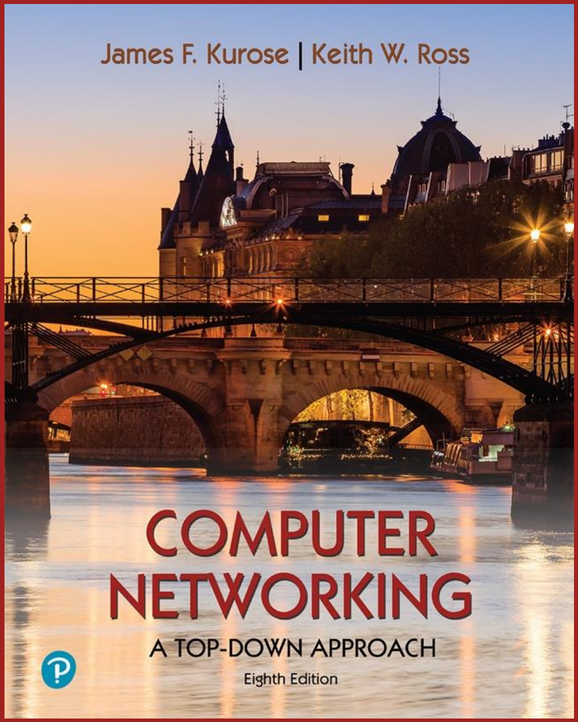
Communication Networks
W. Tavernier

Chapter 3
Transport Layer



Computer Networking: A Top-Down Approach
8th Edition, 2020, Pearson,
James F. Kurose, Keith W. Ross

Part of slides provided by J.F Kurose and K.W. Ross, All Rights Reserved

Transport Layer

our goals:

- understand principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- learn about Internet transport layer protocols:
  - UDP: connectionless transport
  - TCP: connection-oriented reliable transport
  - TCP congestion control

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

[3.4 principles of reliable data transfer]

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

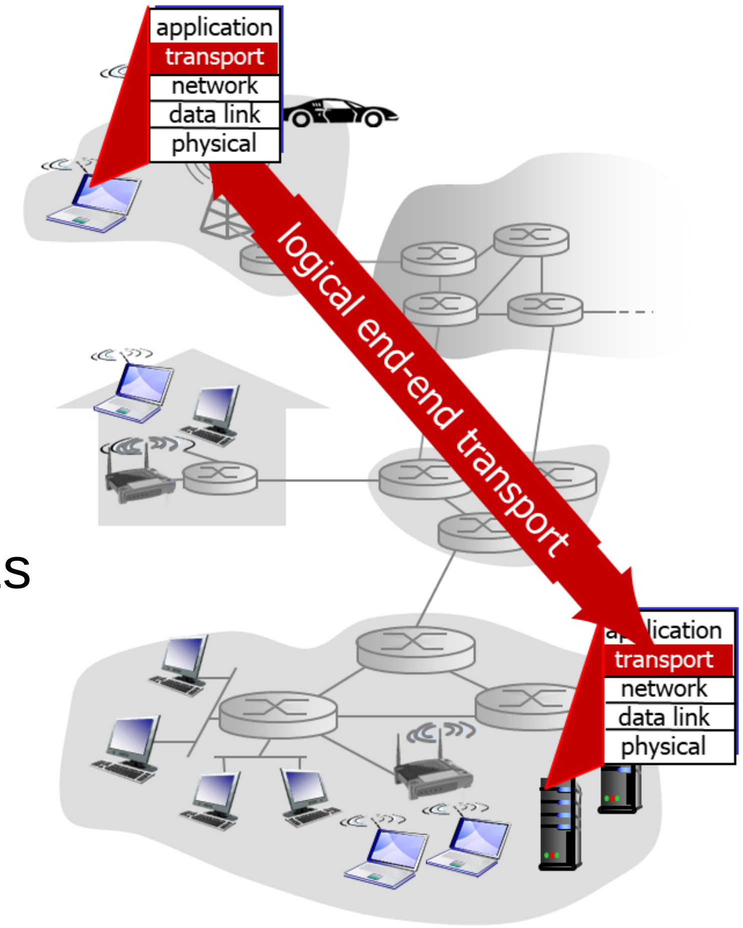
[3.6 principles of congestion control]

[3.7 TCP congestion control]

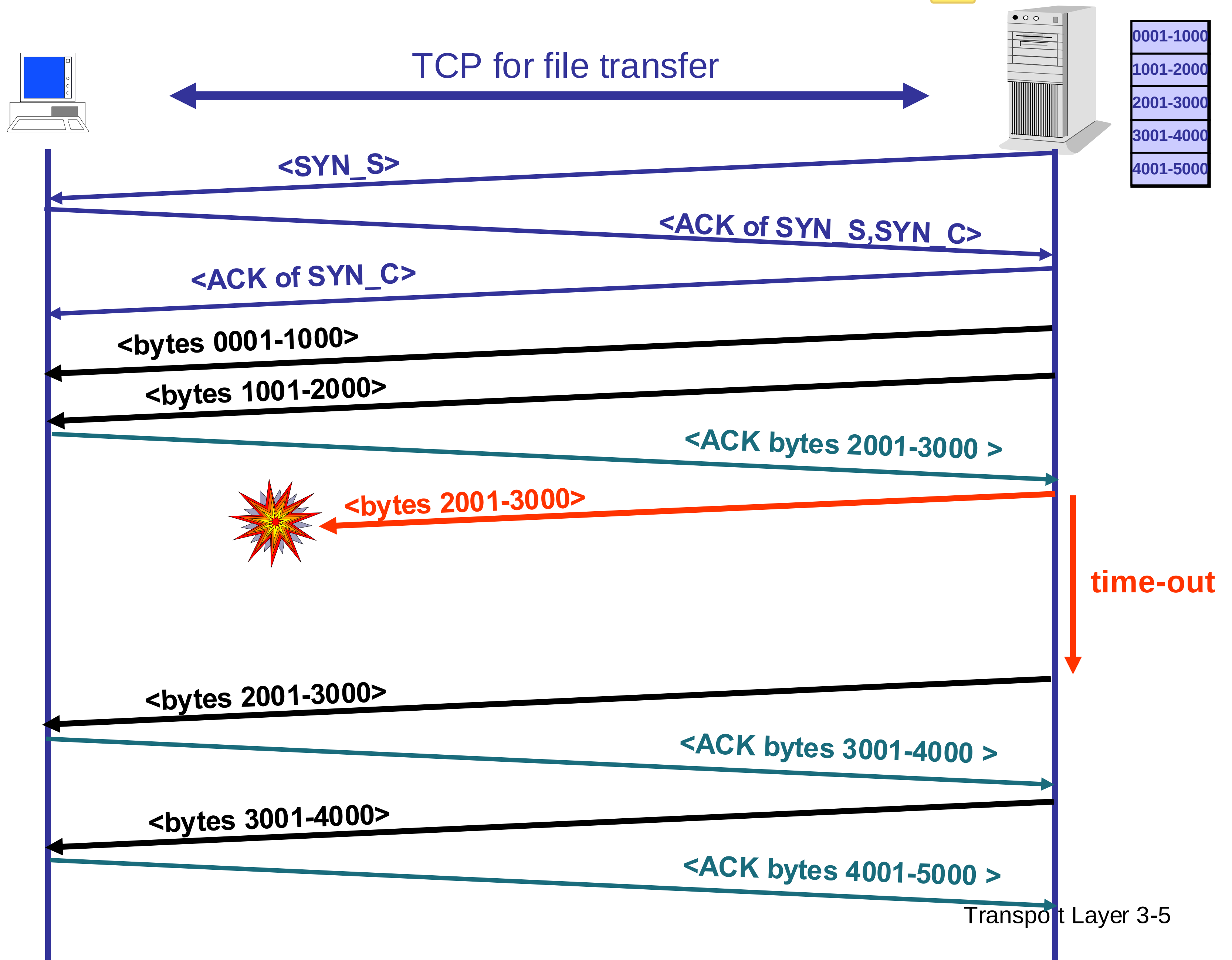
3.8 Evolving transport-layer functionality

Transport Services and Protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP

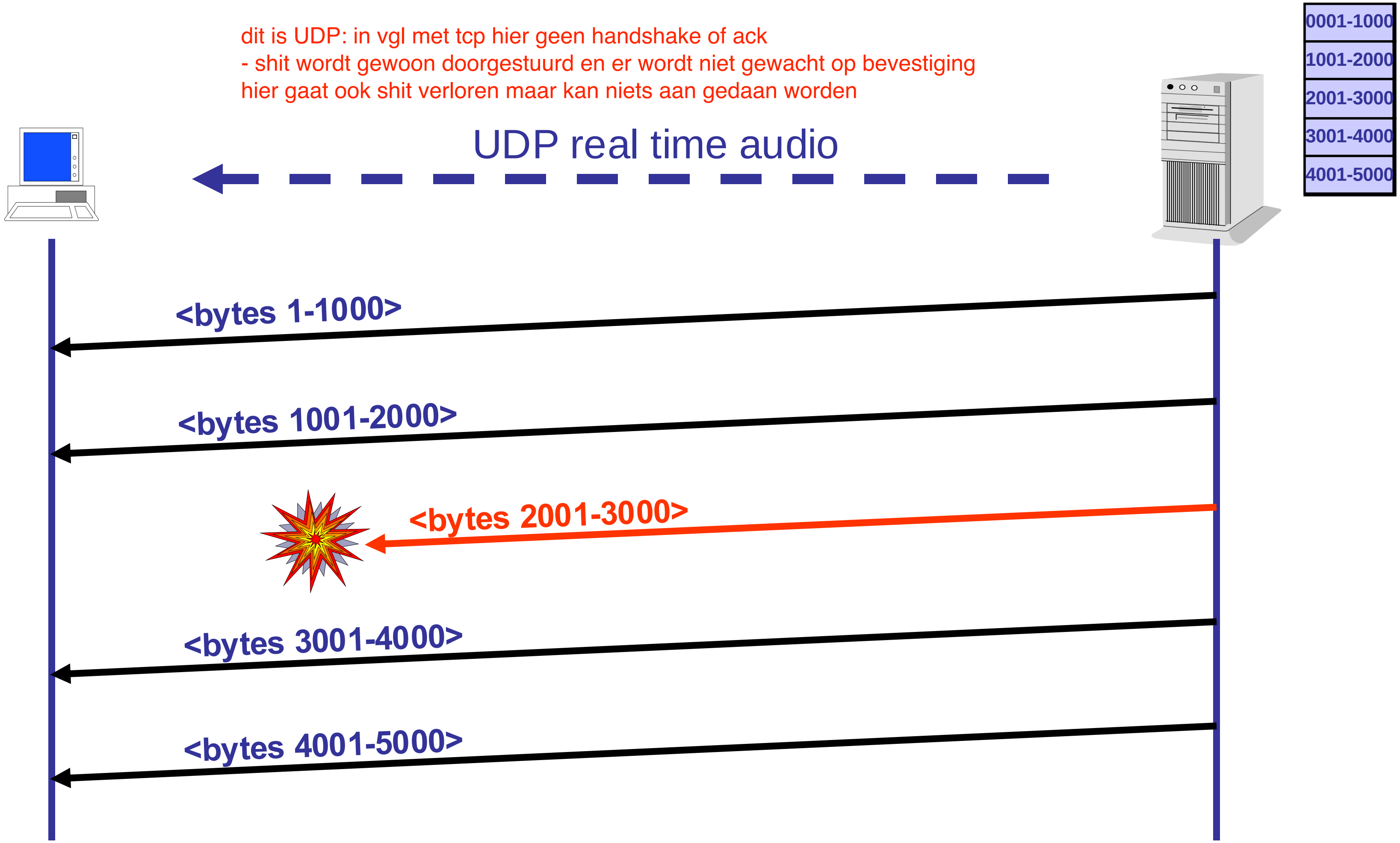


TCP connection for file transfer



UDP for real time audio transfer

dit is UDP: in vgl met tcp hier geen handshake of ack
- shit wordt gewoon doorgestuurd en er wordt niet gewacht op bevestiging
hier gaat ook shit verloren maar kan niets aan gedaan worden



- send as quick as possible
- no extra delay due to acknowledgment
- no retransmissions

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

[3.4 principles of reliable data transfer]

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

[3.6 principles of congestion control]

[3.7 TCP congestion control]

3.8 Evolving transport-layer functionality

Examples: web browsing & DNS

IP address: 213.118.203.222
port: 12355

TCP connectivity

IP address: 104.200.23.95
port: 80

- **Server port numbers are usually fixed by convention to particular applications**
- **Client port numbers are assigned (pseudo-)randomly and temporary**

