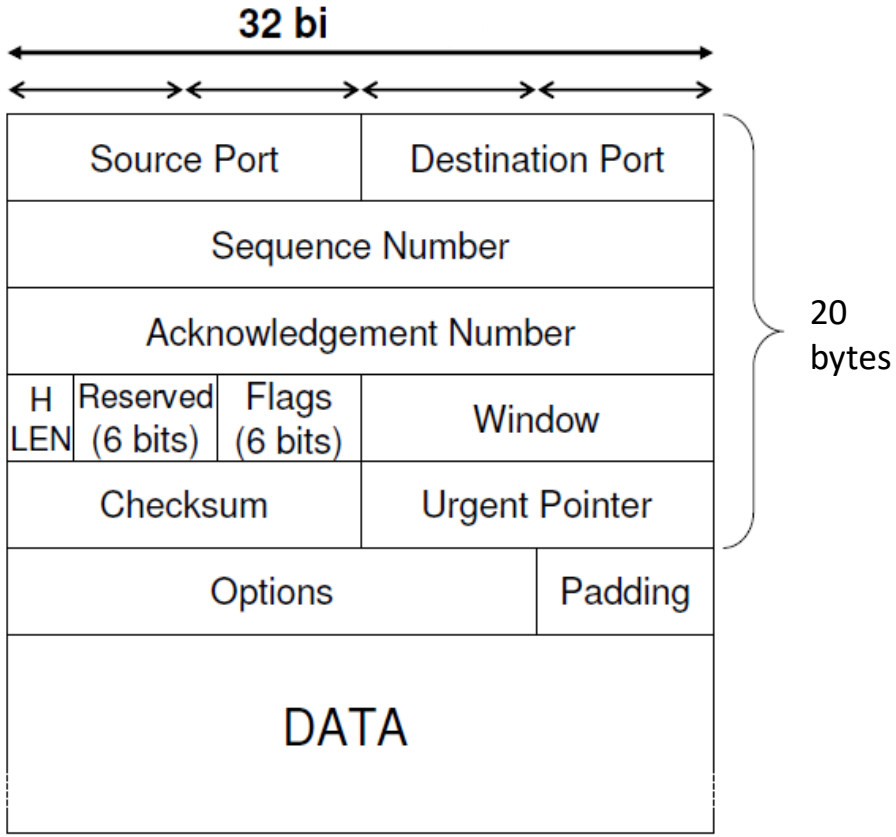
TCP header

- **Reset flag (RST)**
  - used to inform the receiver that the sender has shut this connection down
- **Synchronous flag (SYN)**
  - used at the start of the TCP handshake to establish the connection
- **Finish flag (FIN)**
  - Used to gracefully tear connections down
  - Each side of the connection sends a FIN, followed by an ACK, then the connection is finished



TCP header

- Window (16 bits):
  - Indicates how many bytes can still be fit in the buffer of the receiver
- Checksum (16 bits):
  - To check the integrity of the TCP header
- Urgent Pointer (16 bits):
  - If the segment contains urgent data (URG flag set), it tells where the urgent data starts in the payload
- Options
  - The most often used option is MSS - maximum segment size
  - Provides the maximum segment size the receiver would like to receive

