1.2. TCP Advanced Concepts

PROFESSORS:

Rubén Santiago Montero

Eduardo Huedo Cuesta

OTHER AUTHORS:

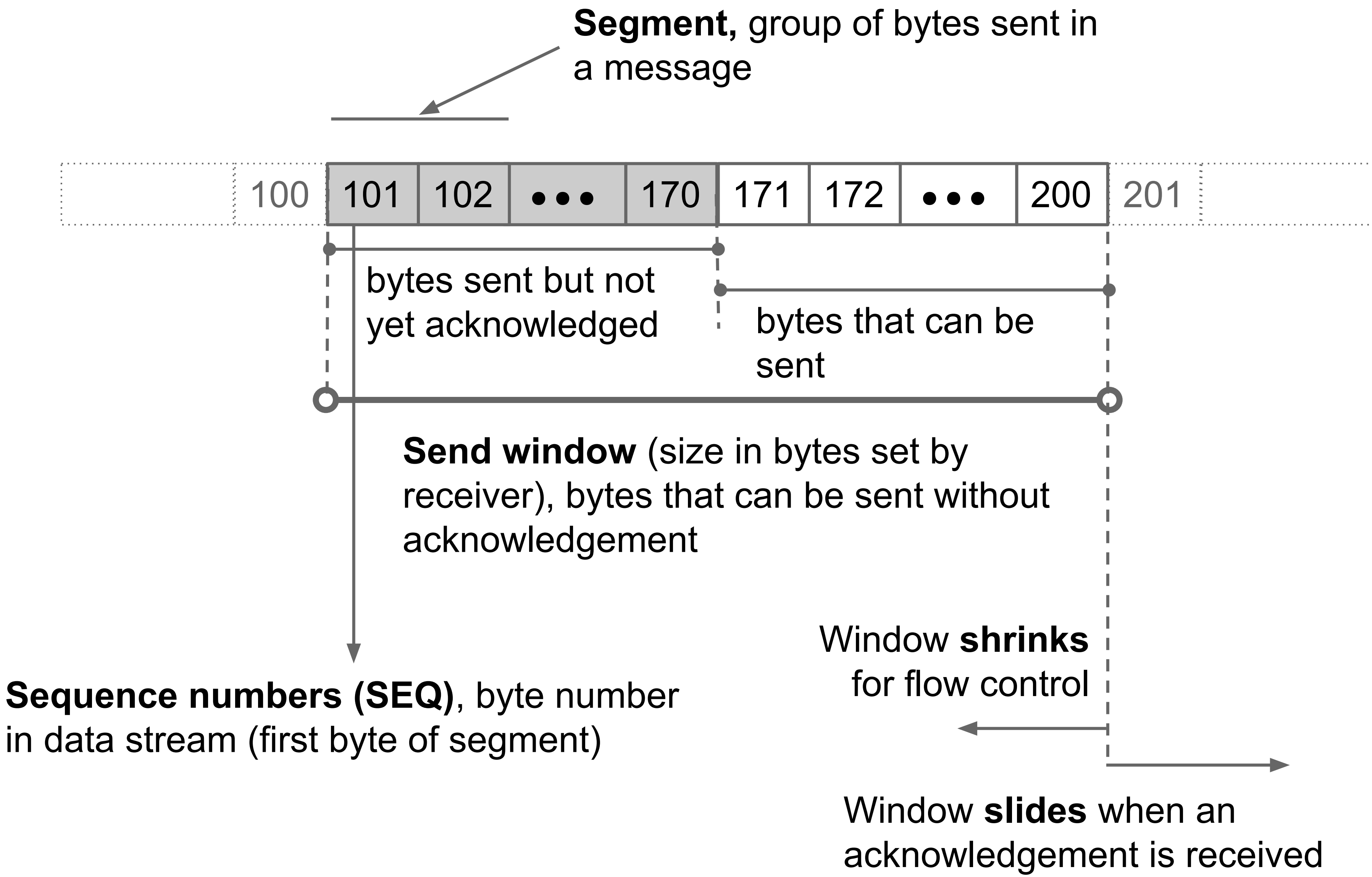
Rafael Moreno Vozmediano

Juan Carlos Fabero Jiménez

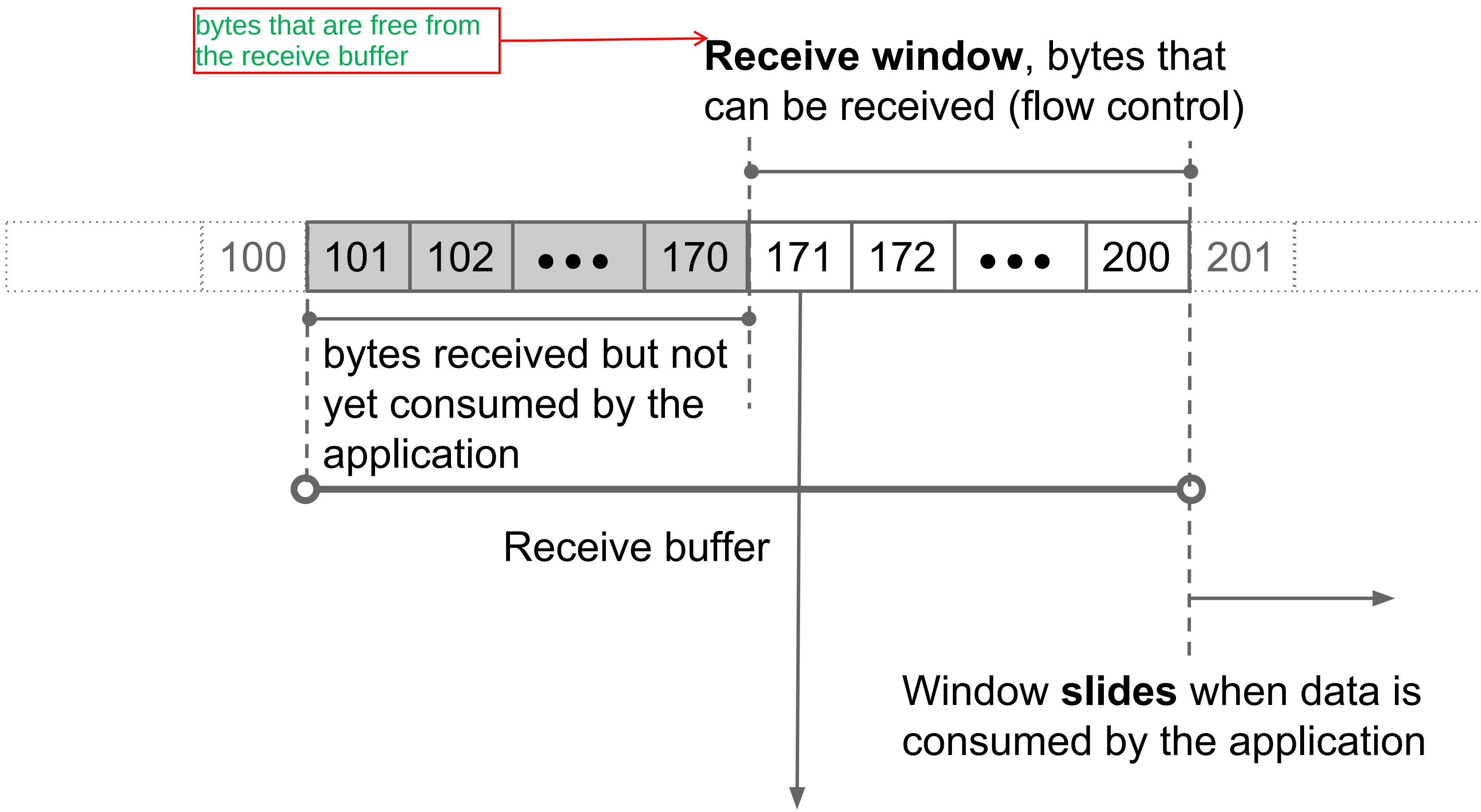
TCP: Characteristics

- Transfer unit: TCP segment
- Connection-oriented and reliable. It defines three phases for the transmission:
    - Connection establishment
    - Data transfer
    - Connection termination
- Error control mechanisms (sliding window) with:
    - Checksum codes
    - Segment numbering
    - Cumulative and (optionally) selective acknowledgements, with piggybacking
    - Retransmission of bad or lost segments
    - Timers
- Services offered by TCP:
    - Process-to-process logical communication, using port numbers
    - Data flow (stream) control for sender and receiver
    - Connection-oriented, reliable transmission
    - Full-duplex communication and multiplexing

