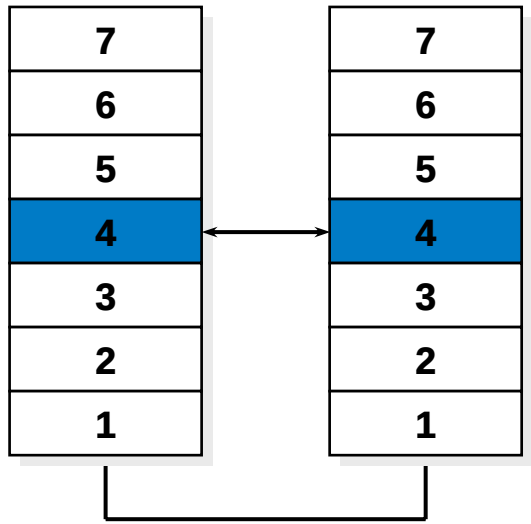
TI III: Operating & Communication Systems

Transport Layer

Protocol mechanisms

TCP, UDP

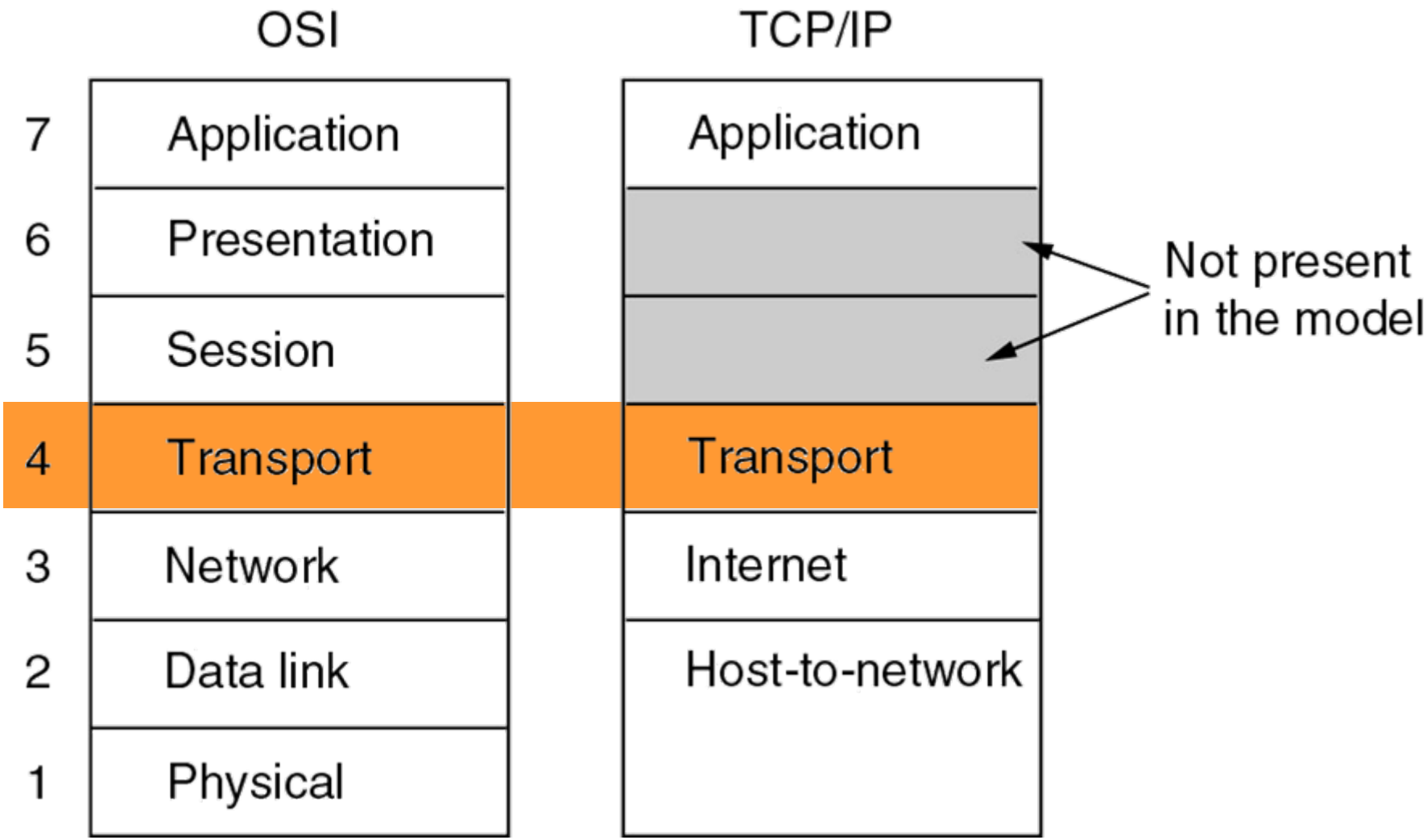
Addressing, ports



Content (2)

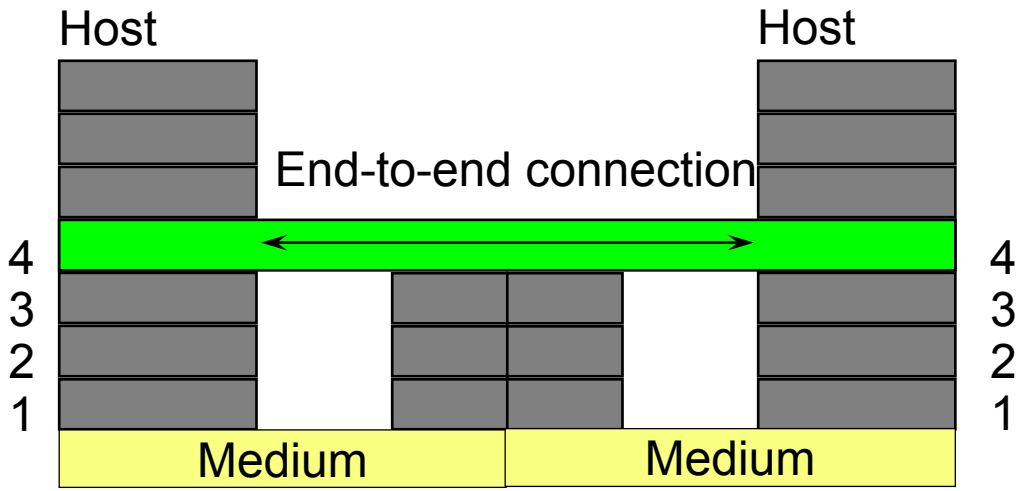
8. Networked Computer & the Internet
   - Sockets
   - Internet
   - Layers, Protocols
9. Host-to-Network I
   - Physical Layer
   - Media, Signals
   - Modems
10. Host-to-Network II
    - Data Link Layer
    - Framing, flow control
    - Error detection/ correction
    - PPP
11. Host-to-Network III
    - Topologies
    - Medium Access
    - Local Area Networks
      - Ethernet, WLAN
12. Internetworking
    - Switches, routers
    - Routing
    - Internet protocol
    - Addressing
13. **Transport Layer**
    - **Protocol mechanisms**
    - **TCP, UDP**
    - **Addressing, Ports**
14. Application Support
15. Programming
16. Example