IP address: 213.118.203.202
port: 2398

UDP connectivity

IP address: 104.200.23.95
port: 53

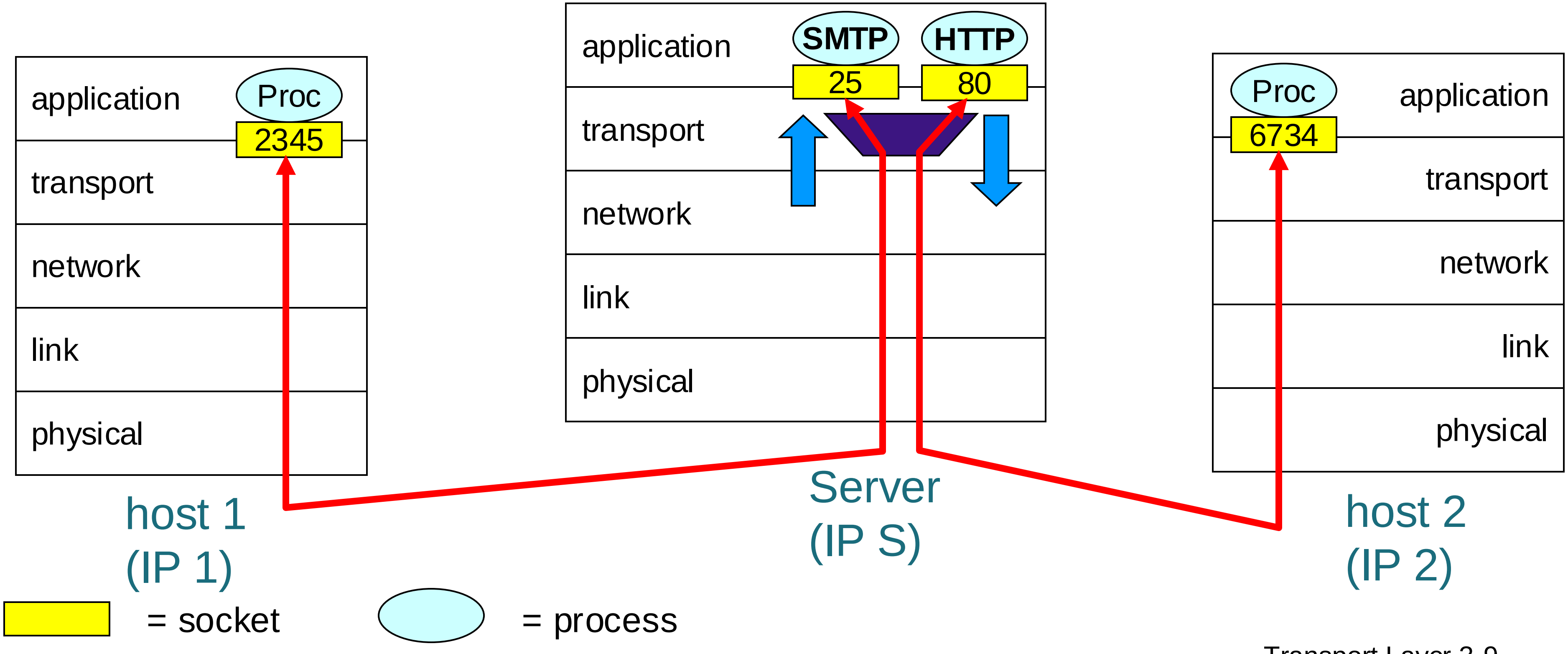
Multiplexing/demultiplexing

Demultiplexing at rcv side:

delivering received segments to correct socket

Multiplexing at send side:

gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

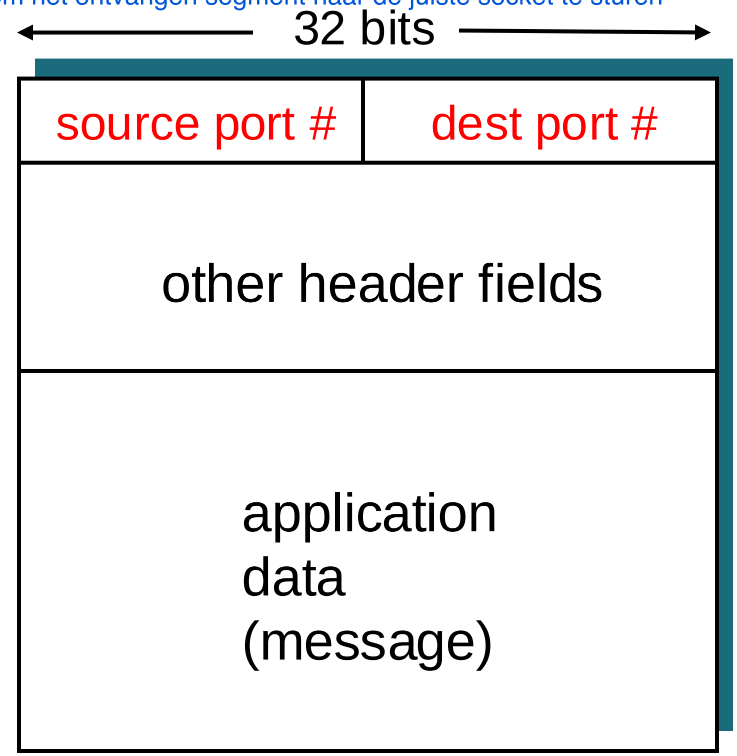


How demultiplexing works

host ontvangt een ip-pakket dat de
- bron en bestemming ip adressen bevat
- een transportlaag-segmet
  - bron en bestemmings pport nymmer

host gebruikt de combinatie vaan ip adresesn en poort nummers om het ontvangen segment naar de juiste socket te sturen

- host receives IP packet
  - each packet has source IP address, destination IP address
  - each packet carries 1 transport-layer segment
  - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket
- Two options:
  - connectionless = UDP (e.g. for DNS)
  - connection oriented = TCP (e.g. for HTTP)



TCP/UDP segment format

Connectionless (de)multiplexing (➔ UDP)

- Create sockets with port numbers:

```
DatagramSocket serverSocket1 =
  new DatagramSocket(53);
```

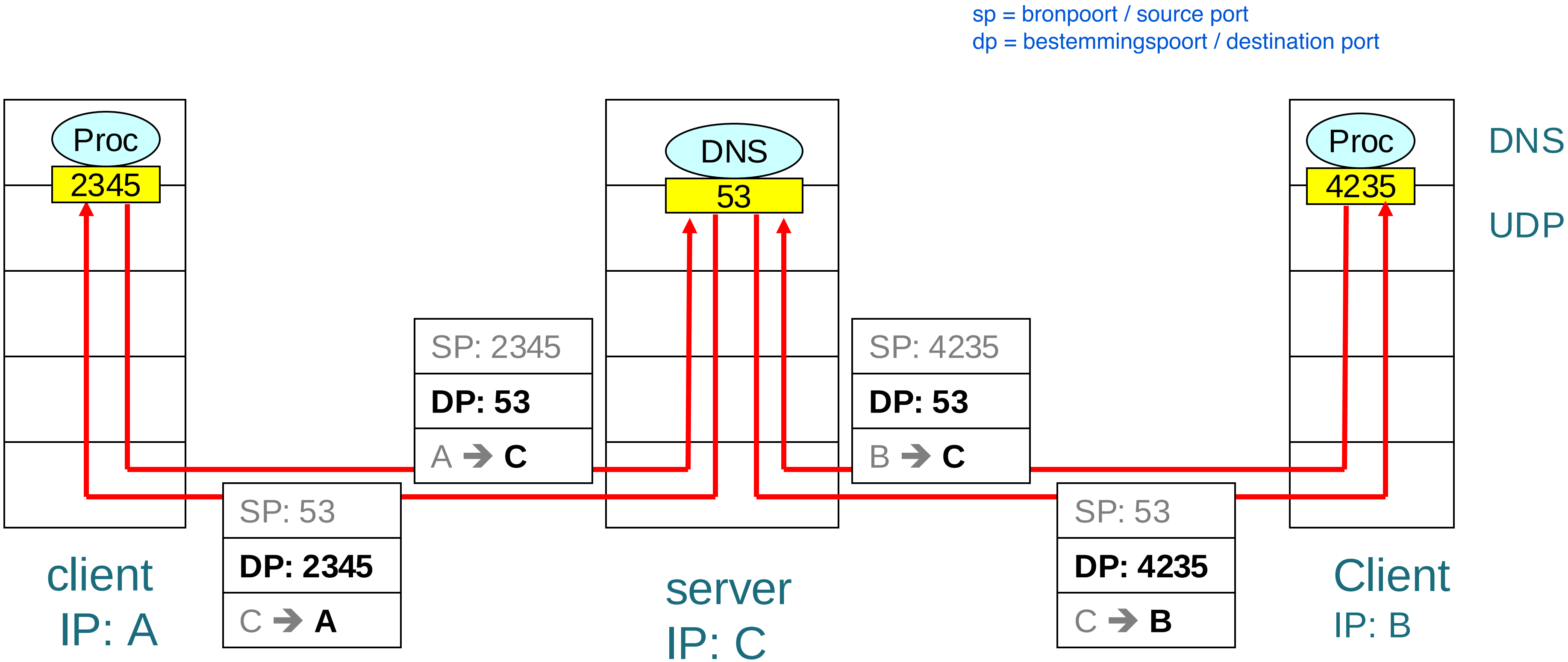
- UDP socket identified by two-tuple:

(dest IP address, dest port number)

- When host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number
- IP datagrams (packets) with different source IP addresses and/or source port numbers directed to same socket on this destination

Connectionless (de)multiplexing

```
DatagramSocket serverSocket = new DatagramSocket(53);
```



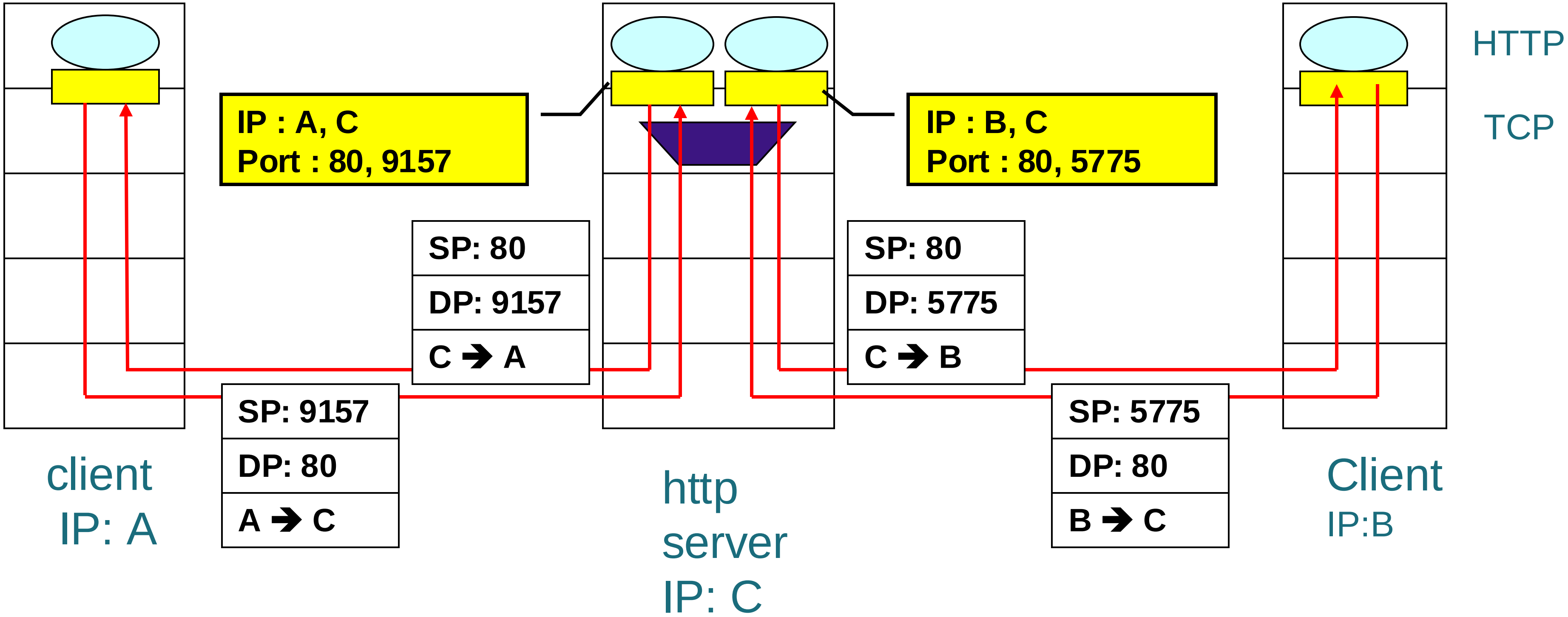
Source Port (SP) provides "return address" to DNS deamon, not used by UDP

Connection-oriented (de)mux (→ TCP)

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- receiver uses all four values to direct segment to appropriate socket

- Server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

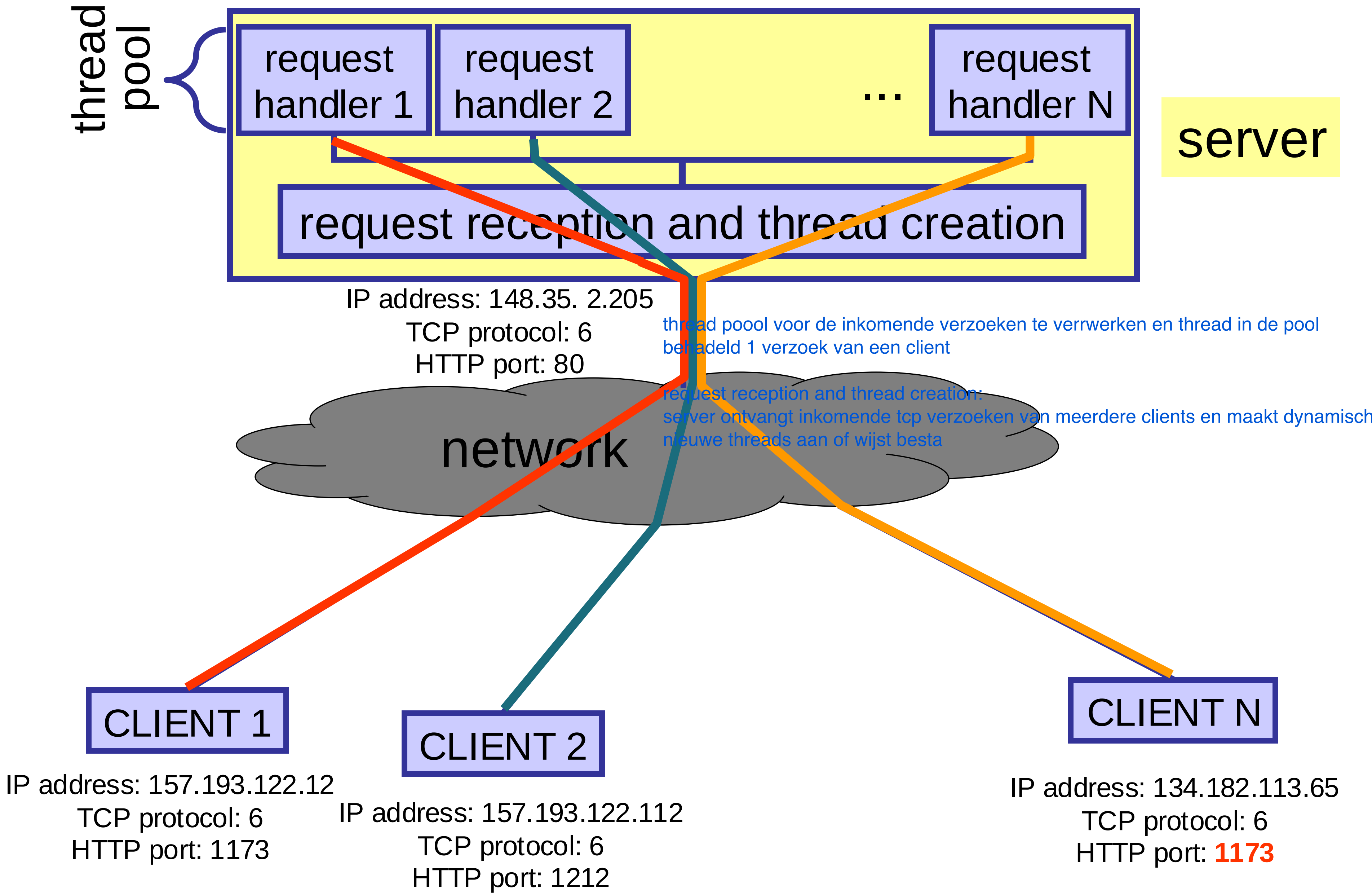
Connection-oriented (de)mux



dit kan niet met udp

New process started for every new client connecting
Two processes from same client will start two processes at server
(only difference in tuple is port at client)

Dynamic Server Process creation



TCP segment structure

20, 21 : FTP
23 : Telnet
25 SMTP
80 : HTTP
0-1023 : reserved
>1023 : ephemeral
(short lived) port

<table>
<tr><td colspan="8">16-bit source port number</td><td>16-bit destination port number</td></tr>
<tr><td colspan="9">32-bit sequence number</td></tr>
<tr><td colspan="9">32-bit acknowledgement number</td></tr>
<tr><td>4-bit header length</td><td>unused (6 bits)</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>16-bit window size</td></tr>
<tr><td colspan="8">16-bit TCP checksum</td><td>16-bit urgent pointer</td></tr>
<tr><td colspan="9">Options (if any)</td></tr>
<tr><td colspan="9">Data</td></tr>
</table>

Common server port mappings for applications

1. Addressing

The netstat command is a diagnostic tool enabling to show which open connections there are.