Sliding Window: Send Window



Sliding Window: Receive Window



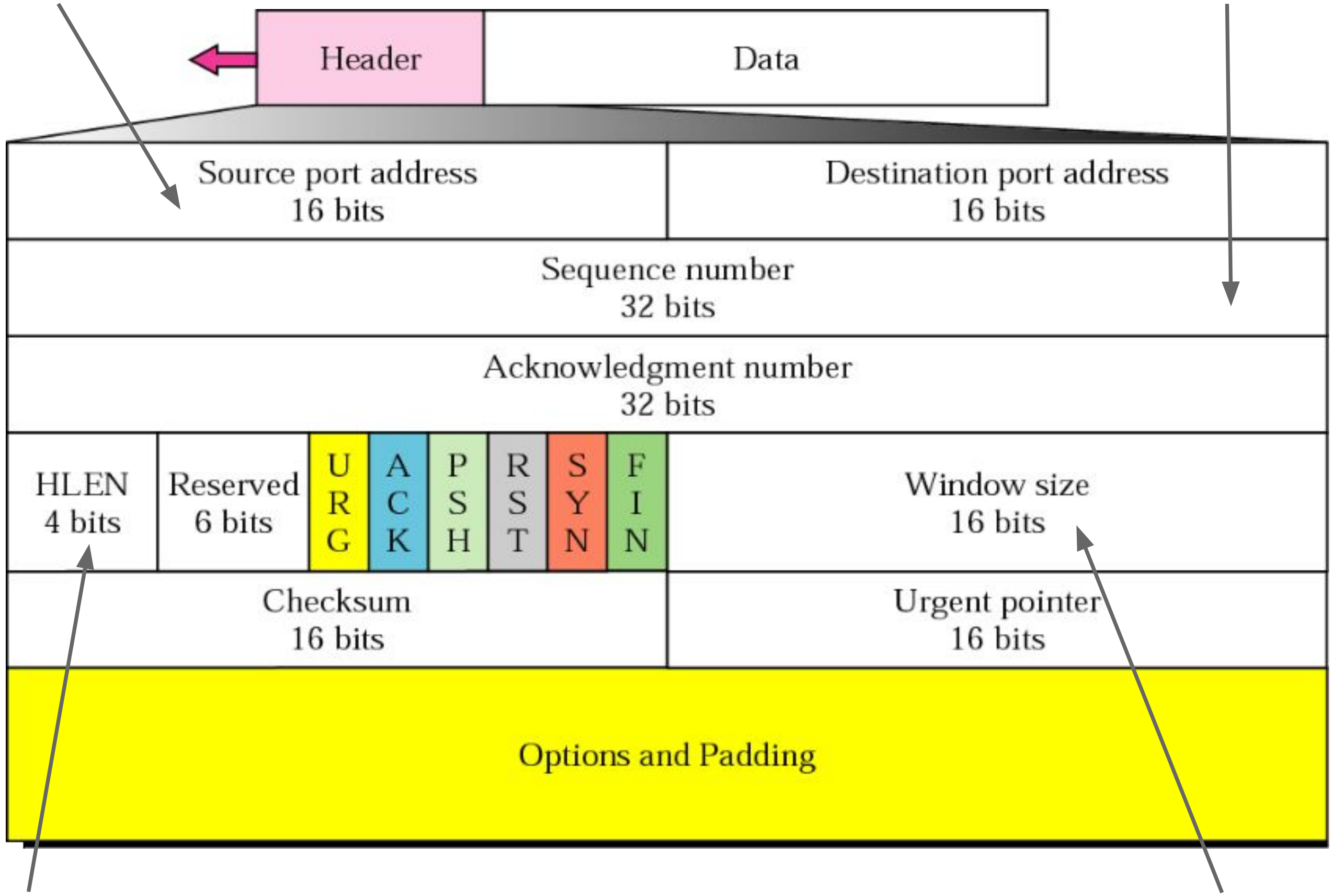
Acknowledgement numbers (ACK), number of the first byte in data stream that is expected from the sender

- Cumulative, they acknowledge all previous bytes
- Overlapped with data transmission (piggybacking)

TCP Segment Format

Ports identify the connection ends

Sequence and **acknowledgement numbers** refer to bytes in data stream



Header length in 32-bit words (20-60 bytes)