Data Link Layer



Tasks of the transport layer

- End-to-end connection
  - From process to process, not node-to-node



  - Insulation of higher layers from technology, the structure and the impairments of lower layers
  - Transparent transmission of user data
  - Support of quality-of-service (not in the classical Internet)
  - Independent addressing of users/processes (no matter what layer 3 does – however: socket [IP-address + port] in the Internet)

Overview

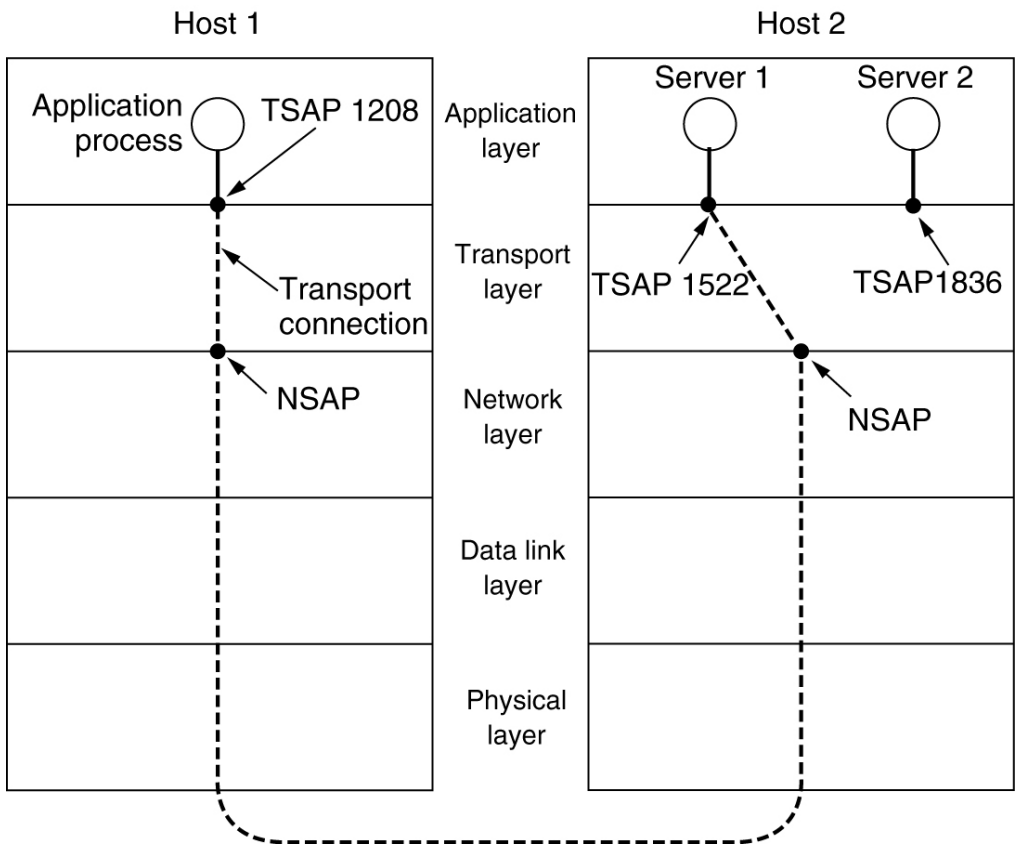
- ***Services & addressing***
- Connection setup
- Connection release
- Flow control
- Timer and timeouts
- TCP summary
- UDP
- Performance issues

Transport protocols – Services to upper layer

- Connection-less and connection-oriented transport service
  - Connection management necessary as auxiliary service – setup and teardown
- Reliable or unreliable transport
  - In-order, all packets
- Be a good citizen in the network – perform congestion control
- Allow several transport endpoints in a single host
  - Demultiplexing
- Support different interaction models
  - Byte stream, messages, remote procedure invocation

Addressing

- Provide multiple service access points to multiplex several applications
  - SAPs can identify connections or data flows
- E.g., “port numbers”
  - Dynamically allocated
  - Predefined for “well-known services” – port 80 for Web server



TCP – some well-known ports

- Many applications choose TCP or UDP as transport protocol, however, the right port must be used to communicate with the correct application on server side
  - 13: day time
  - 20: FTP data
  - 25: SMTP (Simple Mail Transfer Protocol)
  - 53: DNS (Domain Name Server)
  - 80: HTTP (Hyper Text Transfer Protocol)
  - 119: NNTP (Network News Transfer Protocol)

```
> telnet walapai 13
Trying 129.13.3.121...
Connected to walapai.
Escape character is '^]'.
Sun Jan  21 16:57:19 2007
Connection closed by foreign host
```

```
> telnet mailhost 25
Trying 129.13.3.161...
Connected to mailhost .
Escape character is '^]'.
220 mailhost ESMTP Sendmail 8.8.5/8.8.5; Sun,
21 Jan 2007 17:02:51 +0200
HELP
214-This is Sendmail version 8.8.5
214-Topics:
214-    HELO    EHLO    MAIL    RCPT    DATA
214-    RSET    NOOP    QUIT    HELP    VRFY
214-    EXPN    VERB    ETRN    DSN
214-For more info use "HELP <topic>".
...
214 End of HELP info
```

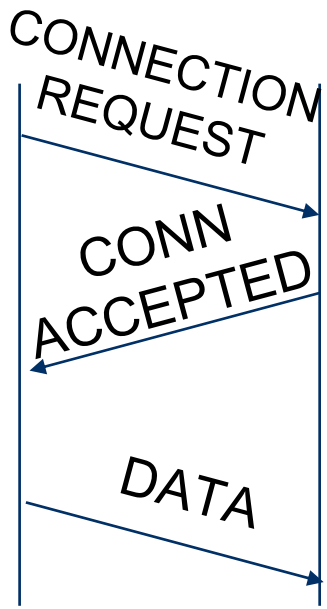
Overview

- Services & addressing
- ***Connection setup***
- Connection release
- Flow control
- Timer and timeouts
- TCP summary
- UDP
- Performance issues