Application	Usual server port	Transport Layer Protocol
FTP	20,21	TCP
SSH	22	TCP
Telnet	23	TCP
SMTP (email transfer server)	25	TCP
DNS	53	UDP/TCP
HTTP/s (web)	80 / 443	TCP
POP3 (email server)	110	TCP

Note that these are assigned by convention (reserved as per RFC 1700).
Port numbers > 1024 can be freely used (e.g., web server at port 443)

```
$ sudo netstat -plnt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:3306            0.0.0.0:*               LISTEN      3686/mysqld
tcp        0      0 :::443                  :::*                    LISTEN      2218/httpd
tcp        0      0 :::80                   :::*                    LISTEN      2218/httpd
tcp        0      0 :::22                   :::*                    LISTEN      1051/sshd
```

Use of TCP

Port	Protocol	Description
7	Echo	Sends back what is received
9	Discard	Discards what is received
13	Daytime	Sends back the time of day
20	FTP data	Data channel for FTP
21	FTP control	Control channel for FTP (get, put, …)
23	Telnet	Default port for telnet application
25	SMTP	Used for sending email to a mailserver
53	DNS	Domain Name System over TCP
80	HTTP	Used in the World Wide Web
109	POPv2	Used for reading email on a mailserver
110	POPv3	Used for reading email on a mailserver
111	SUN RPC	Sun's Remote Procedure Call over TCP
119	NNTP	Network News Transfer Protocol (newsgroups)
143	IMAP	Used for reading email on a mailserver
161-162	SNMP	Simple Network Management Protocol
179	BGP	Border Gateway Protocol
194	IRC	Internet Relay Chat, a chat service
220	IMAPv3	Used for reading email on a mailserver
515	Print Spooler	Used in print servers
666	Doom	The popular 3D game by Id Software
6000-6063	X11	The X Window System

Online: https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers
Linux: `/etc/services`

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

[3.4 principles of reliable data transfer]

3.5 connection-oriented transport: TCP

- **segment structure**
- reliable data transfer
- flow control
- connection management

[3.6 principles of congestion control]

[3.7 TCP congestion control]

3.8 Evolving transport-layer functionality

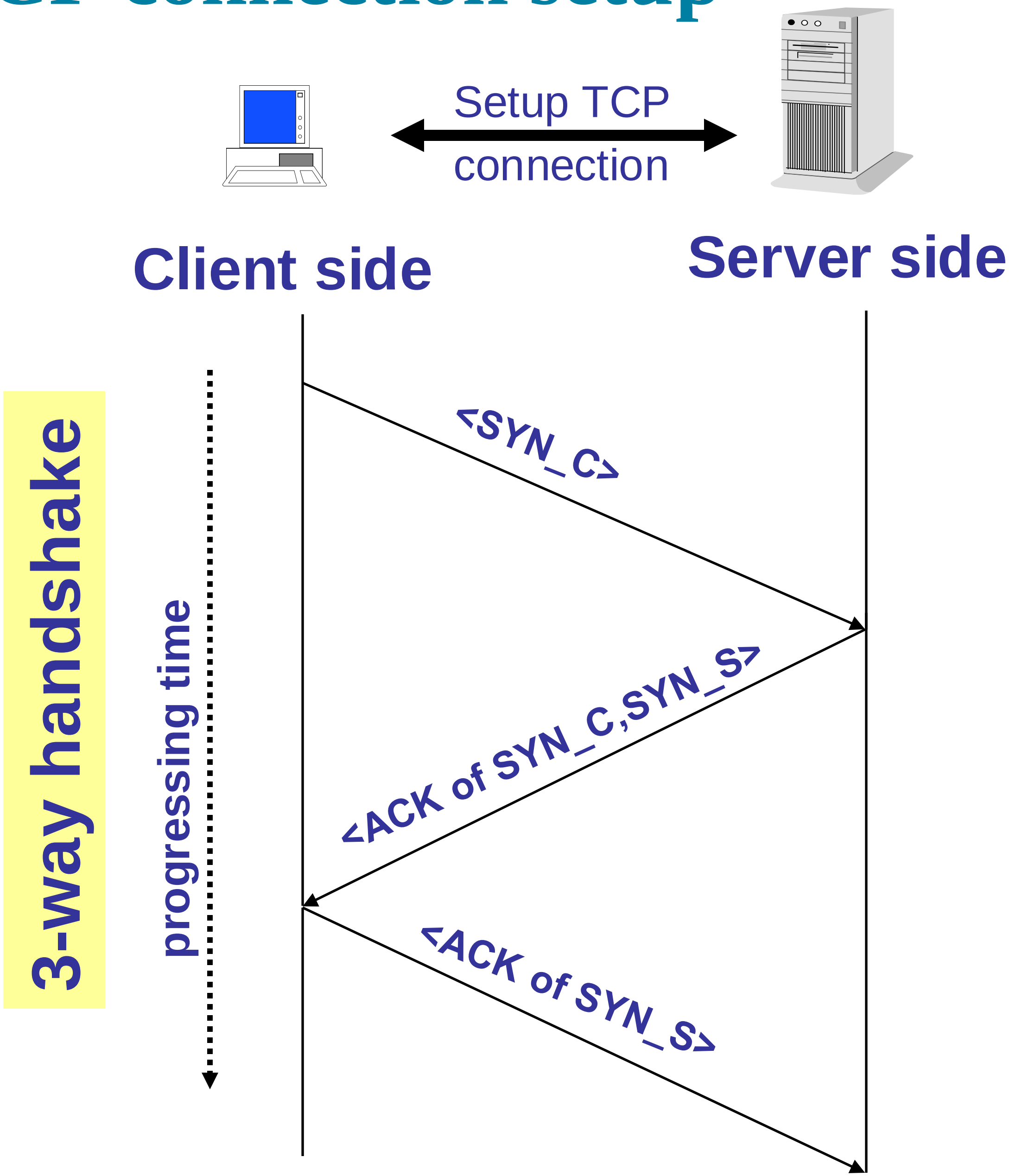
Interesting to understand basics of retransmission protocols and concept of FSM (Finite State Machine)

TCP

Overview RFCs: 793, 1122, 1323, 2018, 2581

- **point-to-point:**
  - one sender, one receiver
- **reliable, in-order byte stream:**
  - no "message boundaries"
- **pipelined:**
  - TCP congestion and flow control set window size
- **full duplex data:**
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- **connection-oriented:**
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- **flow controlled:**
  - sender will not overwhelm receiver
- **Congestion controlled:**
  - Without help from the network
- **No guarantees …**
  - On delay, delay variation, bandwith …
  - Like IP

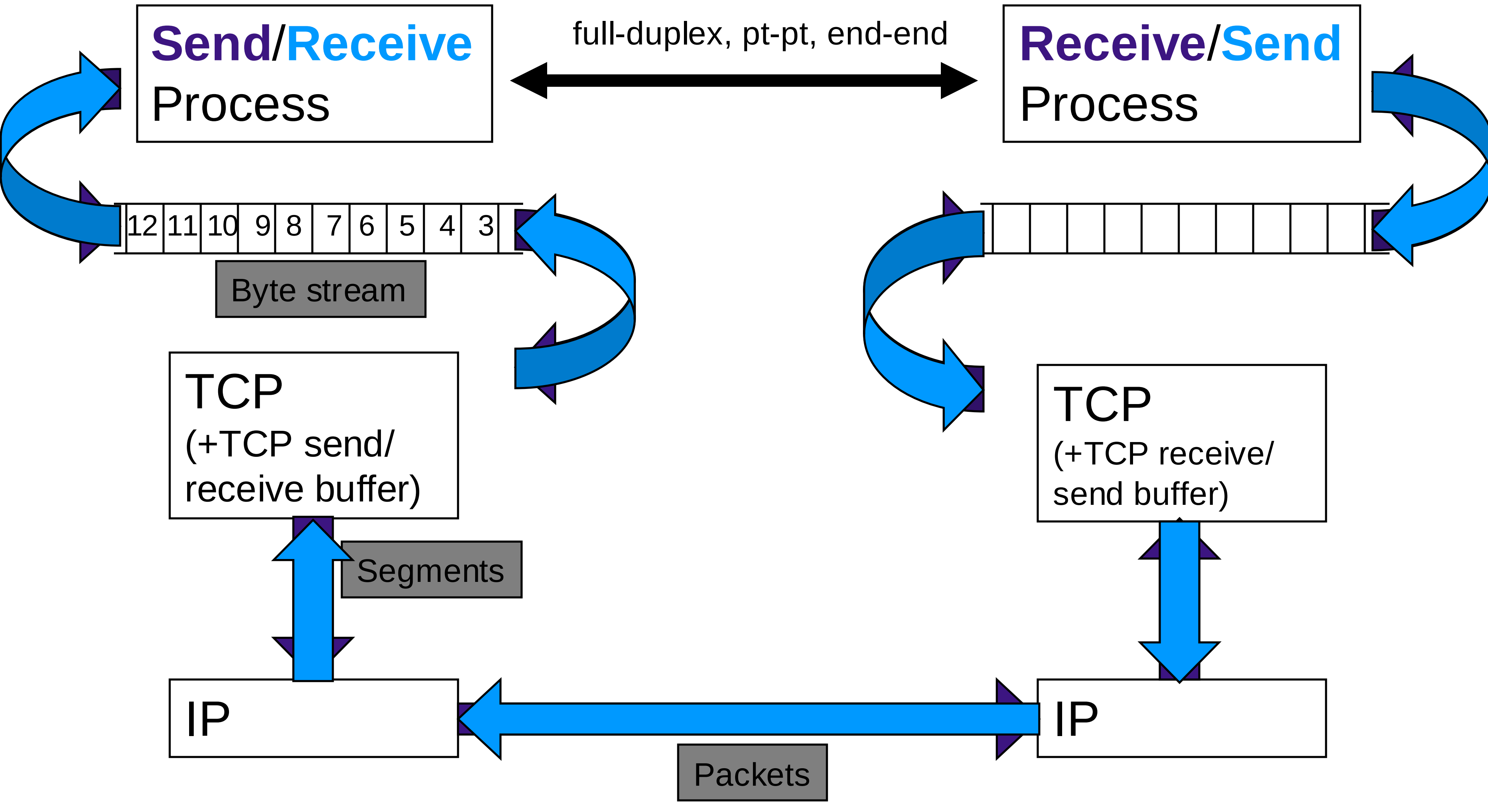
TCP connection setup



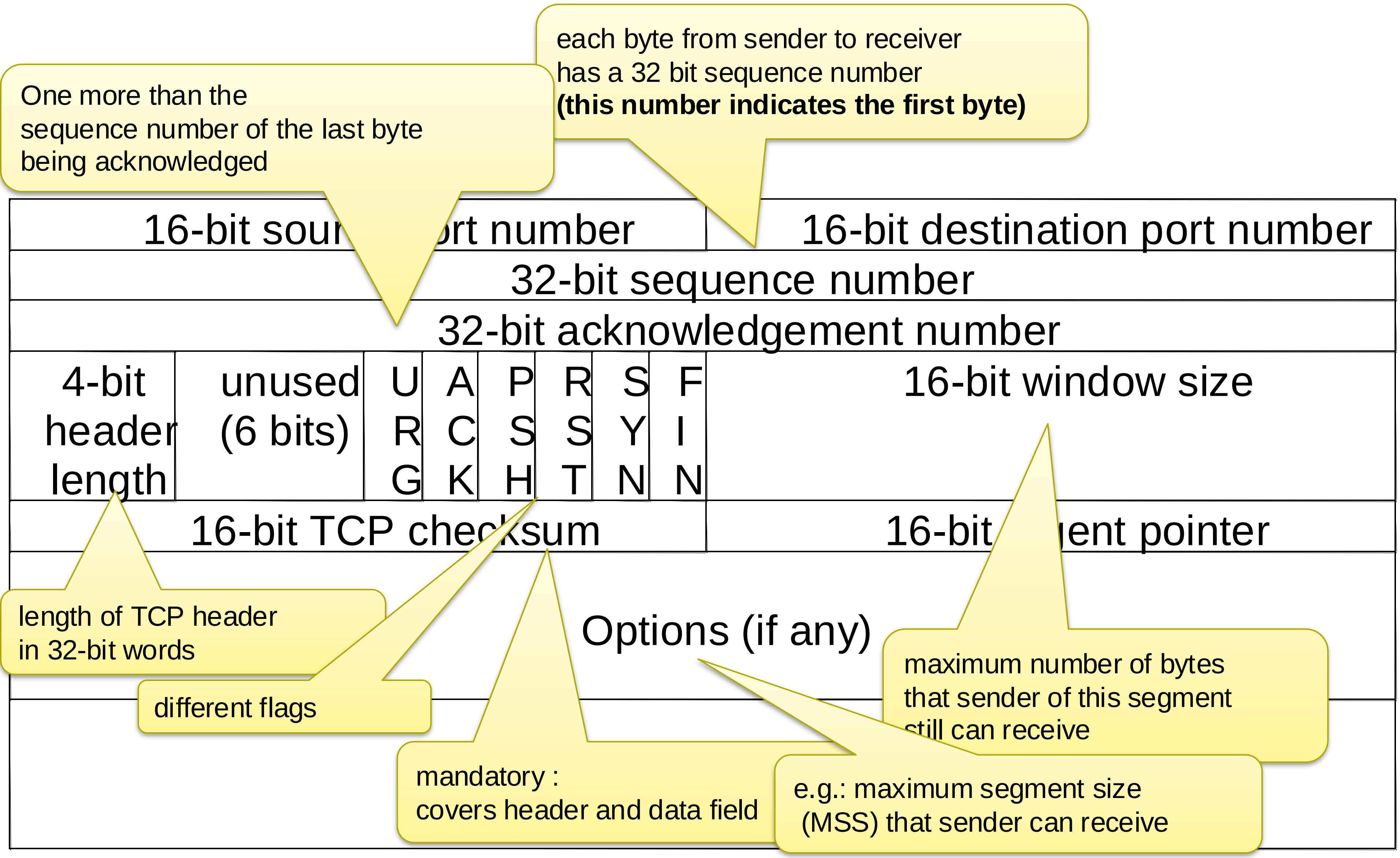
SYN : SYNchronization
ACK : ACKnowledgment
C : Client side
S : Server side