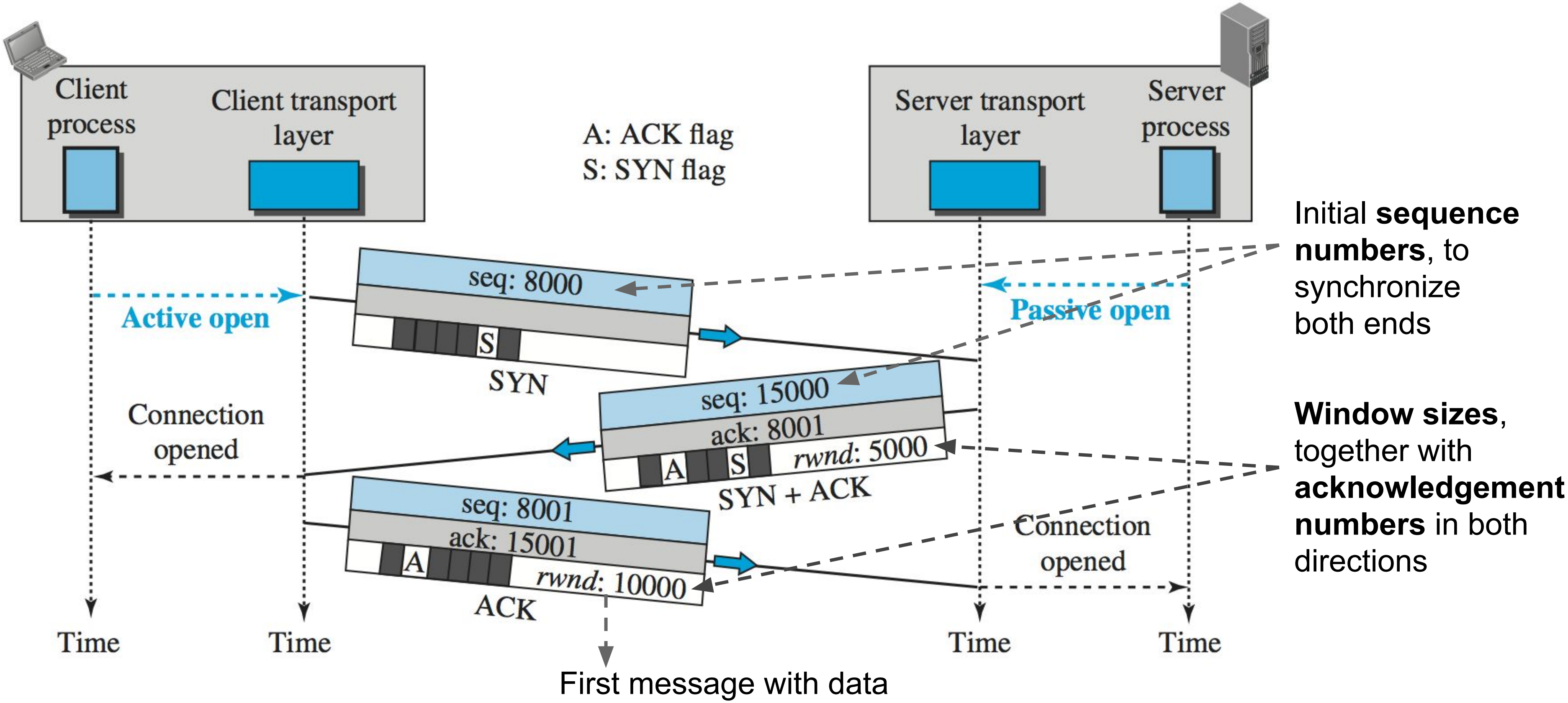
Window size in bytes

TCP Segment Format

Flags (6 bits):

- **SYN**: Used in connection establishment to synchronize initial sequence numbers
- **FIN**: Used in connection termination
- **ACK**: The segment has a valid acknowledgement number (ACK=1). All segments in a TCP connection, but the first, have ACK=1
- **RST**: Used to abort a connection
- **PSH**: Data must be delivered immediately to the receiving application (PSH=1), or it can be buffered (PSH=0)
- **URG**: The segment transports urgent data (URG=1) from the first byte to the byte specified in the **Urgent pointer** field
  - TCP notifies the application about urgent data (SIGURG signal)
  - The application, and not TCP, should manage urgent data appropriately

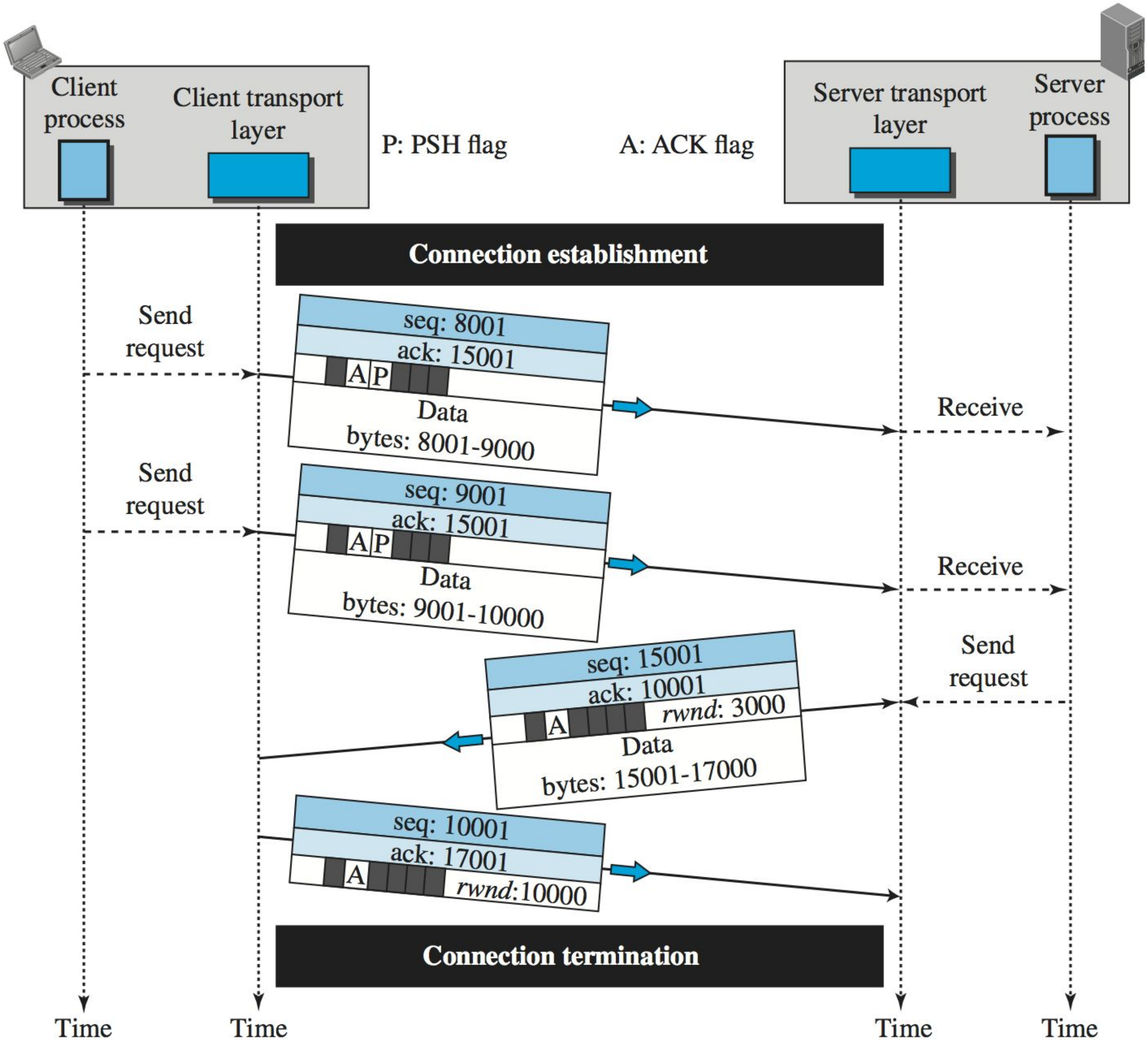
Connection Phases: Establishment (3-way)