Connection establishment

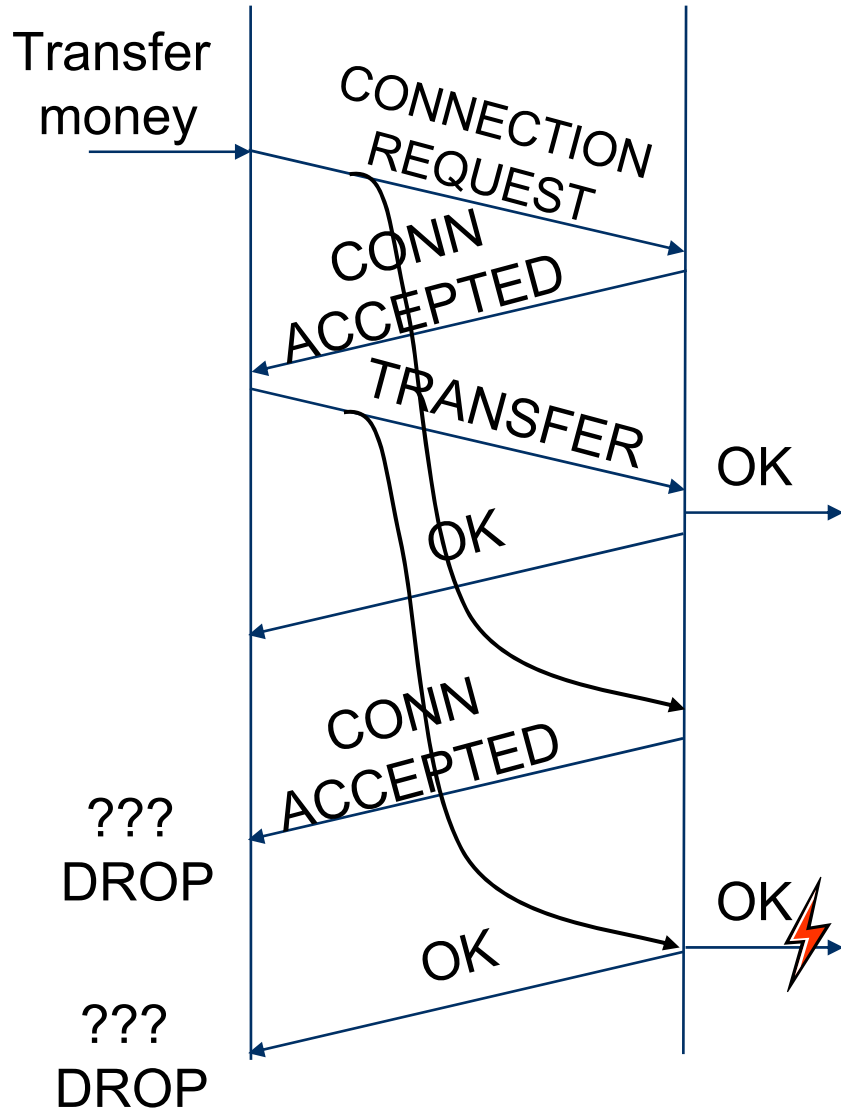
- How to establish a join context, a connection, between sender and receiver?
  - Note: only relevant in end-system, network layer assumed to be connection-less (Internet Protocol)

- Naïve solution:
  - Sender sends a CONNECTION REQUEST
  - Receiver answers by a CONNECTION ACCEPTED
  - Sender proceeds once that message is received



Failure of naïve solution

- Naïve solution fails in realistic networks
  - Where packets can be lost, stored/reordered, and duplicated
- Example failure scenario
  - All packets are duplicated and delayed
  - Due to congestion, errors, ... and re-routing, e.g.
  - In a banking application, e.g.
  - Result: two independent transactions performed, where only one was intended
- Problem are *delayed duplicates*

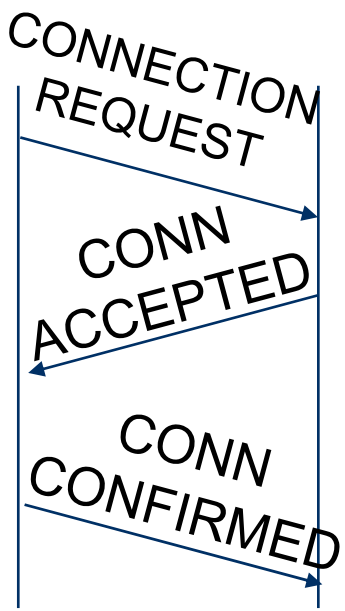


Adding an additional handshake

- Idea: the sender has to re-confirm to the receiver that it actually wants to set up this connection

➔ Add a third message to the connection setup phase

- ***Three-way handshake***
- This third message can already carry data



Connection setup – further issues

- Further problems due to crashing nodes, need some memory
- Sequence numbers negotiated here used for duplicate identification of connection setup messages and acknowledgement of the following data packets

- Terminology for TCP
  - Connection setup – SYN (synchronize) packet
  - Connection accepted – SYN/ACK packet (because both the previous sequence number is acknowledged and a new sequence number from the receiver is proposed)
  - Connection confirmation – ACK packet (combined with DATA, as a rule)
  - Problem: SYN attack – flooding with nonsense SYN packets

Connection identification in TCP

- A TCP connection is setup
  - Between a single sender and a single receiver
  - More precisely, between application processes running on these systems
  - TCP can multiplex several such connections over the network layer, using the port numbers as Transport SAP identifiers

- A TCP connection is thus identified by a four tuple:
  - Known as socket pair

(Source Port, Source IP Address,
Destination Port, Destination IP Address)