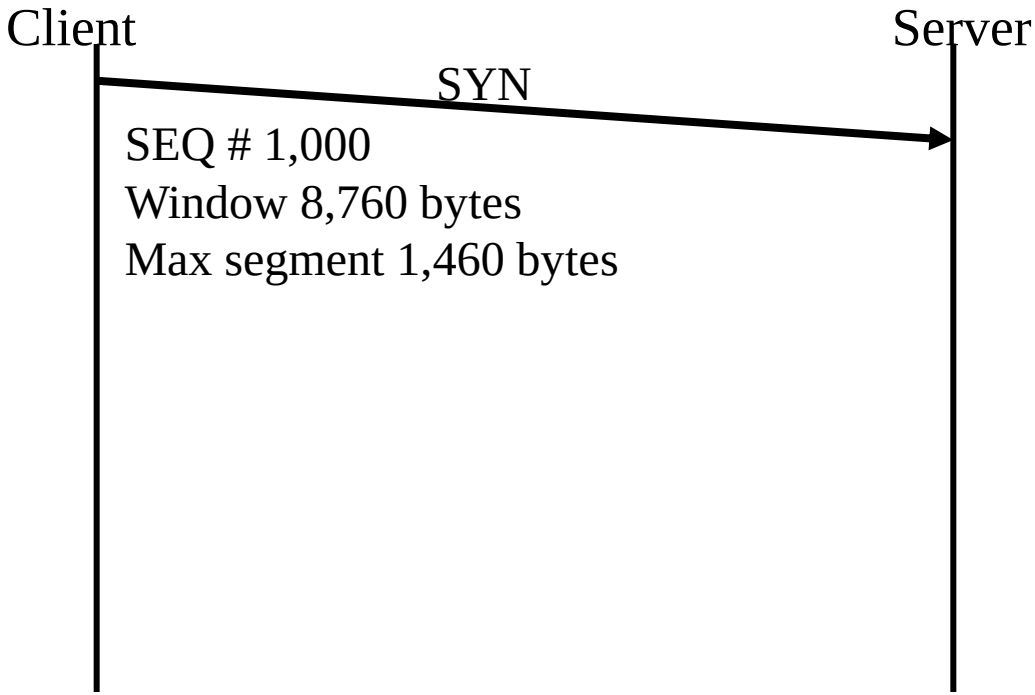


Building a TCP connection

- The TCP protocols handles the following steps:
  - Building the connection
  - Advertising the window size and the Maximum Segment Size
  - Sending the data
  - Sending acknowledgements
  - Tearing down the connection at the end

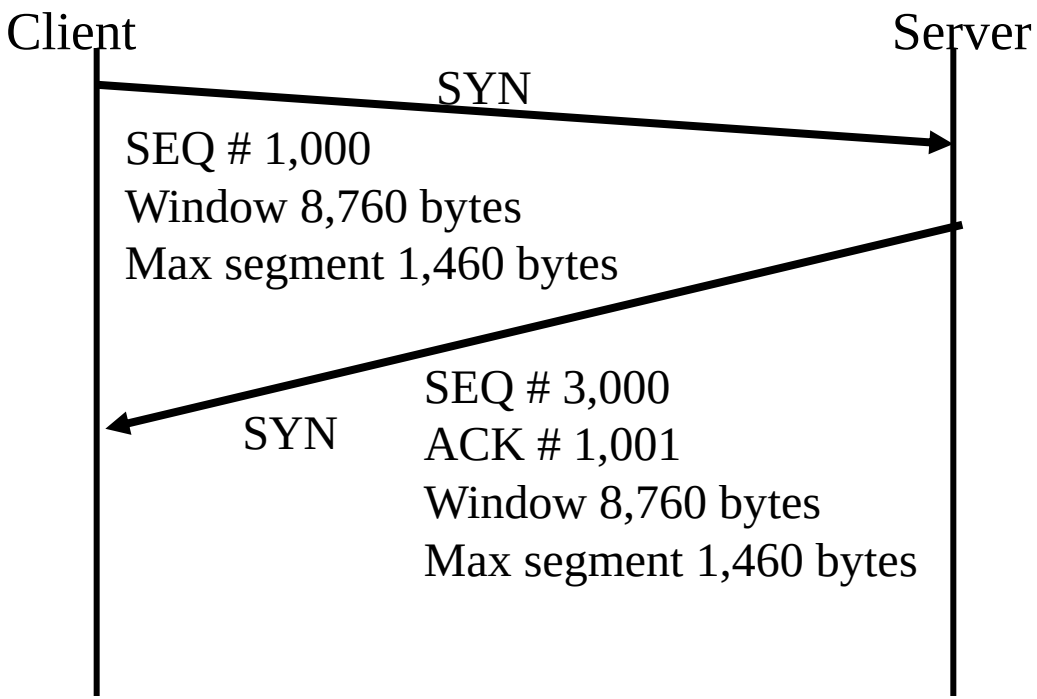
Building the connection

1. Initiating host - client
   - Sending a SYN segment
     - Server port number where I want to connect
   - ISN – initial seq. num.
   - Advertising its own window size and MSS
   - If no MSS, then default is 536 bytes