TCP SYN Flooding Attack

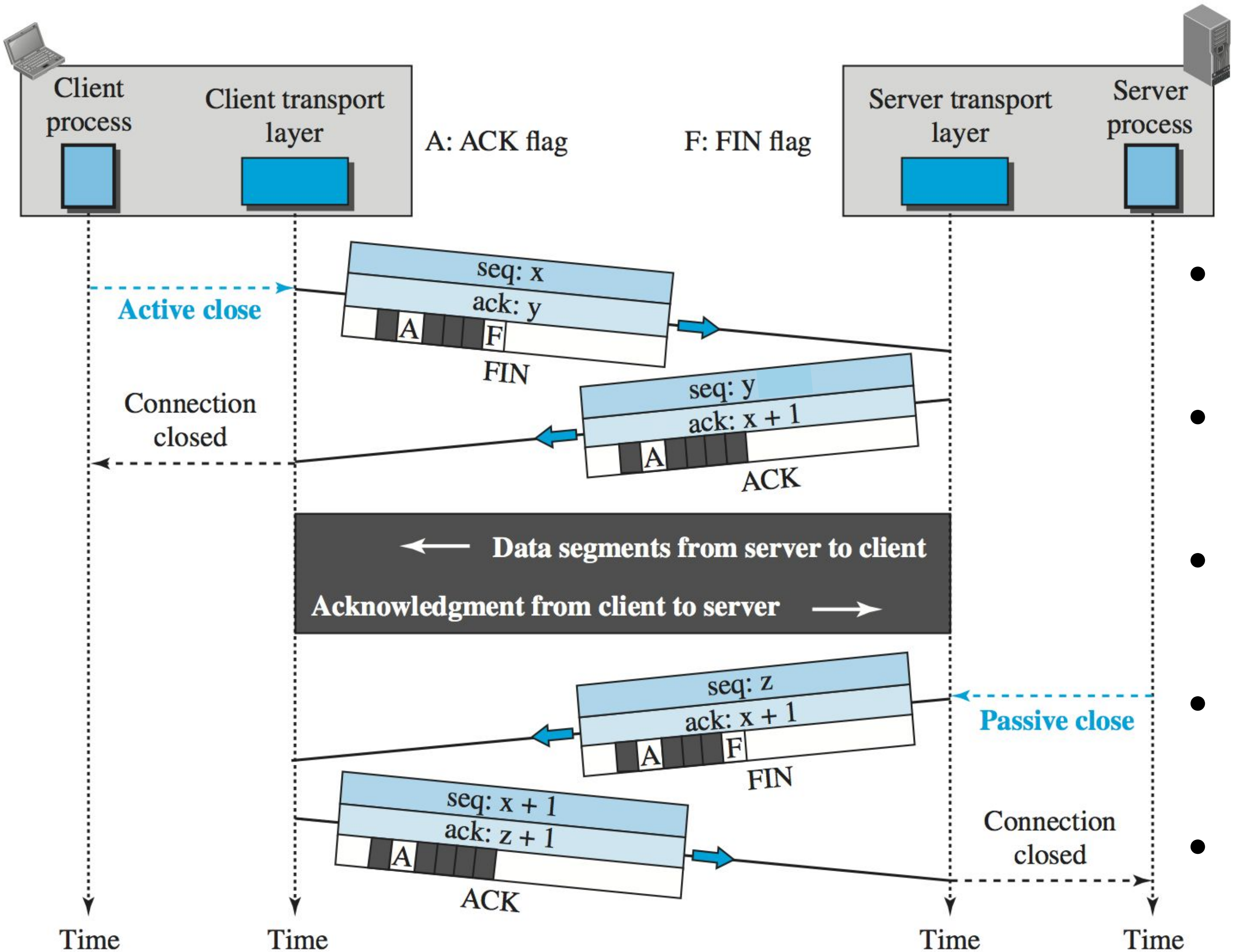
- Lots of TCP segments with the SYN flag enabled are sent to exploit a protocol vulnerability that can saturate the server or prevent legitimate connections (DoS), as TCP allocates resources to each *half-open* connection. Countermeasures:
    - Limit the number of connections
    - Only accept connections from reliable IP addresses
    - Delay resource allocation using SYN *cookies*