Overview

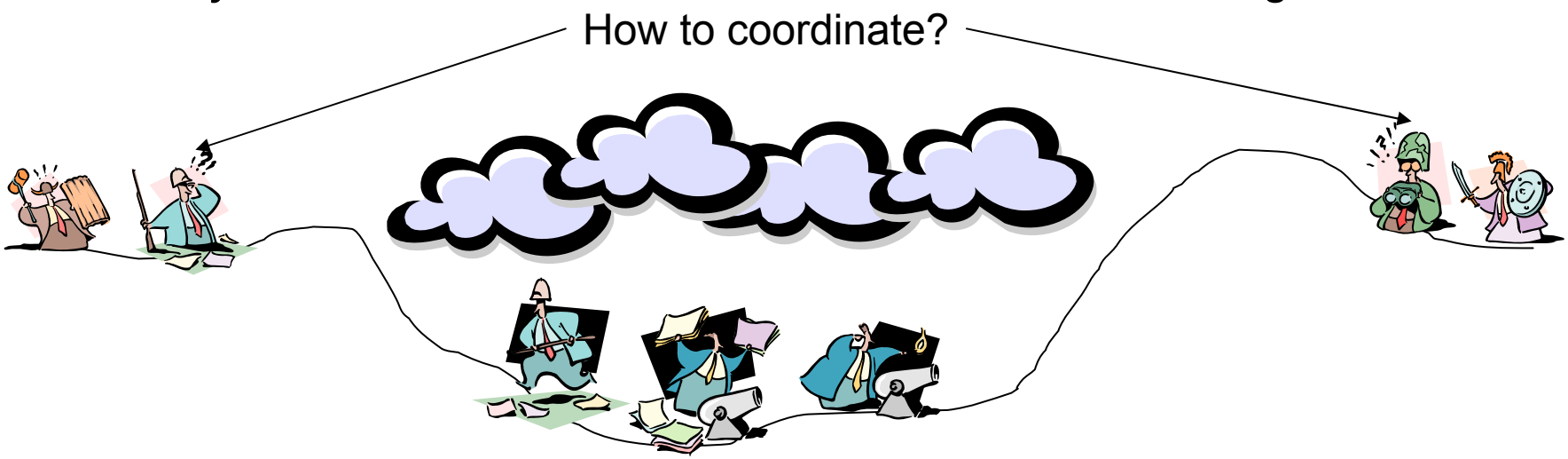
- Services & addressing
- Connection setup
- ***Connection release***
- Flow control
- Timer and timeouts
- TCP summary
- UDP
- Performance issues

Connection release

- Once connection context between two peers is established, releasing a connection should be easy
  - Goal: Only release connection when both peers have agreed that they have received all data and have nothing more to say
  - I.e., both sides must have invoked a “Close”-like service primitive
- It fact, it is impossible
  - Problem: How to be sure that the other peer knows that you know that it knows that you know … that all data have been transmitted and that the connection can now safely be terminated?
- Analogy: Two army problem

Two army problem

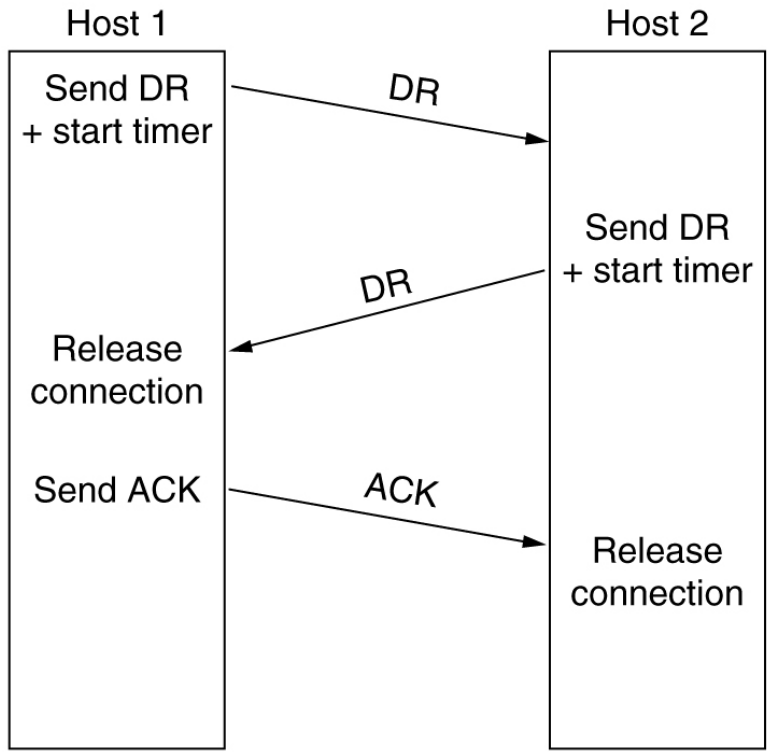
- Coordinated attack
  - Two armies form up for an attack against each other
  - One army is split into two parts that have to attack together – alone they will lose
  - Commanders of the parts communicate via messengers who can be captured
- Which rules shall the commanders use to agree on an attack date?
- Provably unsolvable if the network can loose messages



Connection release in practice

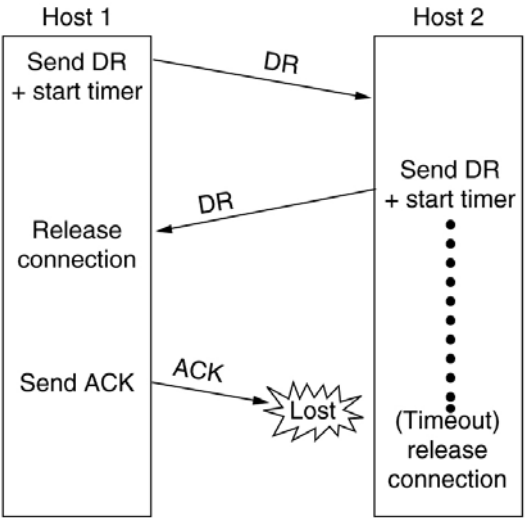
- Two army problem equivalent to connection release
- But: when releasing a connection, bigger risks can be taken

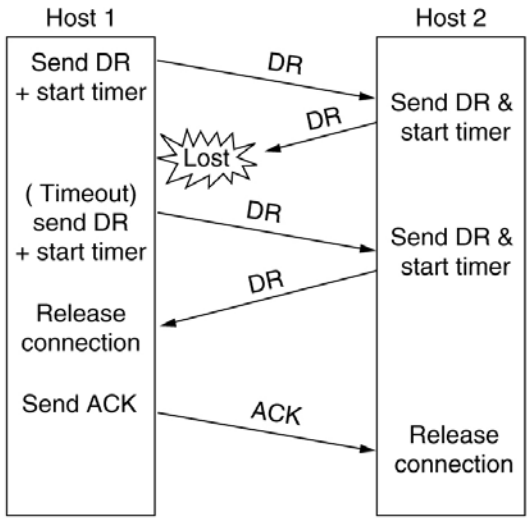
- Usual approach: Three-way handshake again
  - Send disconnect request (DR), set timer,
    wait for DR from peer,
    acknowledge it