If set-up segment is lost
==> time-outs

TCP: overview



TCP segment structure



Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

[3.4 principles of reliable data transfer]

3.5 connection-oriented transport: TCP

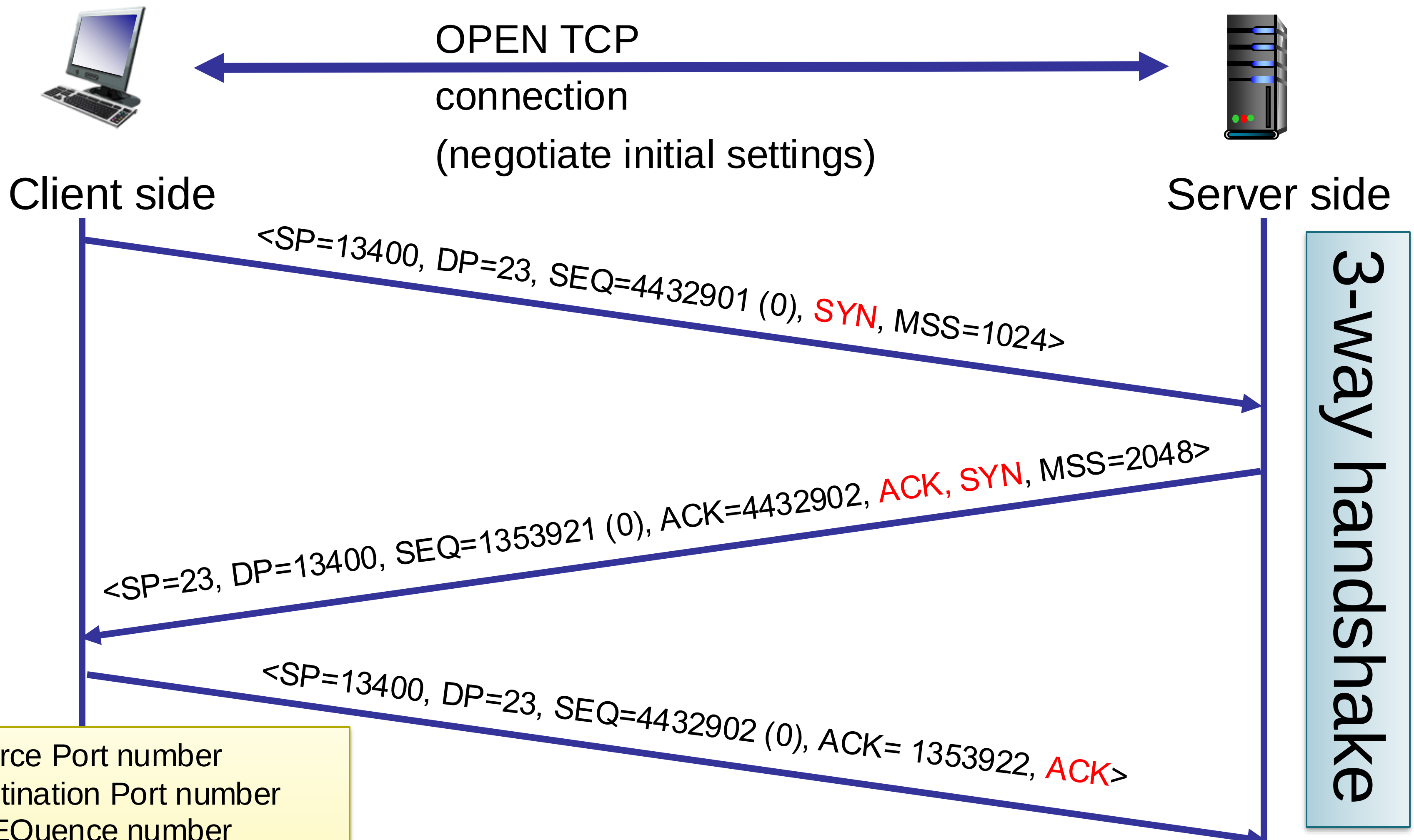
- segment structure
- reliable data transfer
- flow control
- **connection management**

[3.6 principles of congestion control]

[3.7 TCP congestion control]

3.8 Evolving transport-layer functionality

TCP Connection OPEN



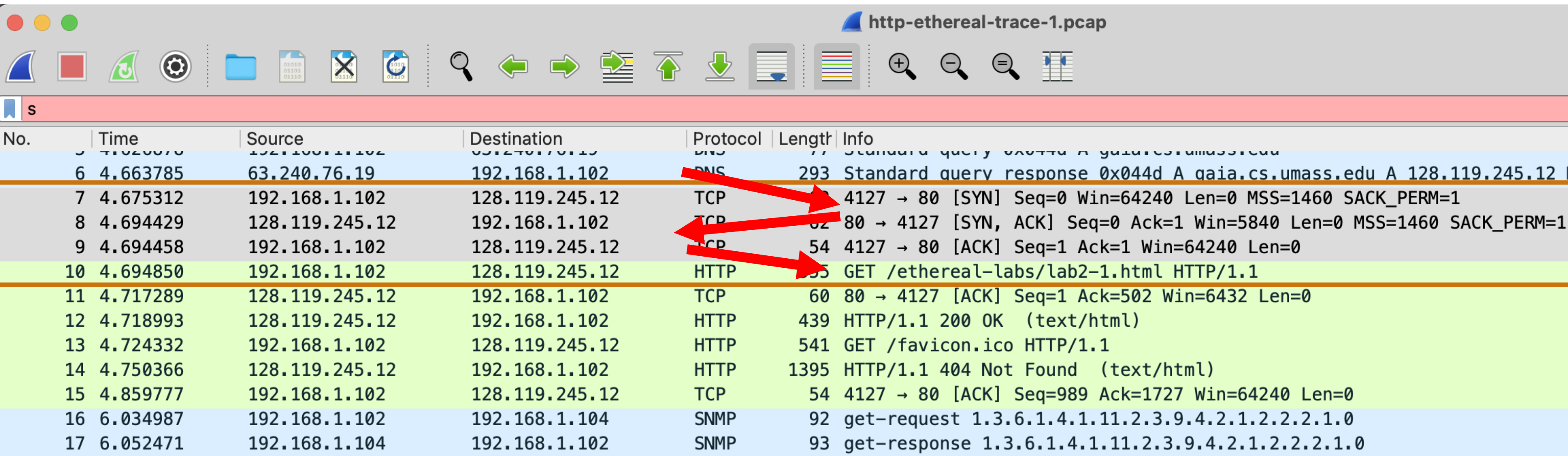
SP : Source Port number
DP : Destination Port number
SEQ : SEQuence number
(…) : length data field
ACK : ACKnowledgment number
SYN : SYN flag set to 1
ACK : ACK flag set to 1
MSS : Maximum Segment Size

- most important fields in TCP header are indicated
- last segment may contain data
- if server has no application protocol running at the requested port (DP), it will return the RST = 1 flag (reset sender)
- ISN (Initial Sequence Number): based on timed counter (see notes)

Notes bij vorige pg

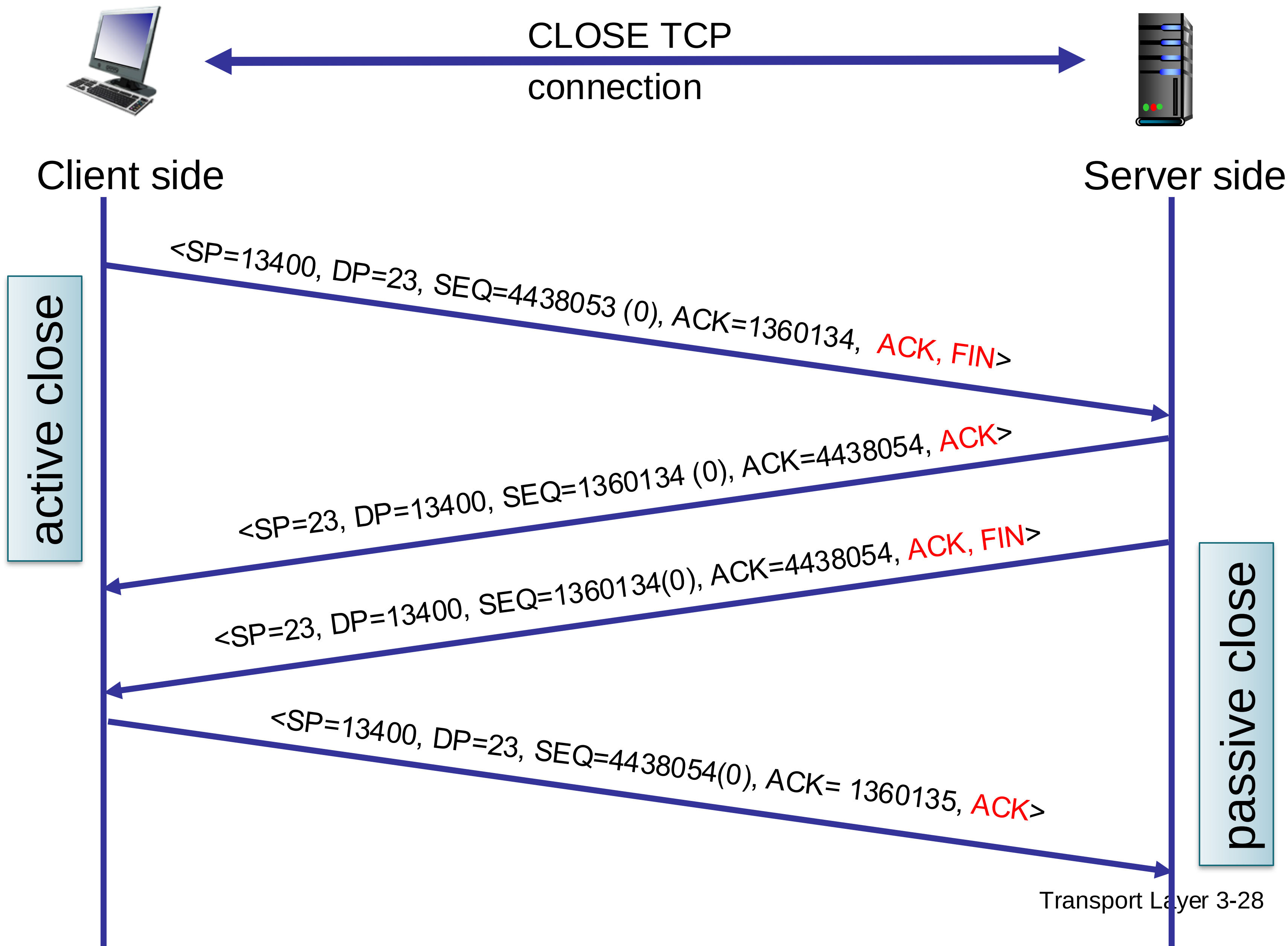
Selecting the Initial Sequence Number:
Traditionally, each device chose the ISN by making use of a timed counter, like a clock of sorts, that was incremented every 4 microseconds. This counter was initialized when TCP started up and then its value increased by 1 every 4 microseconds until it reached the largest 32-bit value possible (4,294,967,295) at which point it "wrapped around" to 0 and resumed incrementing. Any time a new connection is set up, the ISN was taken from the current value of this timer. Since it takes over 4 hours to count from 0 to 4,294,967,295 at 4 microseconds per increment, this virtually assured that each connection will not conflict with any previous ones.
One issue with this method is that it makes ISNs predictable. A malicious person could write code to analyze ISNs and then predict the ISN of a subsequent TCP connection based on the ISNs used in earlier ones. This represents a security risk, which has been exploited in the past (such as in the case of the famous Mitnick attack). To defeat this, implementations now use a random number in their ISN selection process.