Connection Phases: Data Transfer



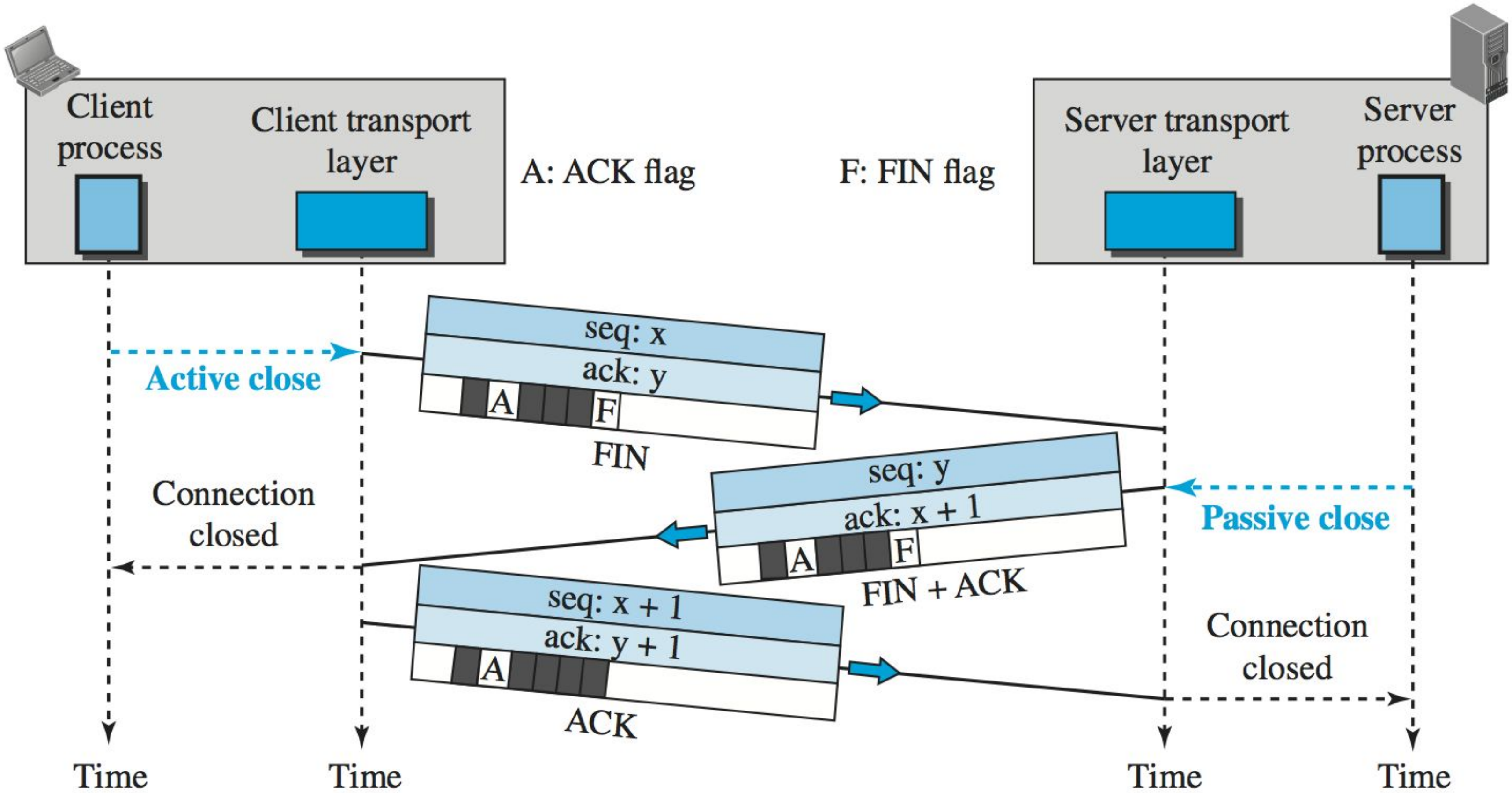
Maximum Segment Size (MSS) is independently set by each end in the Options field

Connection Phases: Termination (4-way)