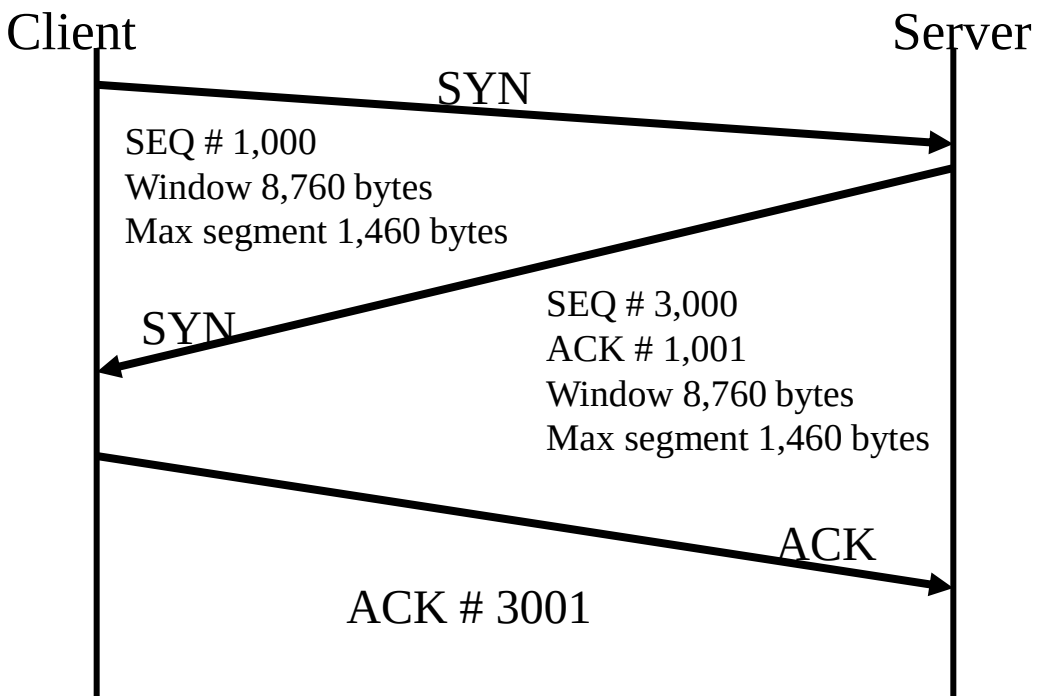
Building the connection 2

2. Answer from the server
   - SYN segment contains
     - Server ISN
     - Ack on the client segment
       - Expect 1001
   - Its own window size and MSS