TCP connection setup with webserver in Wireshark



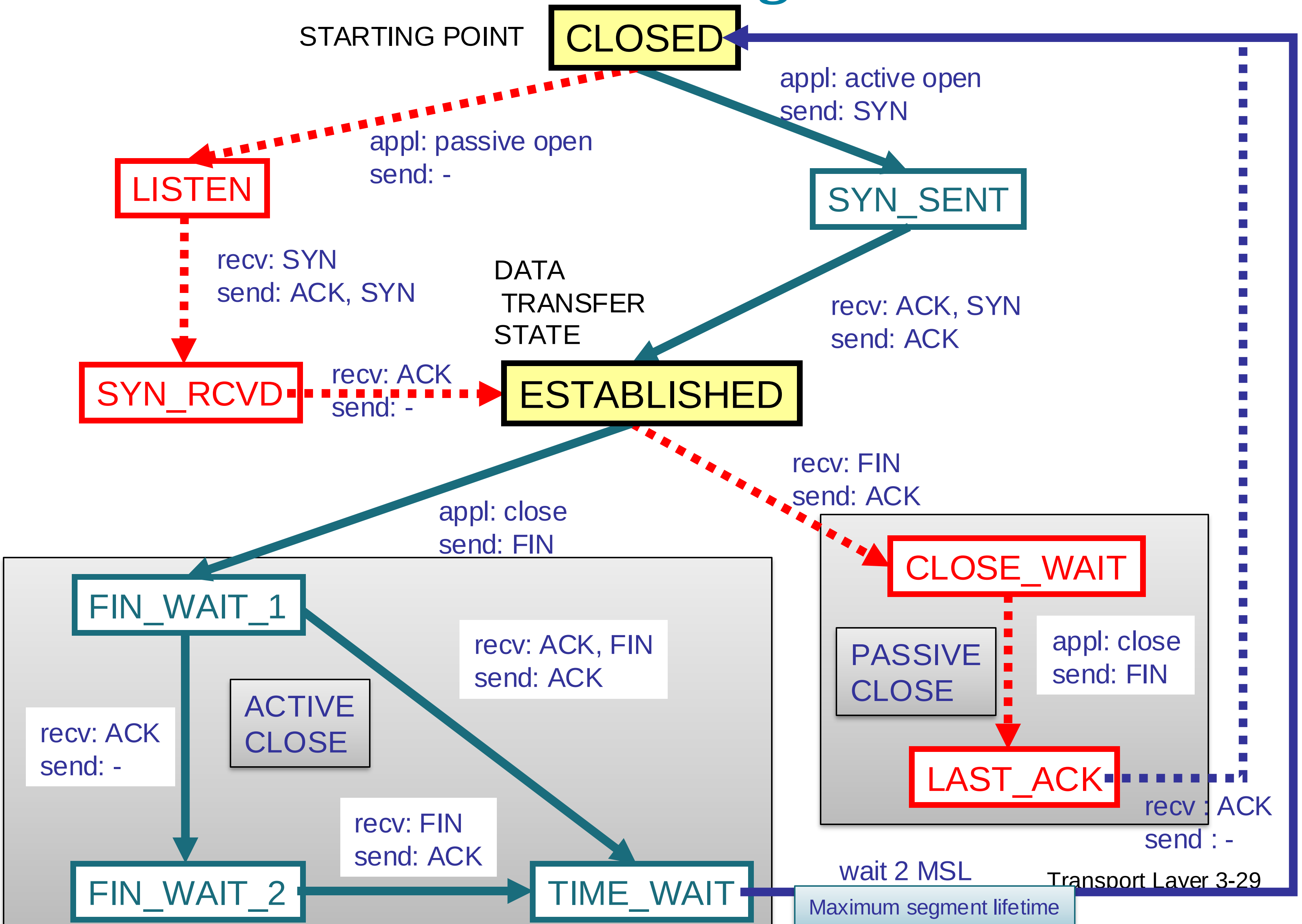
No.	Time	Source	Destination	Protocol	Length	Info
6	4.663785	63.240.76.19	192.168.1.102	DNS	293	Standard query response 0x044d A gaia.cs.umass.edu A 128.119.245.12 NS
7	4.675312	192.168.1.102	128.119.245.12	TCP	[illegible]	4127 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
8	4.694429	128.119.245.12	192.168.1.102	TCP	62	80 → 4127 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM=1
9	4.694458	192.168.1.102	128.119.245.12	TCP	54	4127 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
10	4.694850	192.168.1.102	128.119.245.12	HTTP	[illegible]	GET /ethereal-labs/lab2-1.html HTTP/1.1
11	4.717289	128.119.245.12	192.168.1.102	TCP	60	80 → 4127 [ACK] Seq=1 Ack=502 Win=6432 Len=0
12	4.718993	128.119.245.12	192.168.1.102	HTTP	439	HTTP/1.1 200 OK (text/html)
13	4.724332	192.168.1.102	128.119.245.12	HTTP	541	GET /favicon.ico HTTP/1.1
14	4.750366	128.119.245.12	192.168.1.102	HTTP	1395	HTTP/1.1 404 Not Found (text/html)
15	4.859777	192.168.1.102	128.119.245.12	TCP	54	4127 → 80 [ACK] Seq=989 Ack=1727 Win=64240 Len=0
16	6.034987	192.168.1.102	192.168.1.104	SNMP	92	get-request 1.3.6.1.4.1.11.2.3.9.4.2.1.2.2.2.1.0
17	6.052471	192.168.1.104	192.168.1.102	SNMP	93	get-response 1.3.6.1.4.1.11.2.3.9.4.2.1.2.2.2.1.0

TCP Connection CLOSE



vraag notes aan korneel / gpt legt goed uit

TCP State Transition Diagram



Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

[3.4 principles of reliable data transfer]

3.5 connection-oriented transport: TCP

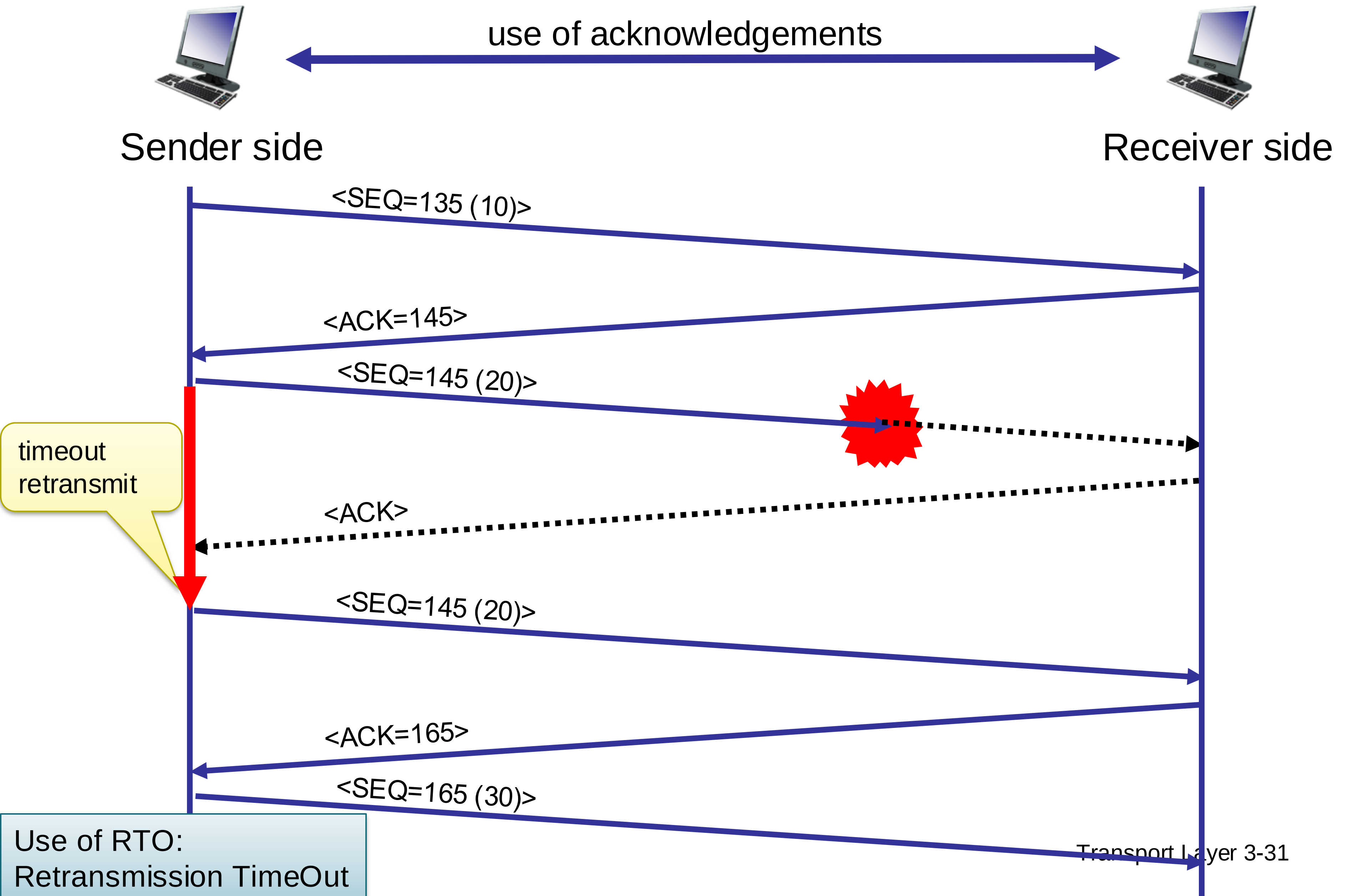
- segment structure
- **reliable data transfer**
- flow control
- connection management

[3.6 principles of congestion control]

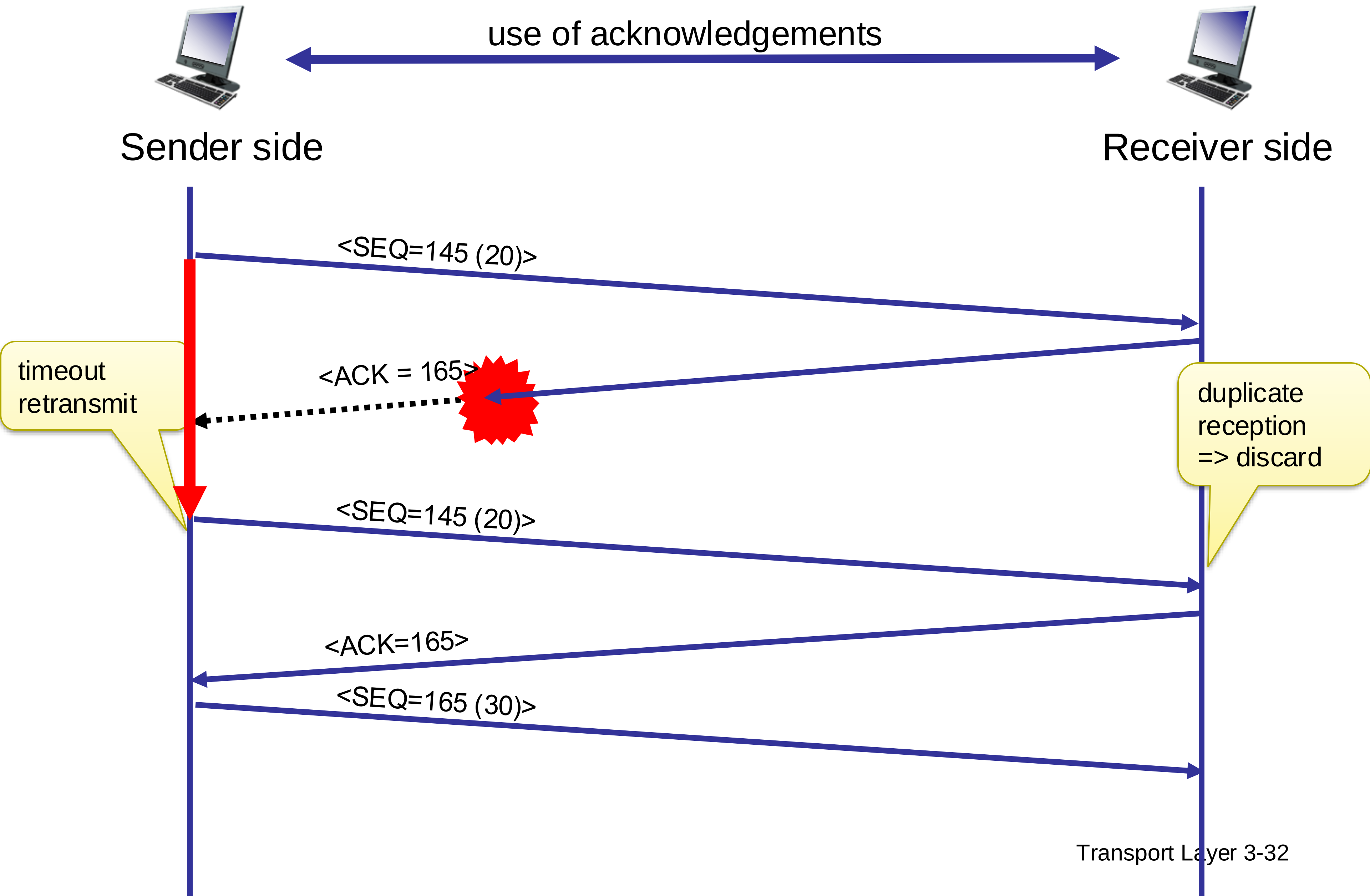
[3.7 TCP congestion control]