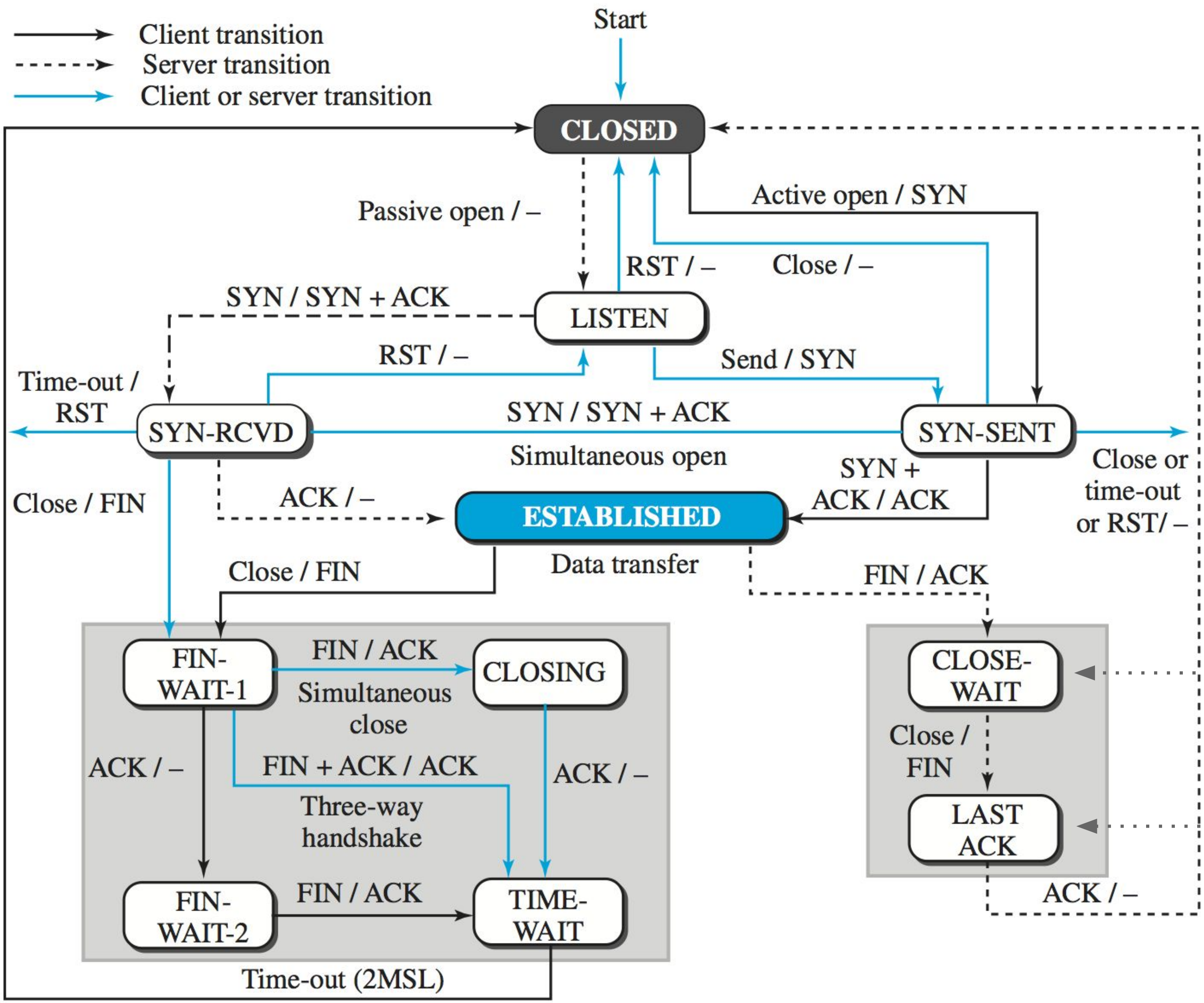
- Client initiates termination with FIN message
- Server acknowledges it, but does not initiate termination
- Client stops sending data, but server continues sending data
- Server initiates termination with FIN message
- Client acknowledges it

Connection Phases: Termination (3-way)



- Both ends stop sending data
- FIN messages may contain data. A sequence number is always consumed, since they must be acknowledged
- Last ACK message doesn't carry data