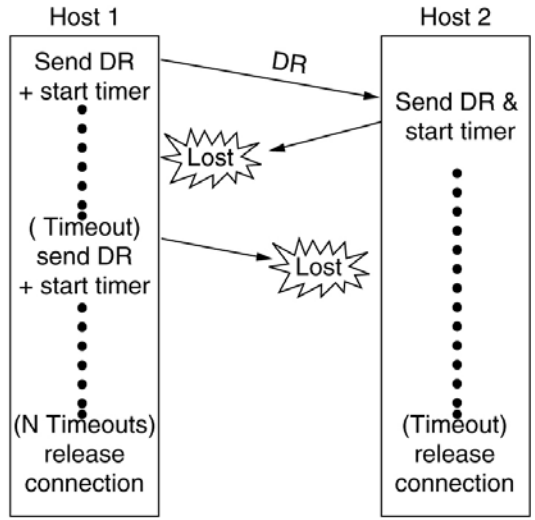


Problem cases for connection release with 3WHS

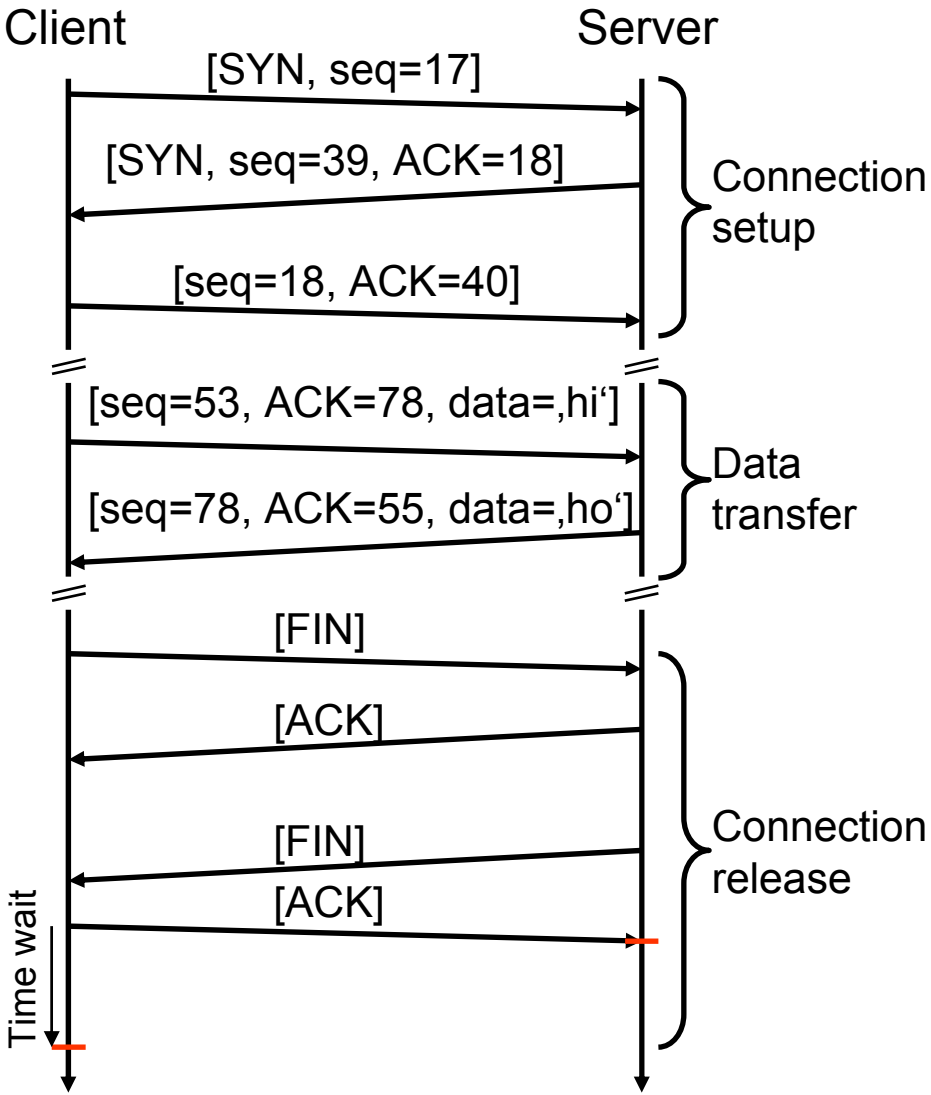
- Lost ACK solved by (optimistic) timer in Host 2
- Lost second DR solved by retransmission of first DR
- Timer solves (optimistically) case when 2nd DR and ACK are lost