3.8 Evolving transport-layer functionality

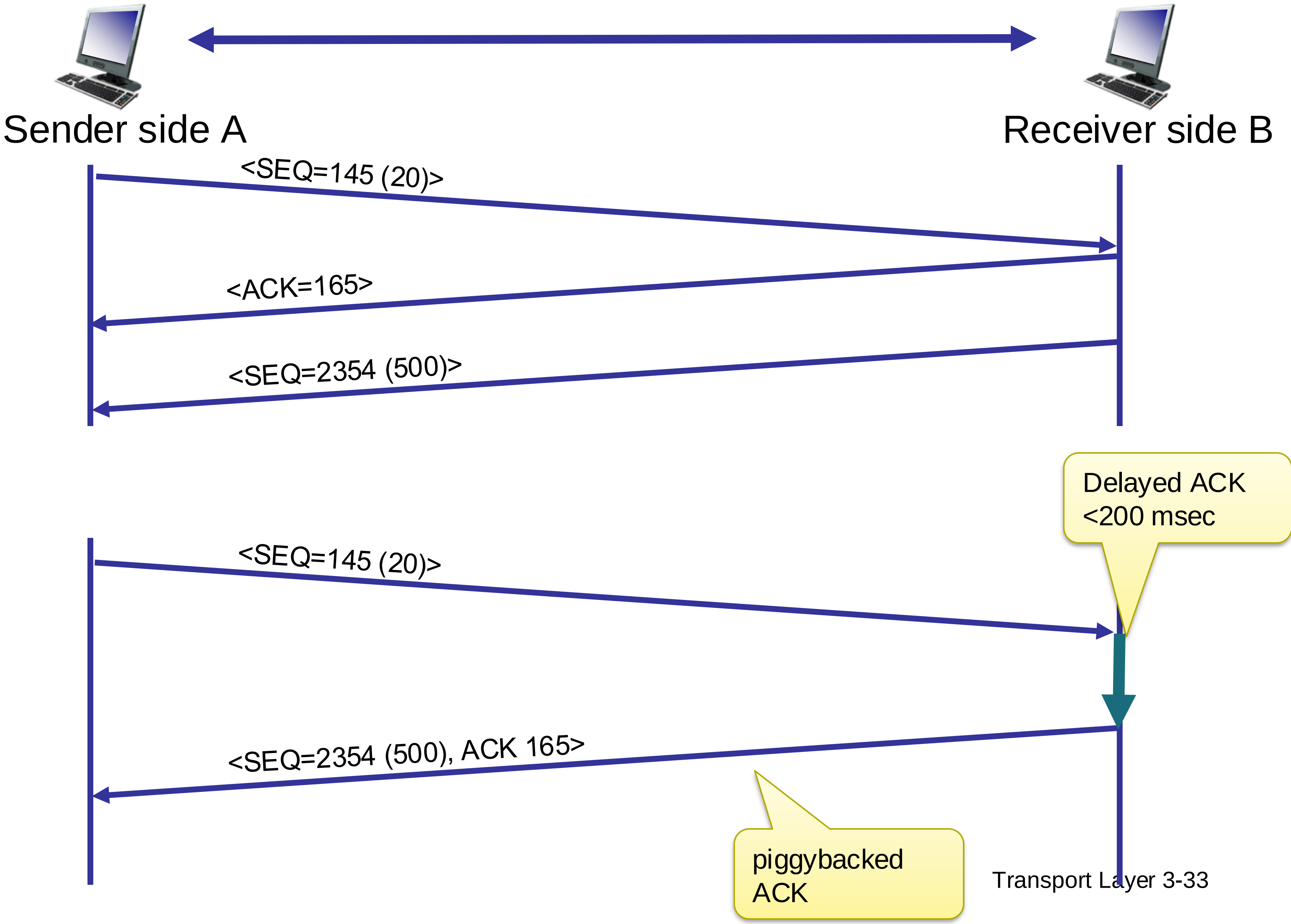
Acknowledgment and retransmission



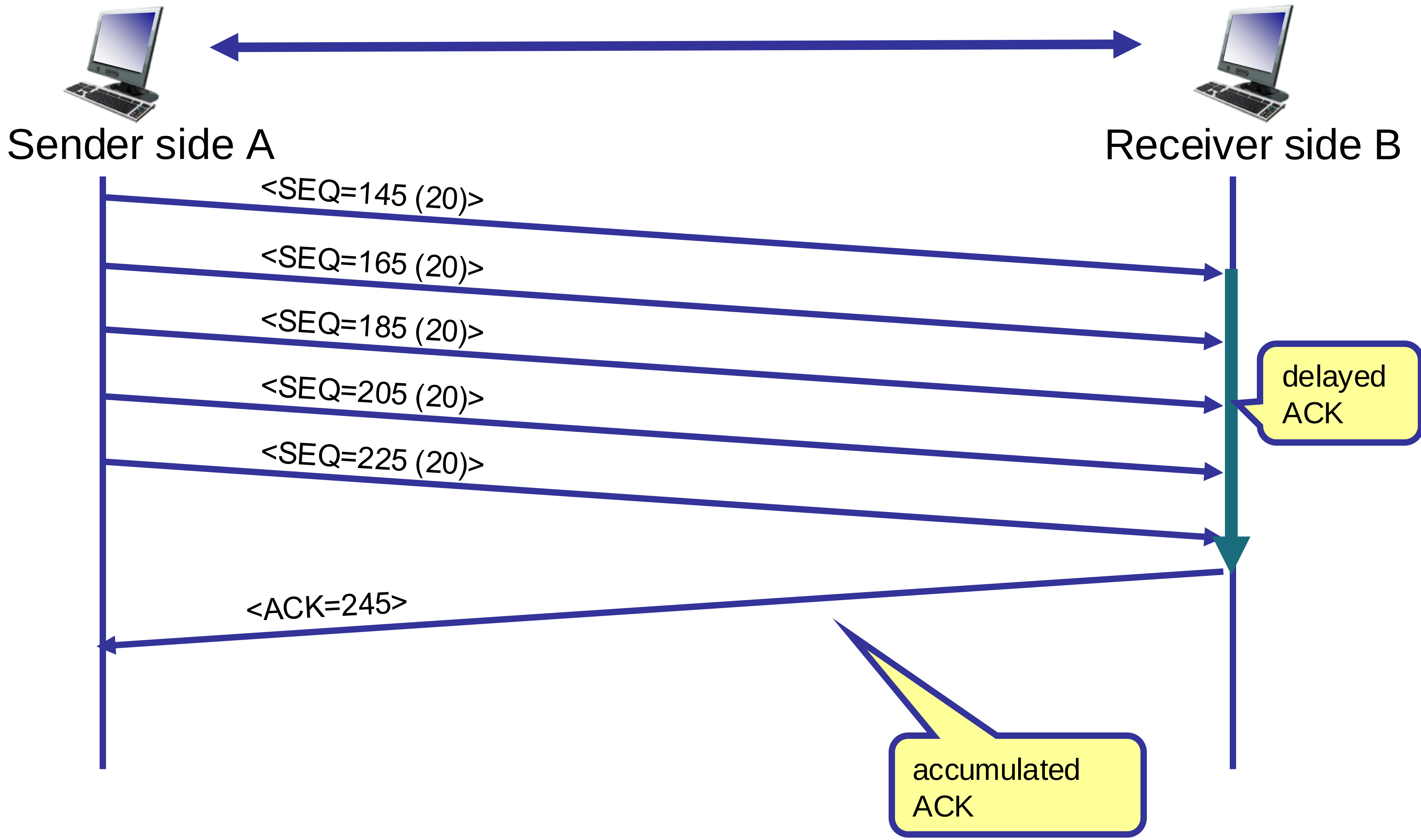
Ack/retrans/duplicate reception



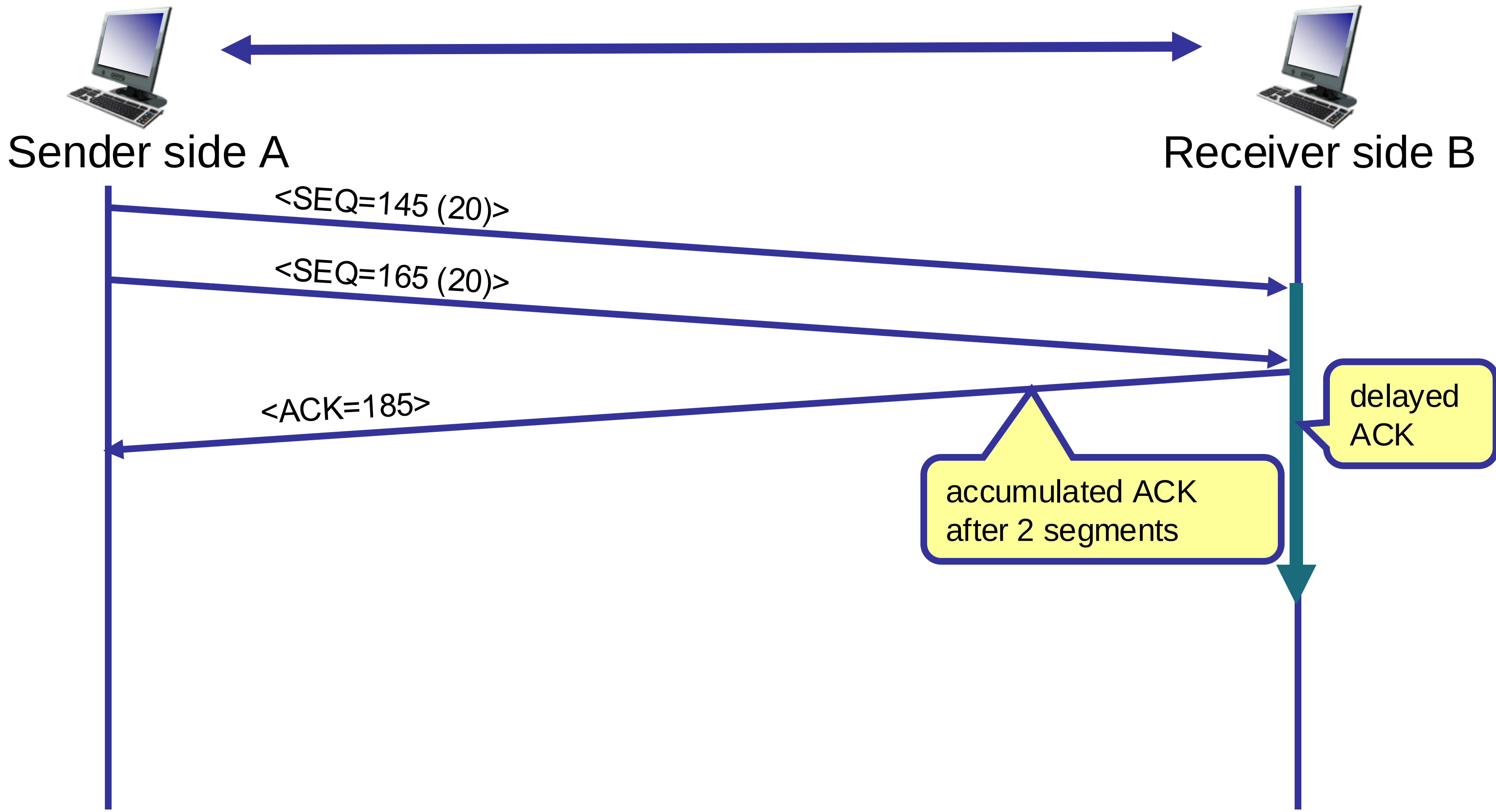
Piggybacking and delayed ACKs



Delayed accumulated ack

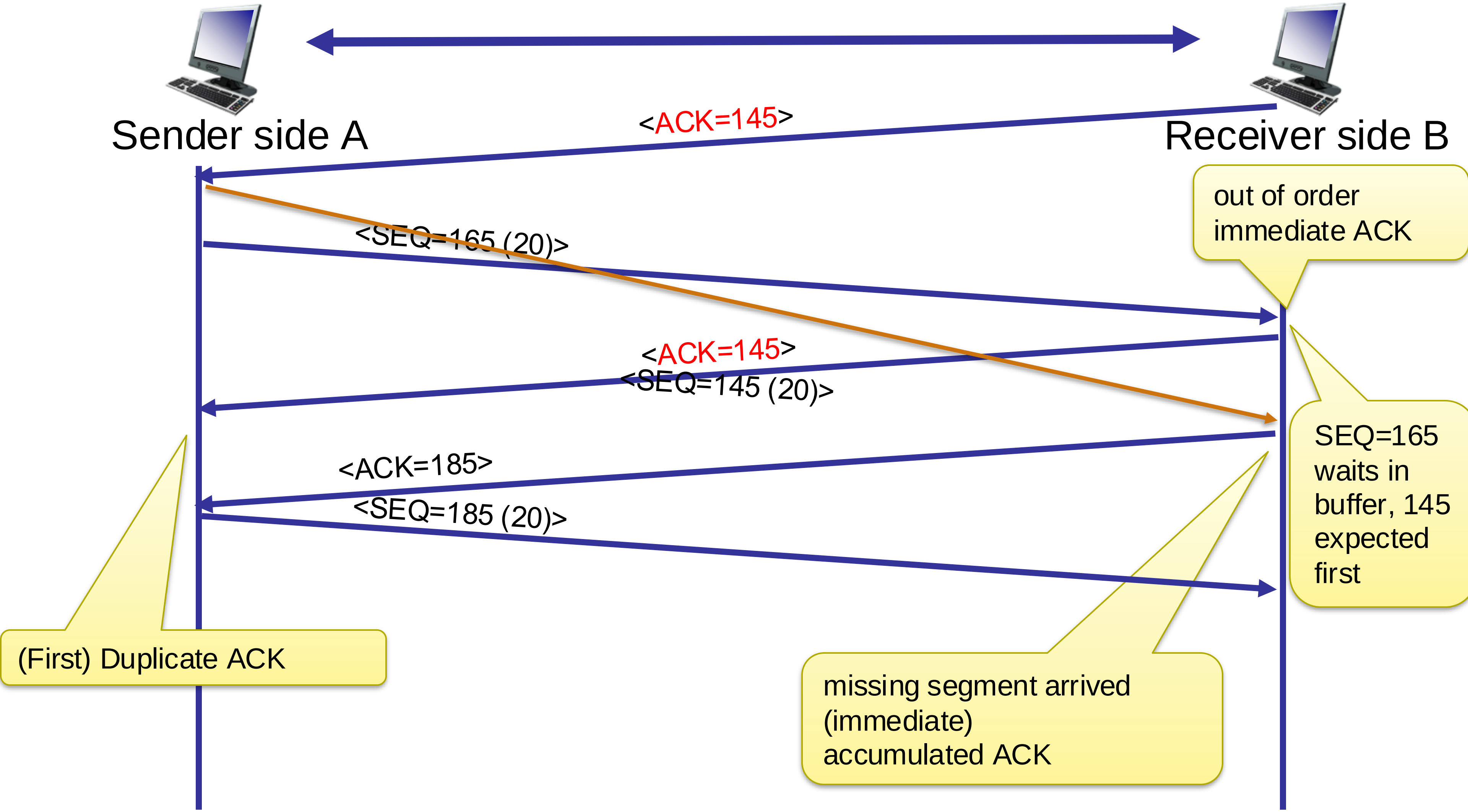


Delayed accumulated ack – real life



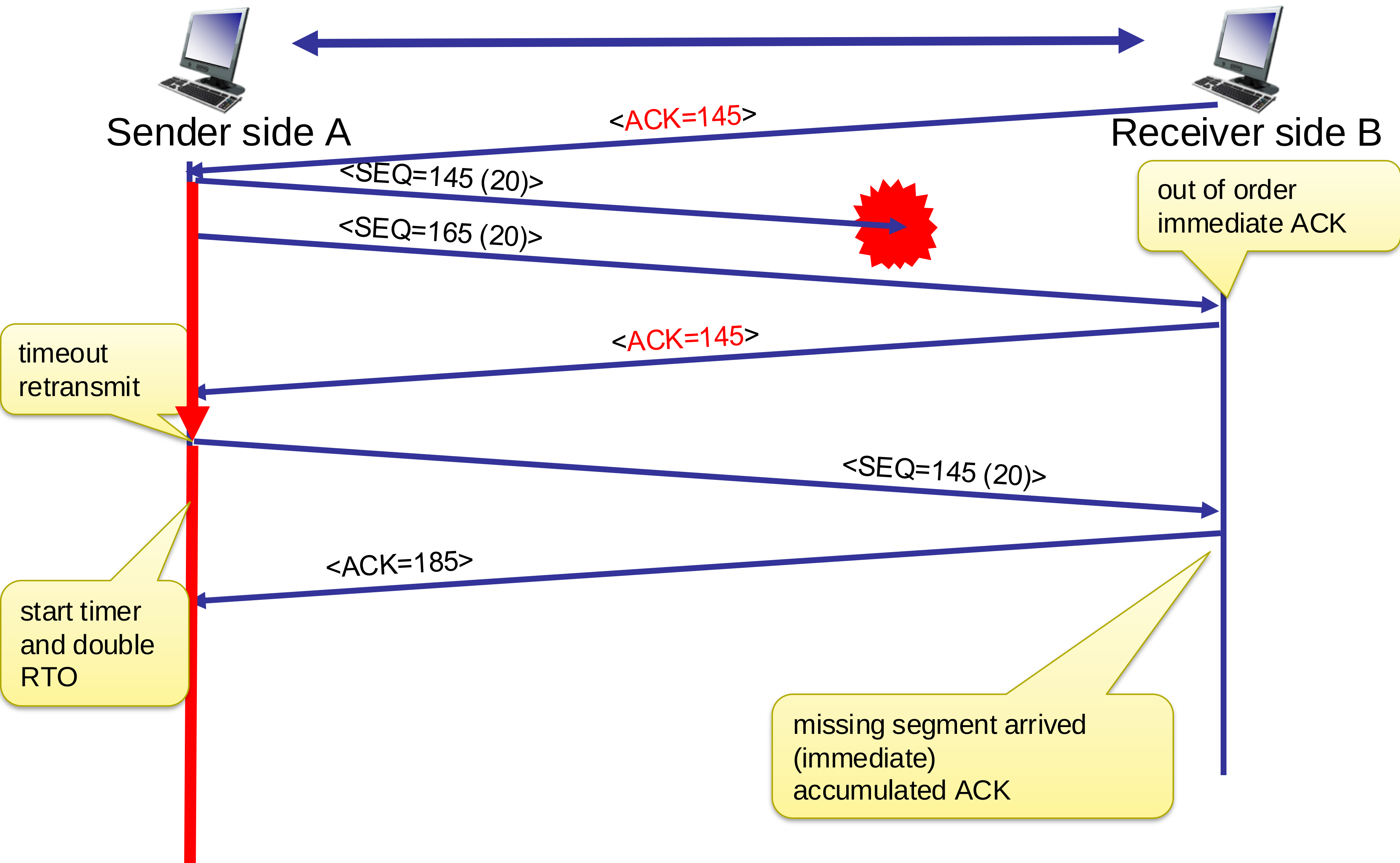
- ACK number indicates that all bytes before the ACK number have been received correctly
- in practice : if a second segment is received, the accumulated ACK is send immediately