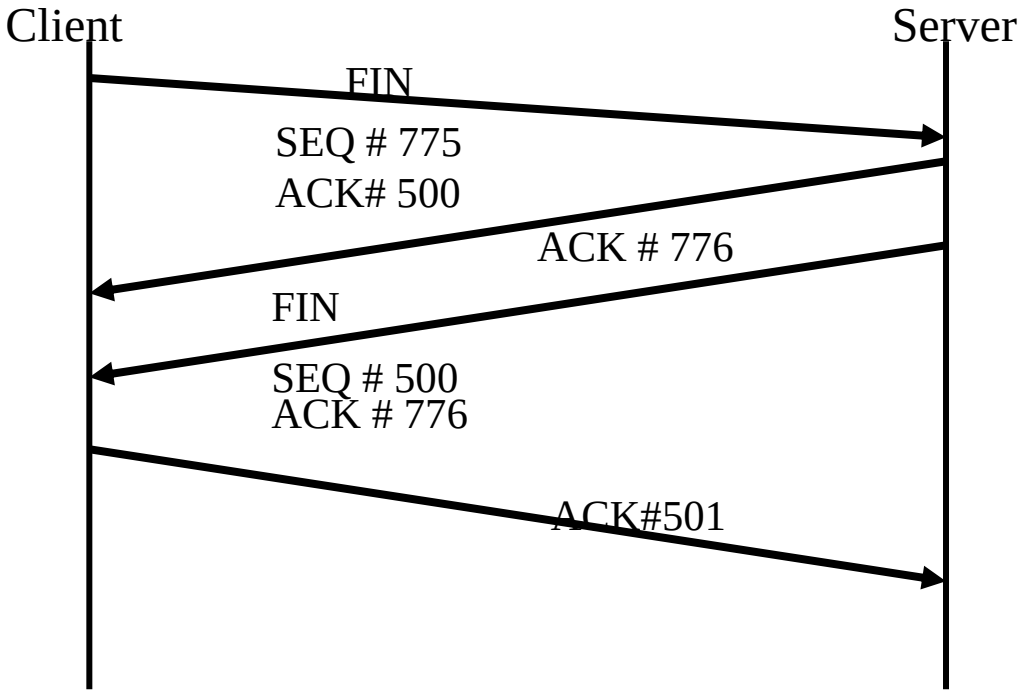
Building the connection 3

3. The client sends ACK
   - Mandatory for the client to send ACK on the SYN segement of the server



Ending a TCP connection

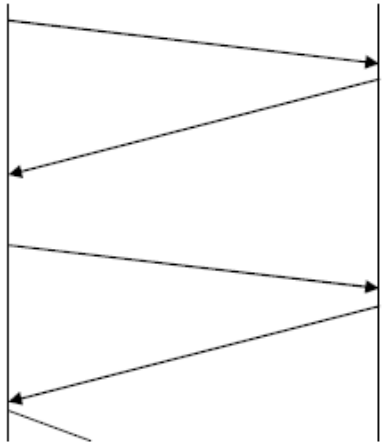
- 4 segments at the normal ending of connection
- Each end-point closes the connection indepenendetly from the other
- Receiving a FIN segment
  - The TCP has to announce the application that the other host has closed the connection



Simple flow control

- **Stop-and-wait protocol**
  - Send a data segment
  - Wait for an ack
  - If ack arrives, send next data segment
  - If no ack until timer expires – resend the data segment and wait for an ack

- Properties
  - Ack for each individual segment
  - Very slow if large distances



TCP flow control

- Sliding window

- Ack is not expected for each individual segment
  - There can be many unacknowledged segments „on the road”
  - The same segment might be sent several times
  - Acknowledgements might arrive in a burst

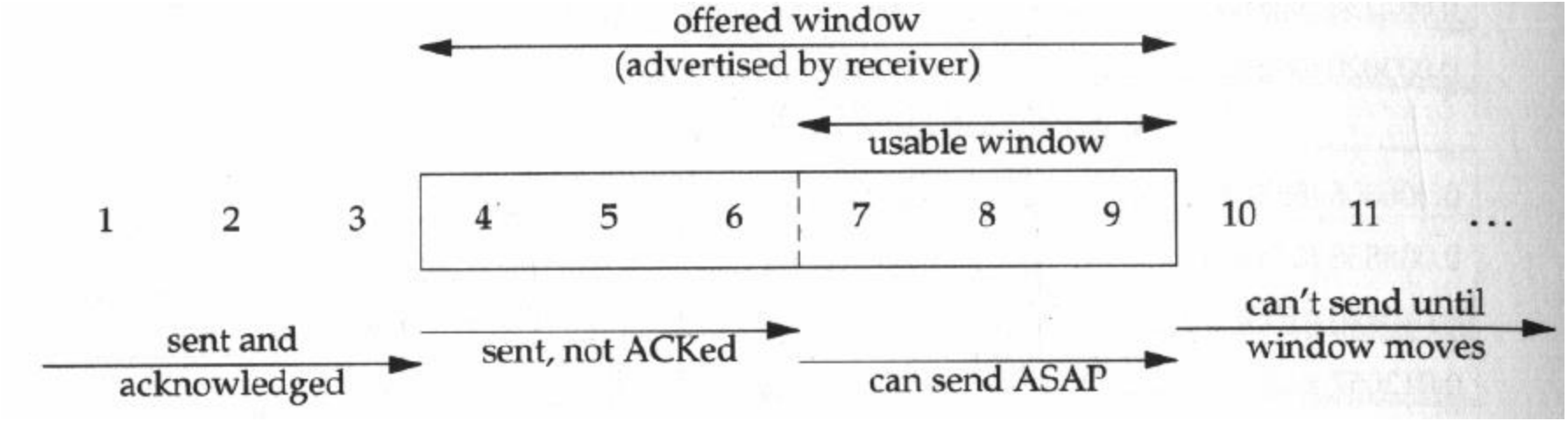
- Faster data transfer
  - If the number of segments „on the road” is somehow controlled