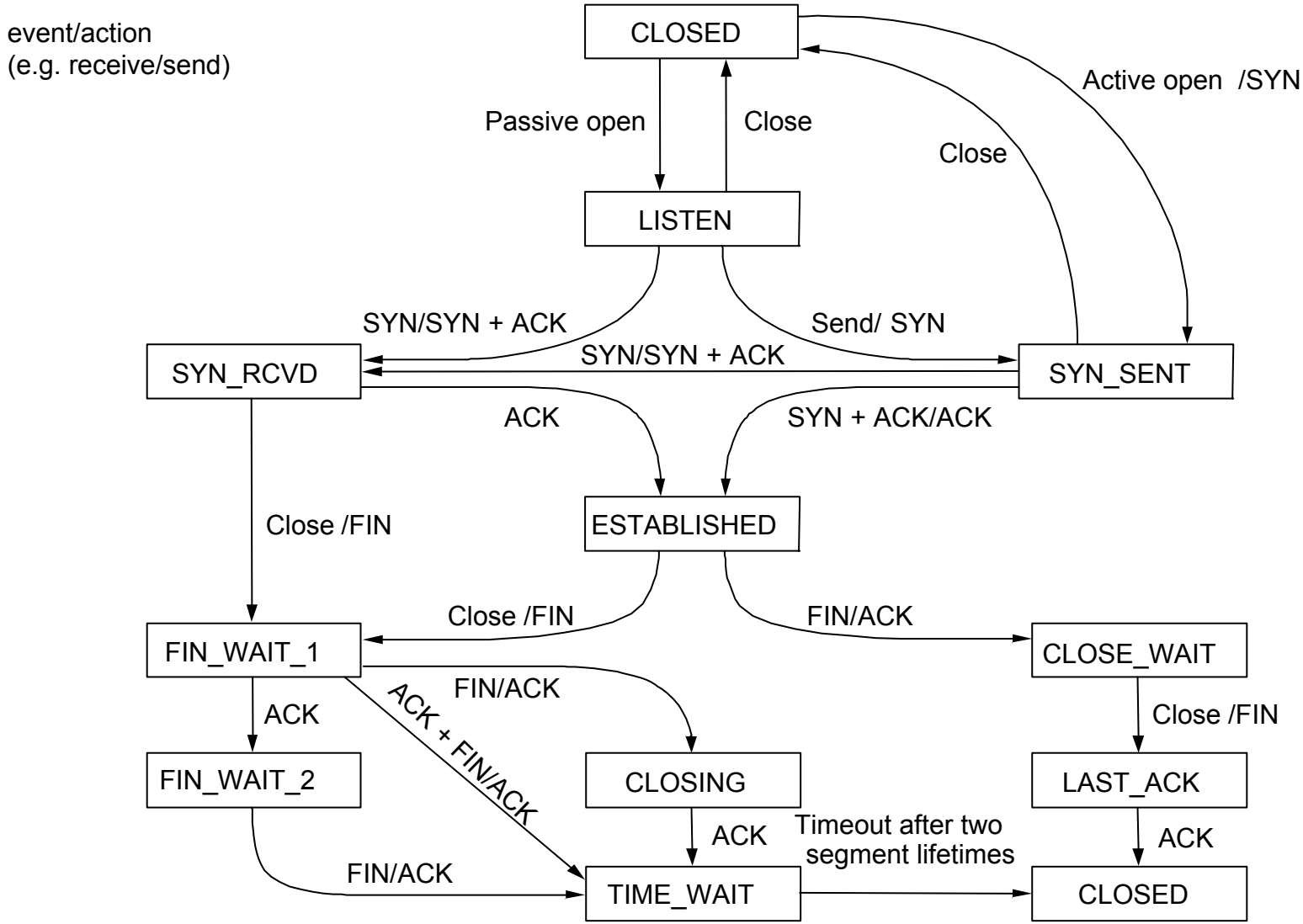


Example: TCP setup, transmission, release



- Connection setup
  - 3-way-handshake
  - Negotiation of window size, sequence numbers
- Data transfer
  - Piggybacked acknowledgement
- Connection release
  - Confirmed
  - Resources on client-side released after time-wait (frozen reference), e.g. 30 s, 1 min, 2 min – determines performance!

TCP state transition diagram



Overview

- Services & addressing
- Connection setup
- Connection release
- ***Flow control***
- Timer and timeouts
- TCP summary
- UDP
- Performance issues

Flow control

- Recall: Flow control = prevent a fast sender from overrunning a slow receiver
  - Similar issue in link and transport layer

- In transport layer more complicated
  - Many connections, need to adapt the amount of buffer per connection dynamically (instead of just simply allocating a fixed amount of buffer space per outgoing link)
  - Transport layer PDUs can differ widely in size, unlike link layer frames
  - Network's packet buffering capability clouds the picture

Flow control – buffer allocation

- In order to support outstanding packets, the sender either
  - Has to rely on the receiver to process packets as they come in (packets must not be reordered) – unrealistic, or
  - Has to assume that the receiver has sufficient buffer space available
- The more buffer, the more outstanding packets
  - Necessary to obtain highly efficient transmission, recall bandwidth-delay product!
- How does sender have buffer assurance?
  - Sender can request buffer space
  - Receiver can tell sender: “I have X buffers available at the moment”
    - For sliding window protocols: Influence size of sender’s send window

Flow control – permits and acknowledgements

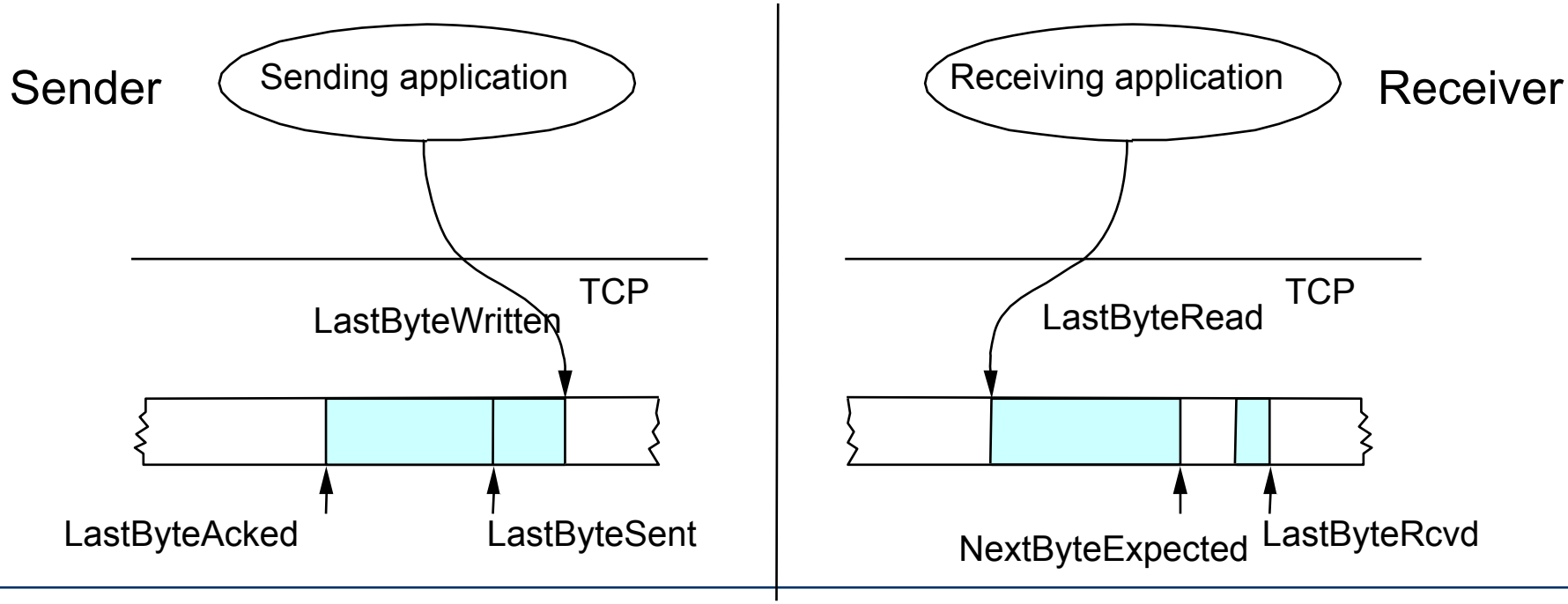
- Distinguish
  - Permits (“Receiver has buffer space, go ahead and send more data”)
  - Acknowledgements (“Receiver has received certain packets”)
- Should be separated in real-world protocols!