Avoid retransmission



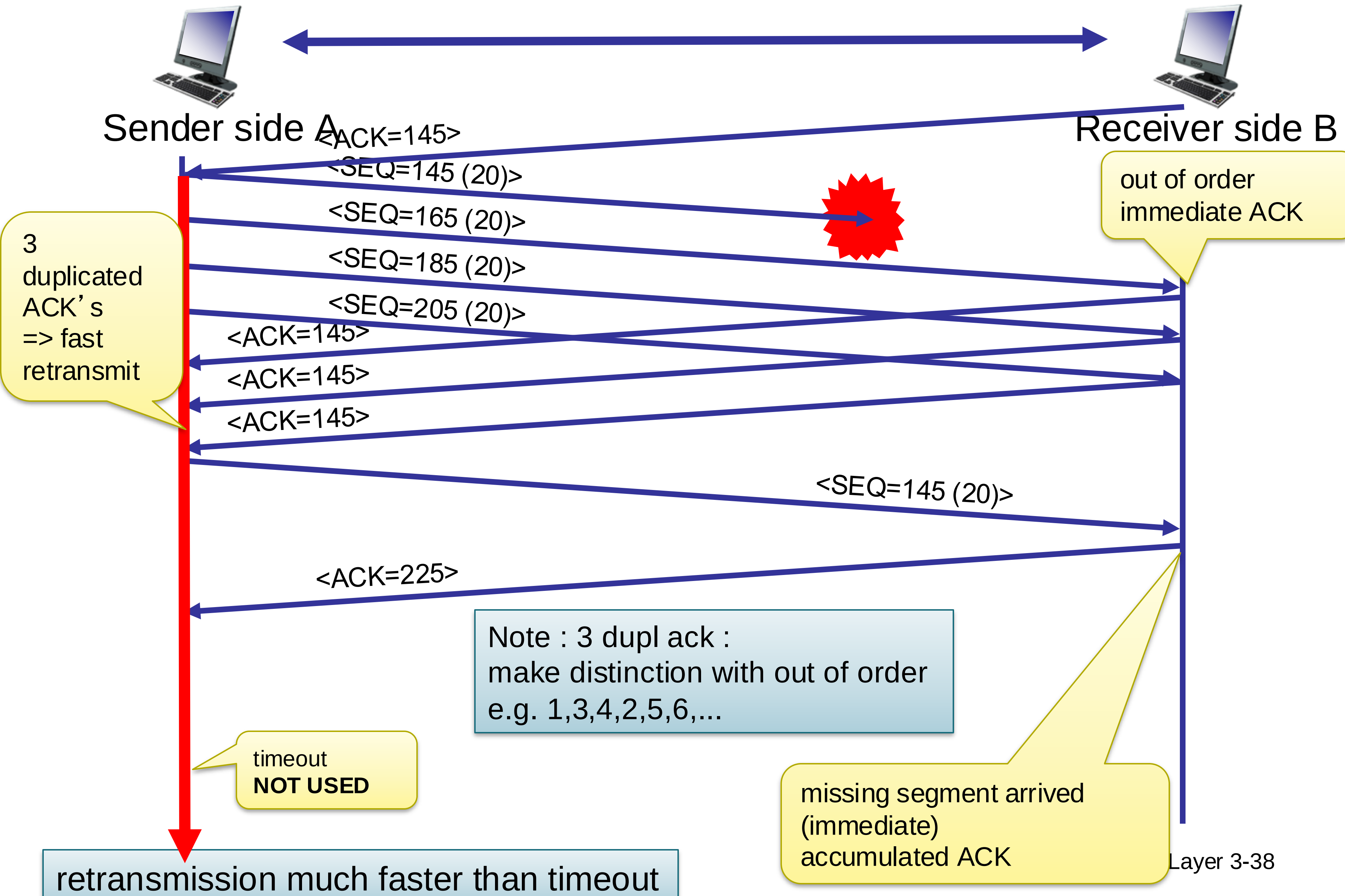
Different delay for segments
=> re-ordered but **no need to resend**

Avoid retransmission (2)



not all segments have to be retransmitted !

Fast retransmission



TCP ACK generation

Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single accumulated ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	Immediately send duplicate ACK, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediate send ACK, provided that segment starts at lower end of gap

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

[3.4 principles of reliable data transfer]

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- **flow control**
- connection management

[3.6 principles of congestion control]

[3.7 TCP congestion control]

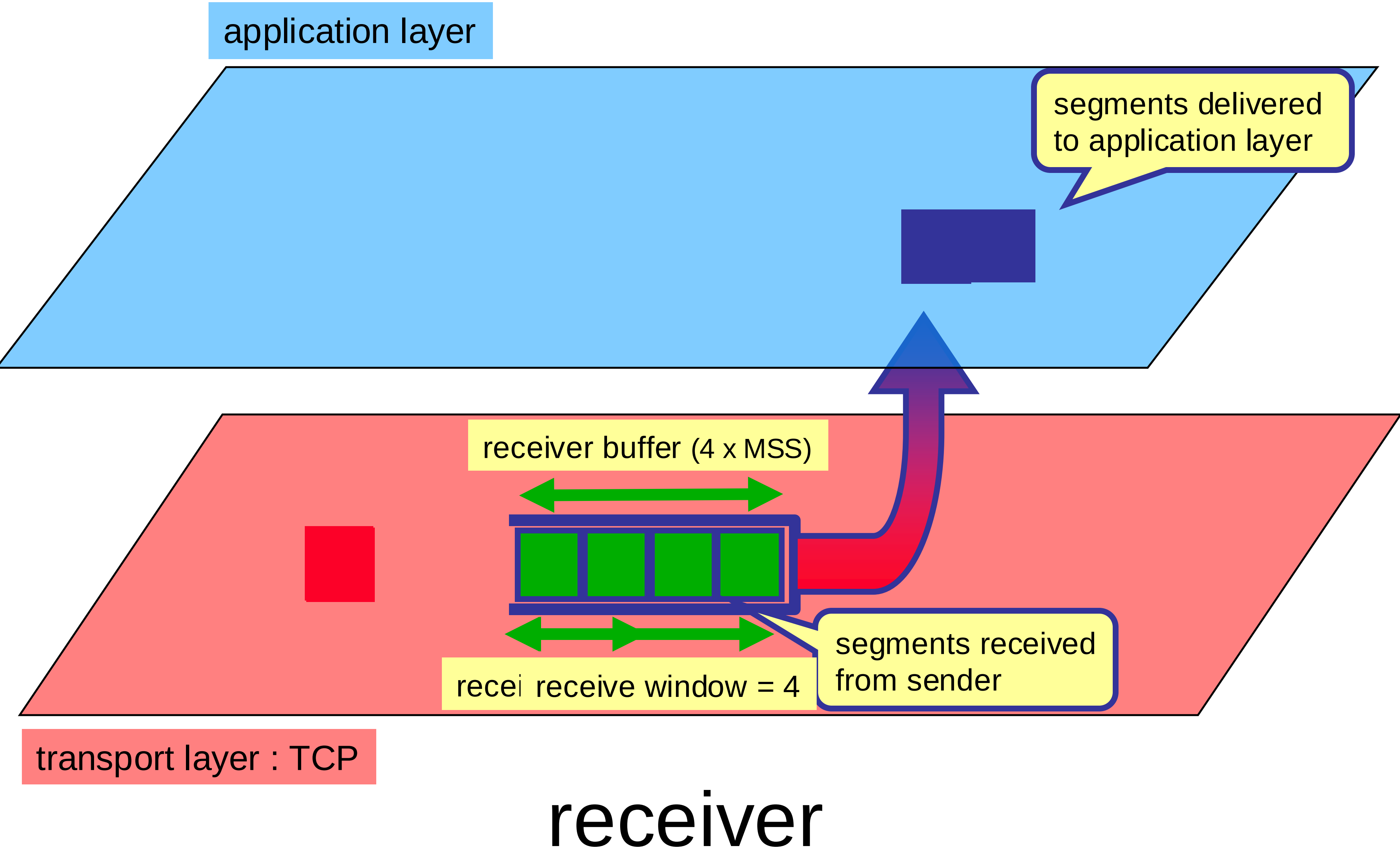
3.8 Evolving transport-layer functionality

Flow control

Flow Control :
receiver limits sender speed
based on own buffer filling

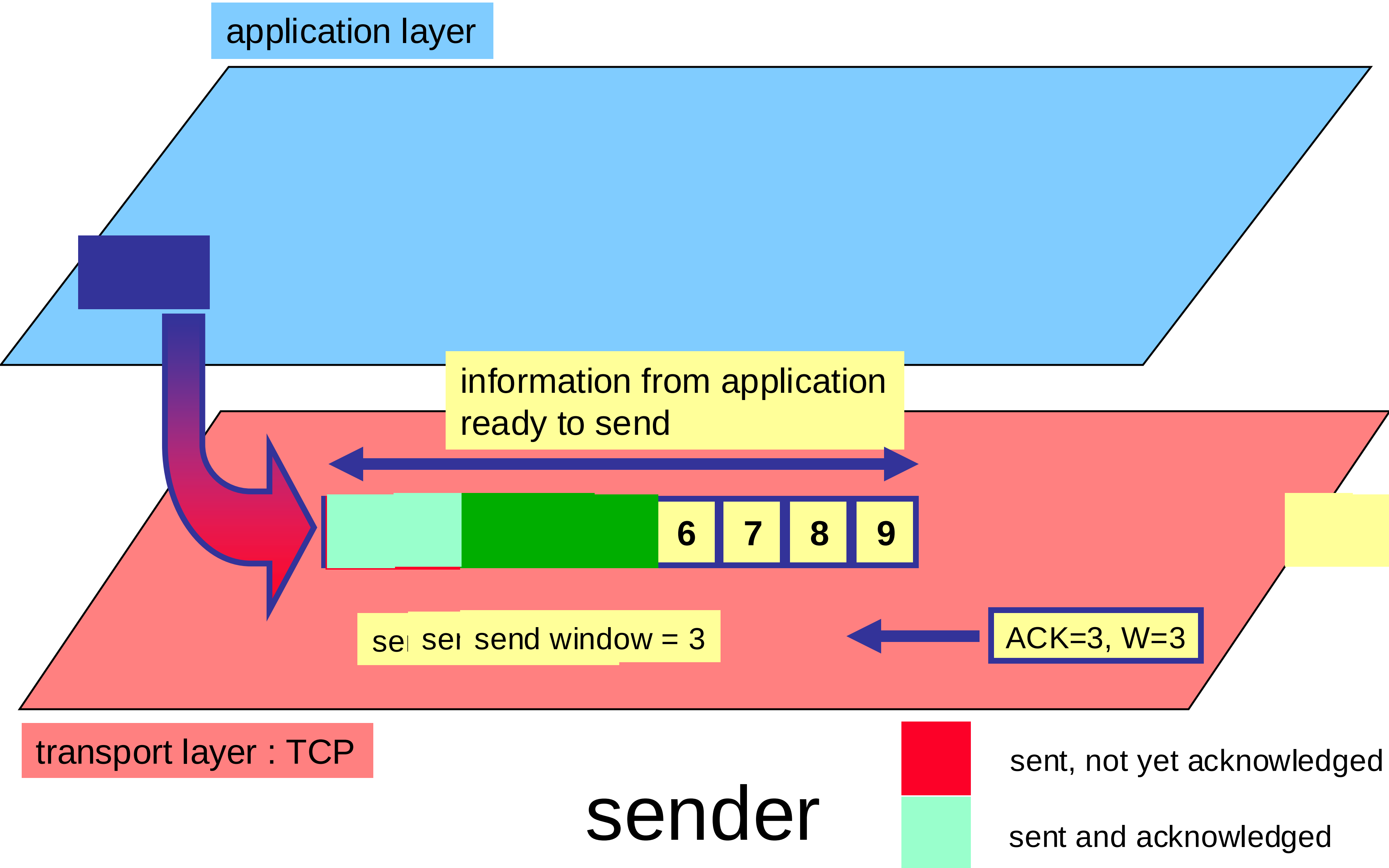
- slow receiver may not be able to cope with segment stream from fast sender
- receiver will measure buffer filling
- receiver will advertise to sender its free buffer space (advertise receive window : **`RcvWindow`**)
- sender will limit outgoing traffic
- ONLY terminals participate in flow control (layer 4!)