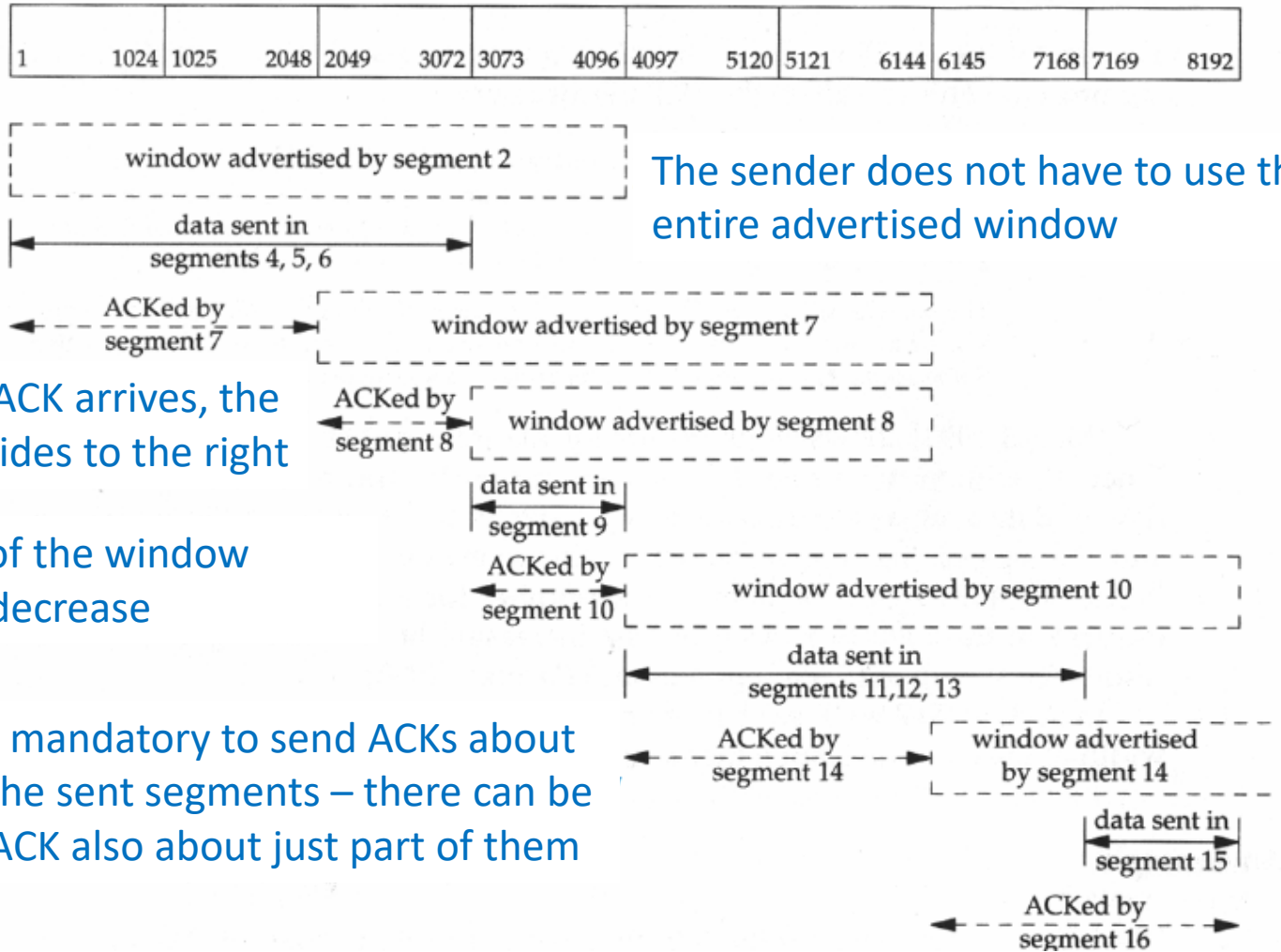
Fast sender, slow receiver

- Sender sends according to the advertised window size
  - To fill up the receiver's buffer
- Sender waits for the ack
- The receiver is slow, cannot forward the segments to the application – buffer remains full
  - The receiver sets in its ack the „advertised window size" to 0
- The sender does not send any more segments

- Later, the sender should be triggered to start sending again
  - If the buffer of the receiver gets empty, it sends a Window Update message
    - A new ack, to the same segment, but with a new adv. Window size

Sliding window





The sender does not have to use the entire advertised window

When an ACK arrives, the window slides to the right

The size of the window can also decrease

Not mandatory to send ACKs about all the sent segments – there can be an ACK also about just part of them