- Can be combined with dynamically changing buffer space at the receiver
  - Due to, e.g., different speed with which the application actually retrieves received data from the transport layer

  - Example: TCP

Send and receive buffers in TCP

- TCP maintains buffer at
  - Sender, to assist in error control
  - Receiver, to store packets not yet retrieved by application or received out of order
    - Old TCP implementations used GoBack-N, discarded out-of-order packets



TCP flow control: Advertised window

- In TCP, receiver can ***advertise*** size of its receiving buffer
  - Buffer space occupied:
    (NextByteExpected-1) – LastByteRead
  - Maximum buffer space available: MaxRcvdBuffer
  - Advertised buffer space ***(Advertised window):***
    MaxRcvdBuffer – ((NextByteExpected-1) – LastByteRead)

- Recall: Advertised window limits the amount of data that a sender will inject into the network
  - TCP sender ensures that
    LastByteSent – LastByteAcked ≤ AdvertisedWindow
  - Equvialently:
    EffectiveWindow = AdvertisedWindow –
    (LastByteSent – LastByteAcked)

Overview

- Services & addressing
- Connection setup
- Connection release
- Flow control
- ***Timer and timeouts***
- TCP summary
- UDP
- Performance issues

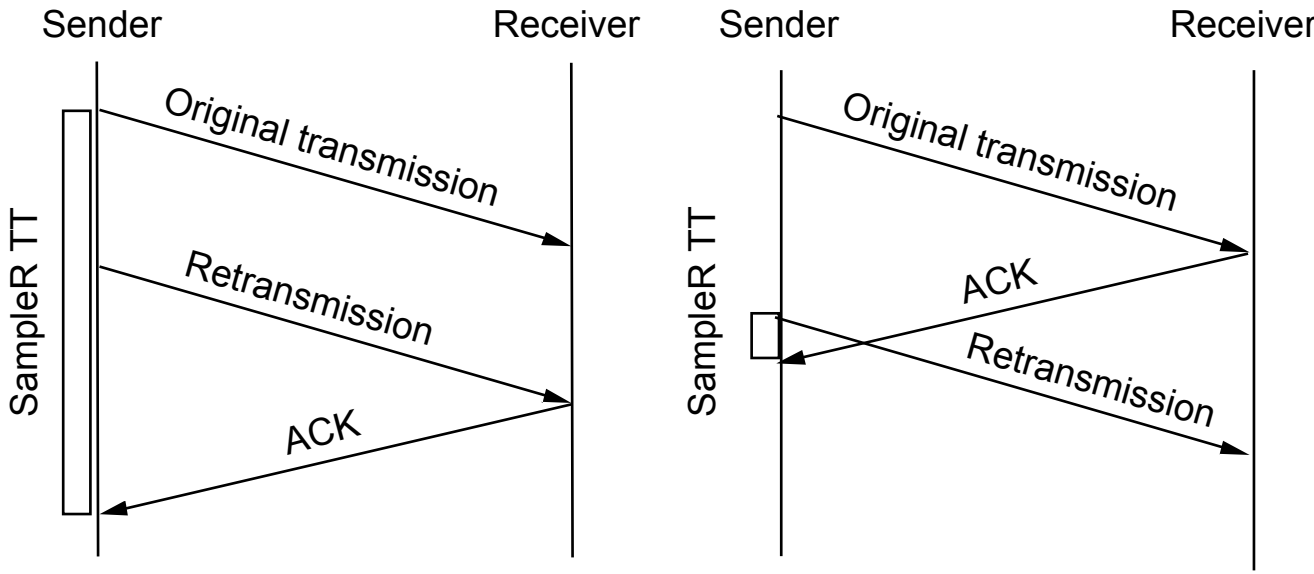
Timeout computations

- Timeouts necessary to protect against lost packets
- Timeouts should reflect round-trip time between sender and receiver
  - Problem: RTTs can be highly variable, range over several orders of magnitude
  - Has to be adapted dynamically

- Simple approach: Keep a running average of RTTs
  - Computed by an autoregressive model:
    $EstimatedRTT_n = \alpha\ EstimatedRTT_{n-1} + (1-\alpha)\ RTTSample_n$
  - Timeout = 2 EstimatedRTT
    as a conservative choice
  - Parameter α smoothes estimation (α = 0.8 ... 0.9, typically)

Problem: ACKs do not refer to a transmission!

- Simple algorithm described above cannot obtain correct RTT samples if packets have been retransmitted
  - ACKs refer to data/sequence numbers, not to individual packets!
- Two examples:



- Solution: Do not take RTT samples for retransmitted packets (Karn/Partridge algorithm), more optimizations exist (e.g., Jacobsen/Karels algorithm for RTT estimation)

Timer and packet loss

- What happens after a packet loss?
  - Basic idea: After packet loss detected by timeout, use successively larger timeout values
  - ***Exponential backoff:*** Double timeout value for each additional retransmission
- The estimation of RTT and “basic” timeout value is performed as described before
- The multiplicative factor for exponential backoff is reset upon ACK arrival

Overview

- Services & addressing
- Connection setup
- Connection release
- Flow control
- Timer and timeouts
- ***TCP summary***
- UDP
- Performance issues

TCP – Summary

- TCP consists of
  - Reliable byte stream – error control via GoBack-N or Selective Repeat (depending on version)
  - Congestion control – window-based, AIMD, slow start, congestion threshold (see literature)
  - Flow control – advertised receiver window
  - Connection management – three-way handshake for setup and teardown