Flow control : receiver side



receive window = available buffer at receiver

Flow control : sender side

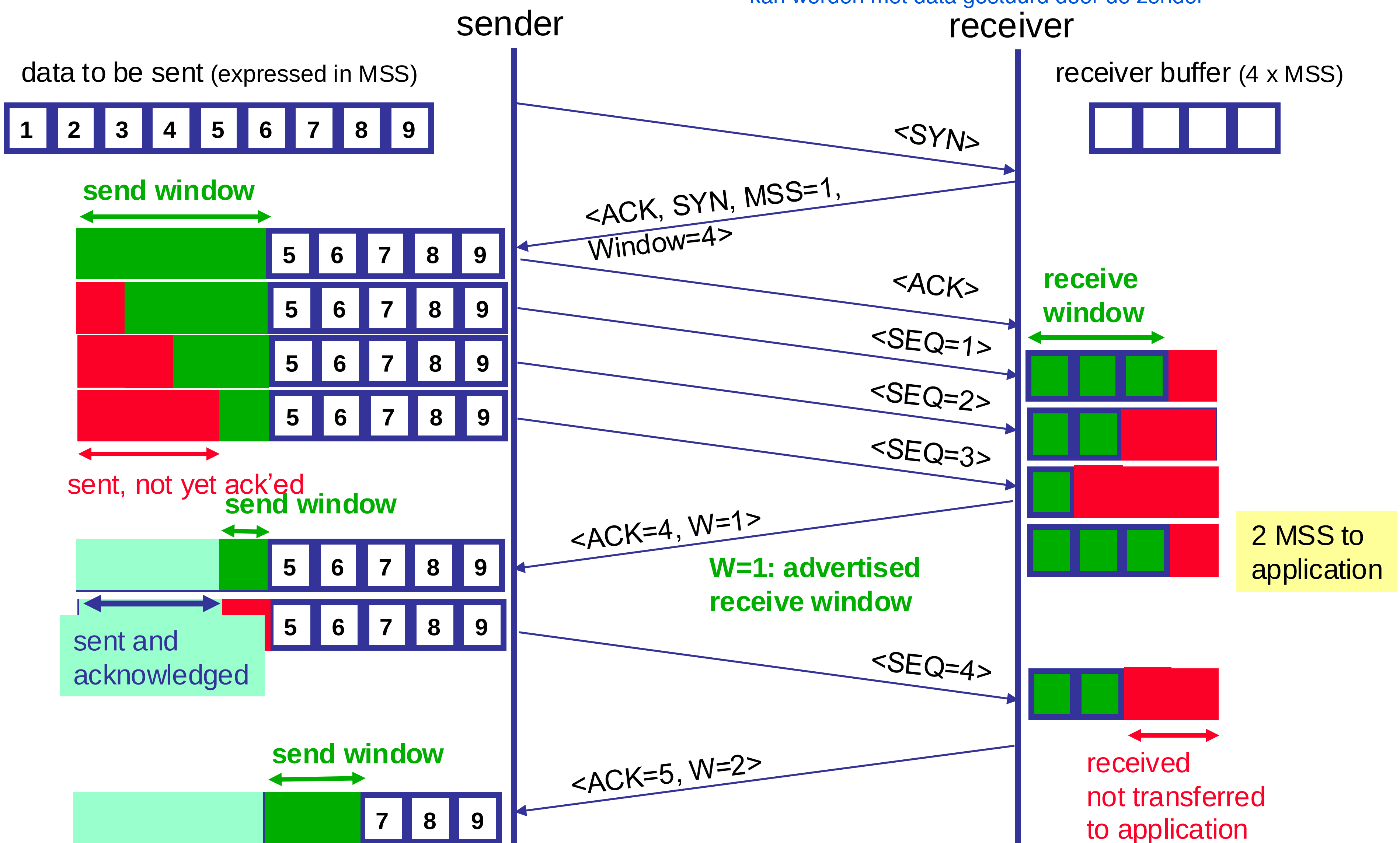


send window = amount of data still allowed to send

Flow control : example

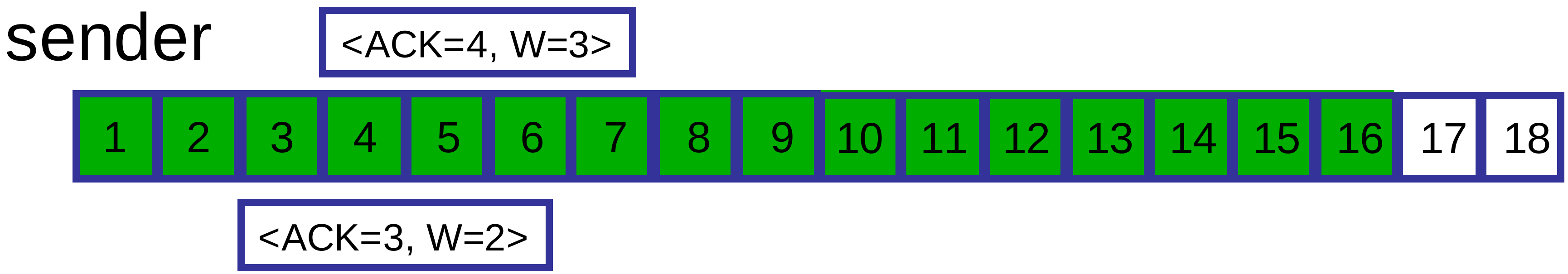
send window toont hoeveel data de zender mag sturen voordat hij moet wachten op een ack van de ontvanger

receive window toont aan hoeveel de buffer nog gevuld kan worden met data gestuurd door de zender



zero window <ACK=5, W=0> will stop communication (no more window updates) ?
➔ regular probing from sender (use persistency timer)

Flow control : sliding window



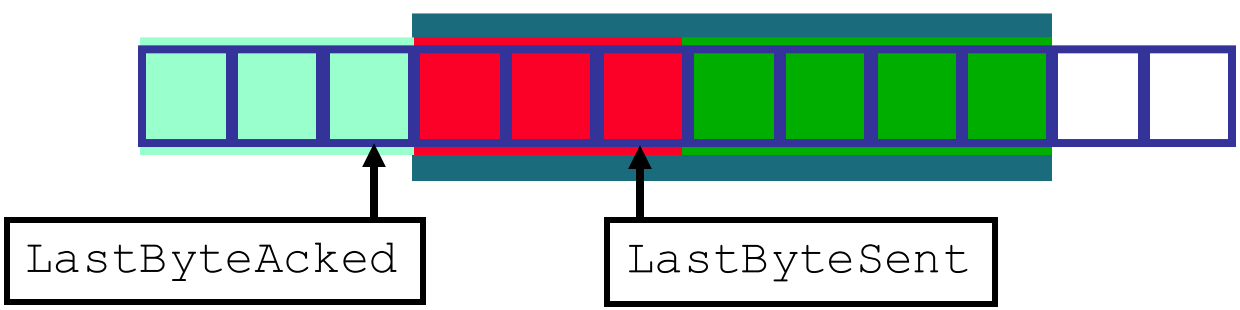
SEND WINDOW =

(advertised) **receive window** (RcvWindow)

MINUS

number of sent but not yet acknowledged bytes

(LastByteSent – LastByteAcked)



allowed to send if :

LastByteSent – LastByteAcked < RcvWindow or SndWindow > 0

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

[3.4 principles of reliable data transfer]

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

[3.6 principles of congestion control]

[3.7 TCP congestion control]

3.8 Evolving transport-layer functionality

UDP: User Datagram Protocol

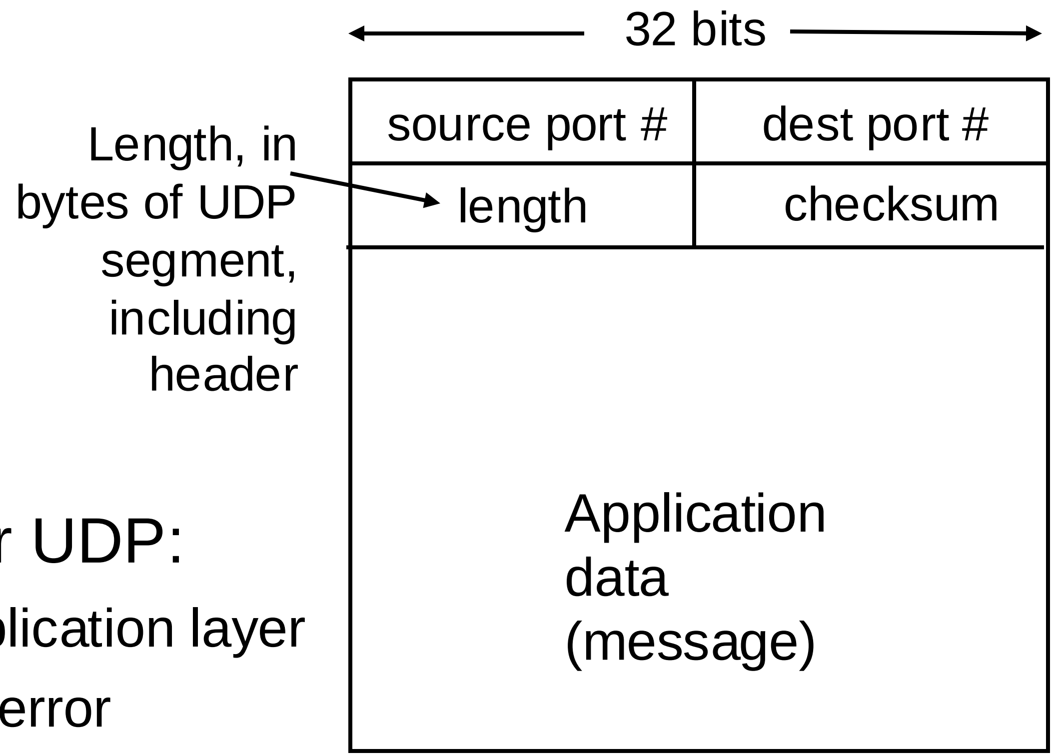
- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header size
- no congestion control: UDP can blast away as fast as desired

UDP

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!



UDP segment format

UDP Checksum

Goal*:* detect "errors" (example, flipped bits) in transmitted segment

Sender:

- treat segment contents, including header fields, as sequence of 16-bit integers
- checksum: addition (one's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?* More later ….

Internet Checksum: Example

example: add two 16-bit integers

		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

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

[3.4 principles of reliable data transfer]

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

[3.6 principles of congestion control]

[3.7 TCP congestion control]

3.8 Evolving transport-layer functionality

Evolving transport-layer functionality

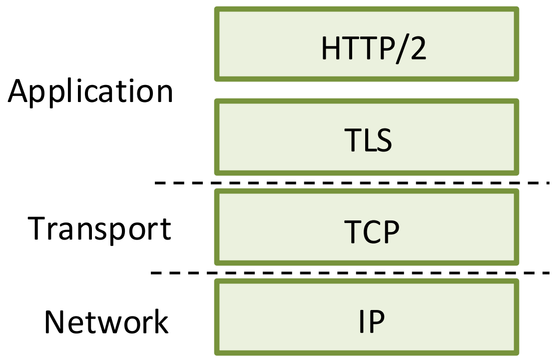
- TCP, UDP: principal transport protocols for 40 years
- different "flavors" of TCP developed, for specific scenarios:

Scenario	Challenges
Long, fat pipes (large data transfers)	Many packets "in flight"; loss shuts down pipeline
Wireless networks	Loss due to noisy wireless links, mobility; TCP treat this as congestion loss
Long-delay links	Extremely long RTTs
Data center networks	Latency sensitive
Background traffic flows	Low priority, "background" TCP flows

- moving transport–layer functions to application layer, on top of UDP
  - HTTP/3: QUIC

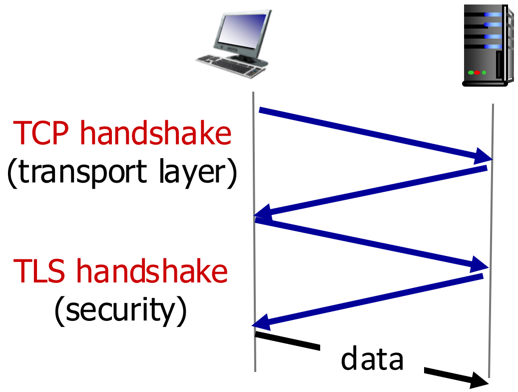
QUIC: Quick UDP Internet Connections

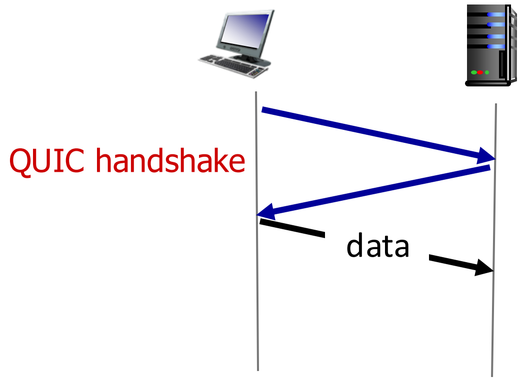
- application-layer protocol, on top of UDP
  - increase performance of HTTP
  - deployed on many Google servers, apps (Chrome, mobile YouTube app)



HTTP/2 over TCP

QUIC: Connection establishment





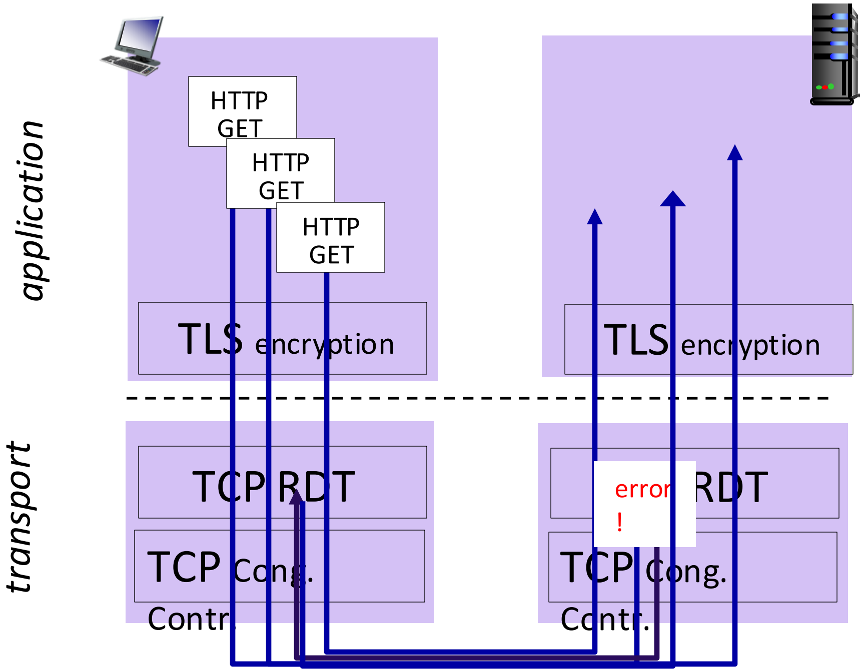
TCP (reliability, congestion control state) +
TLS (authentication, crypto state)

- 2 serial handshakes

QUIC: reliability, congestion control, authentication, crypto state

- 1 handshake

QUIC: streams: parallelism, no HOL blocking



(a) HTTP 1.1

Chapter 3