$$BW \leq \frac{0.93 * MSS}{RTT * \sqrt{p}}$$

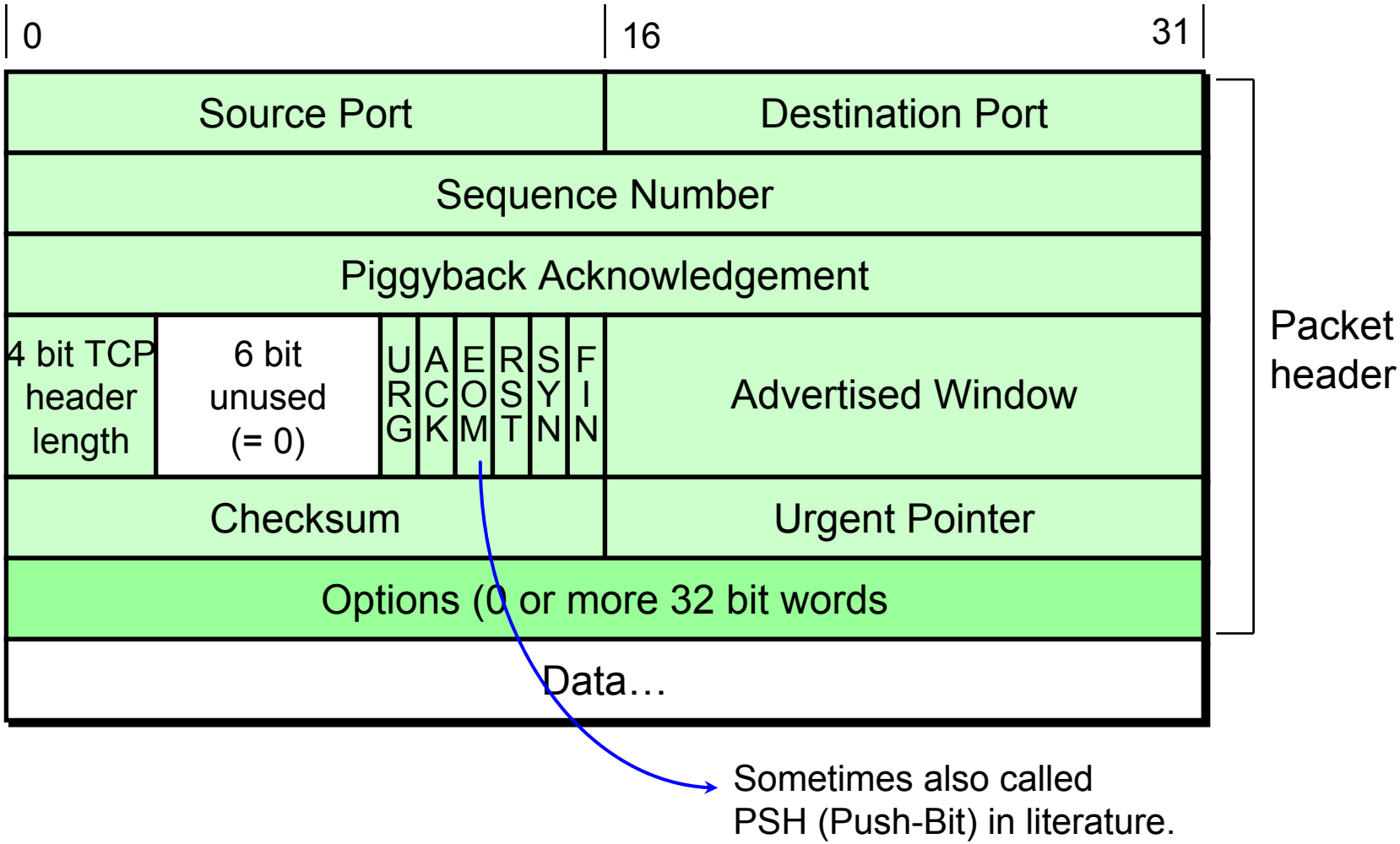
- max. TCP **B**and**W**idth
- **M**ax. **S**egment **S**ize
- **R**ound **T**rip **T**ime
- loss **p**robability

- TCP is perhaps the single most complicated and subtle protocol in the internet
  - Many little details and extensions are not discussed here
  - Interaction of TCP with other layers is more complicated than it looks (e.g., wireless) because of hidden, implicit assumptions

TCP fairness & TCP friendliness

- TCP attempts to
  - Adjust dynamically to the available bandwidth
  - Fairly share this bandwidth among all connections
  - I.e.: If n connections share a given bottleneck link, each connection obtains 1/n of its bandwidth (in the long run)
- Interaction with other protocols
  - Bottleneck bandwidth depends on load of other protocols as well, e.g., UDP – which is NOT congestion-controlled
  - I.e., UDP traffic can “push aside” TCP traffic
- Consequence: Transport protocols should be ***TCP friendly***
  - They should not consume more bandwidth than a TCP connection in a comparable situation

TCP packet header

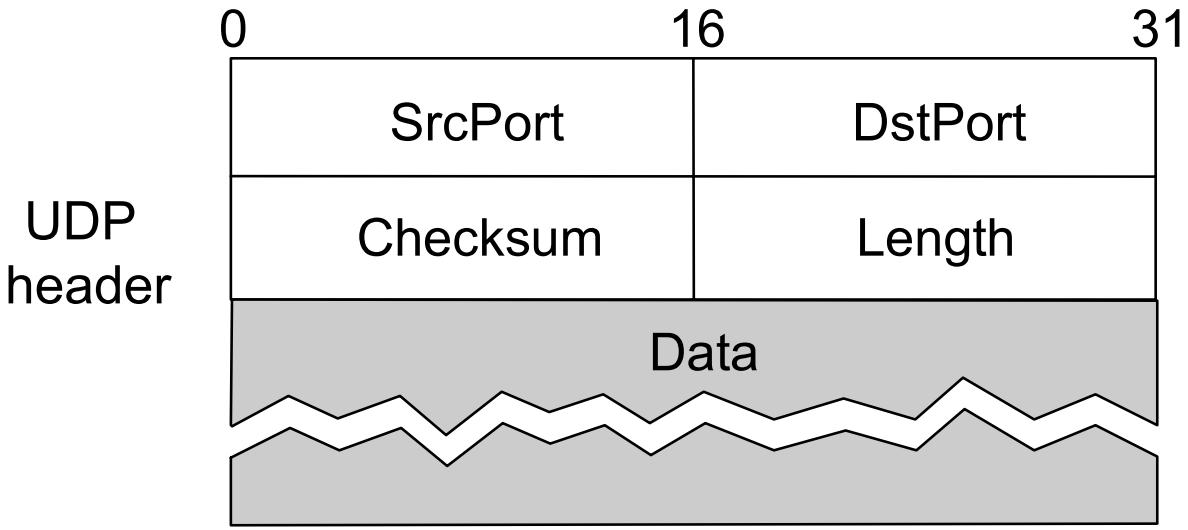


Overview

- Services & addressing
- Connection setup
- Connection release
- Flow control
- Timer and timeouts
- TCP summary
- ***UDP***
- Performance issues

A simple demultiplexer transport protocol – UDP

- An example for an unreliable, datagram protocol: User Datagram Protocol (UDP)
- Only real function: (de)multiplex several data flows onto the IP layer and back to the applications
- In addition: ensures packet's correctness
  - Checksum out of UDP header + data + "pseudoheader" (pivotal IP header fields)



Conclusion

- Transport protocols can be anything from trivial to highly complex, depending on the purpose they serve
- They determine to a large degree the dynamics of a network and – in particular – its stability
  - It is trivial to build “faster” than TCP protocols, but they are unstable
- The interdependencies of various mechanisms in a transport protocols can be very subtle with big consequences
  - E.g., fairness, coexistence of different TCP versions