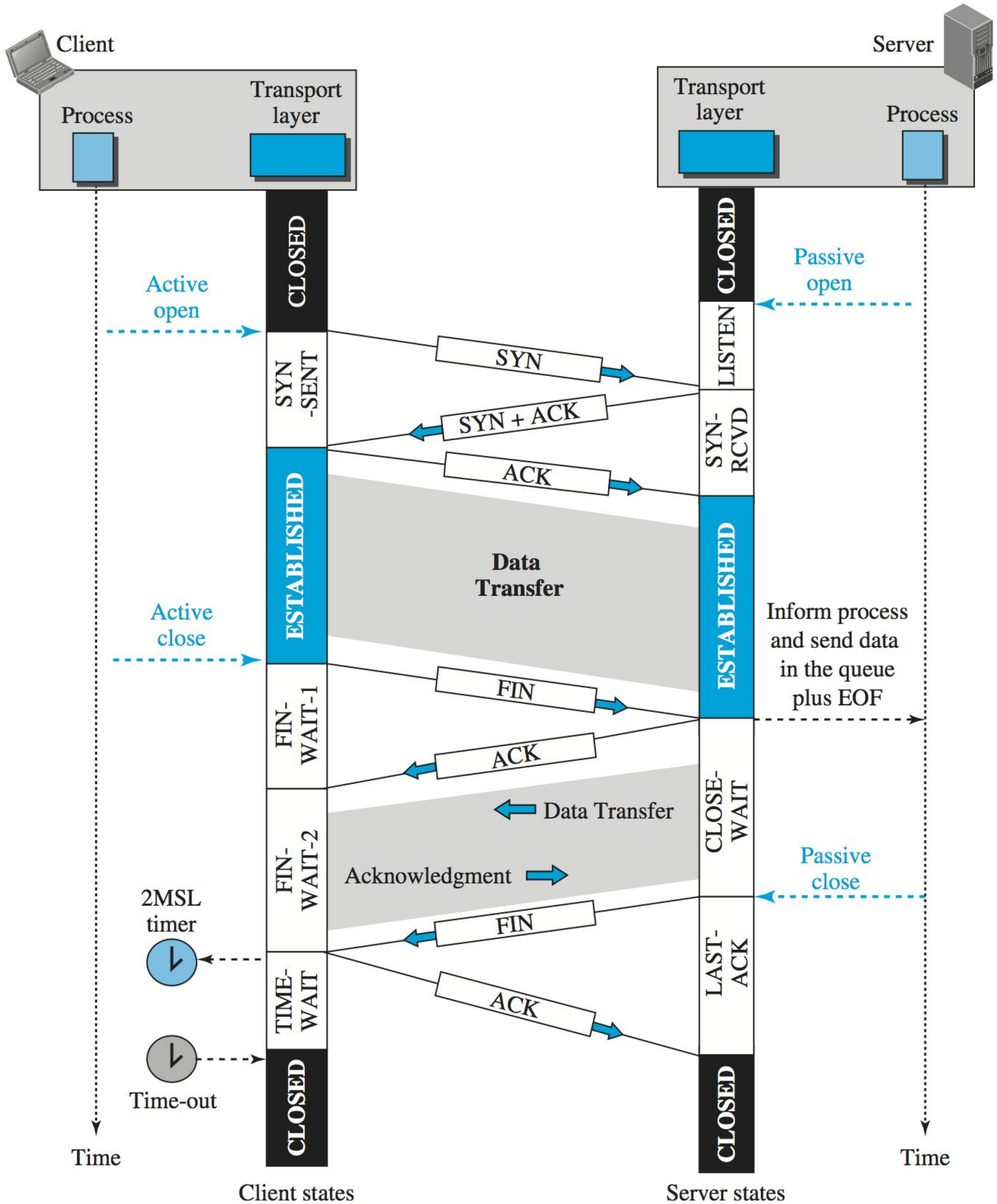
Connection Phases: State Machine

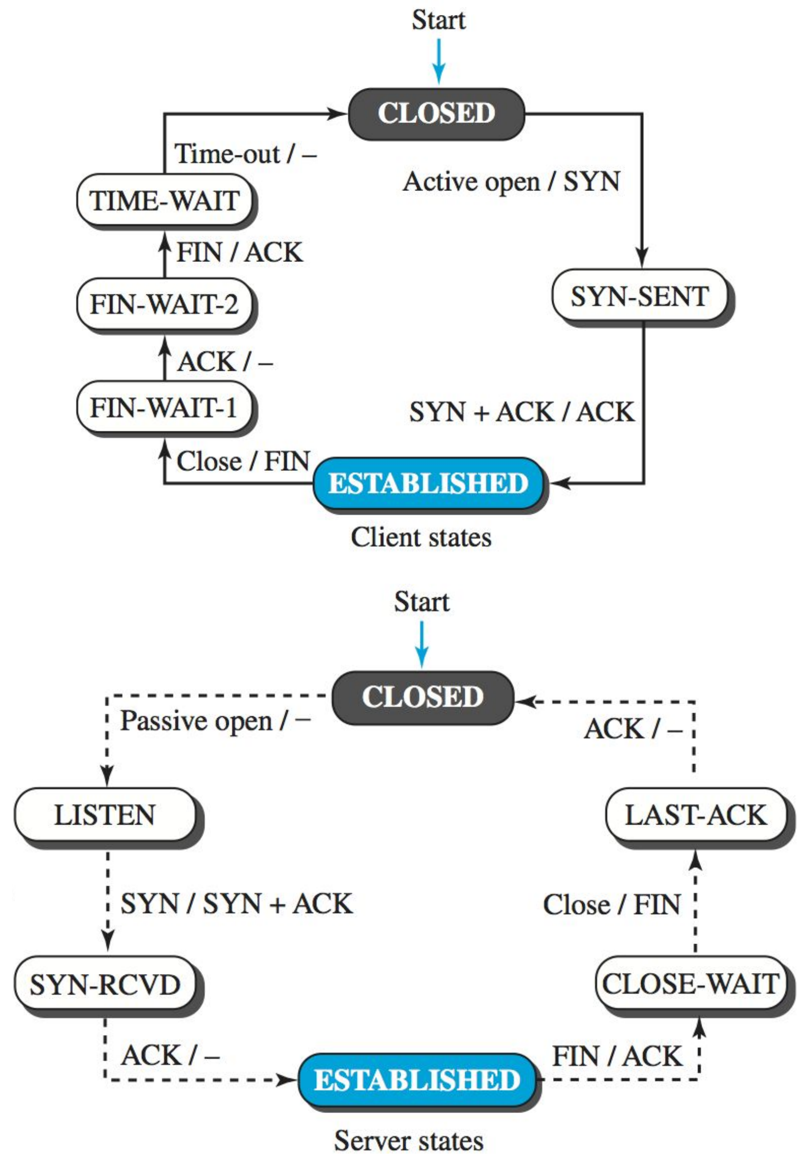


Waiting for the application to close

Wait for last ACK

Connection Phases: State Machine





Error Control: Acknowledgements

- Error control is done using the sliding window mechanism that allows managing:
  - The reception of duplicate segments
  - The retransmission of bad or lost segments
  - The reception of out-of-order segments

Rules for acknowledgements:

1. Acknowledgement must be included in data segments (piggybacking), indicating the next byte expected to receive
2. If there is no data to send, acknowledgement of in-order segments is delayed a maximum of 500 ms in order to be included in a data segment

   Delayed ACKs

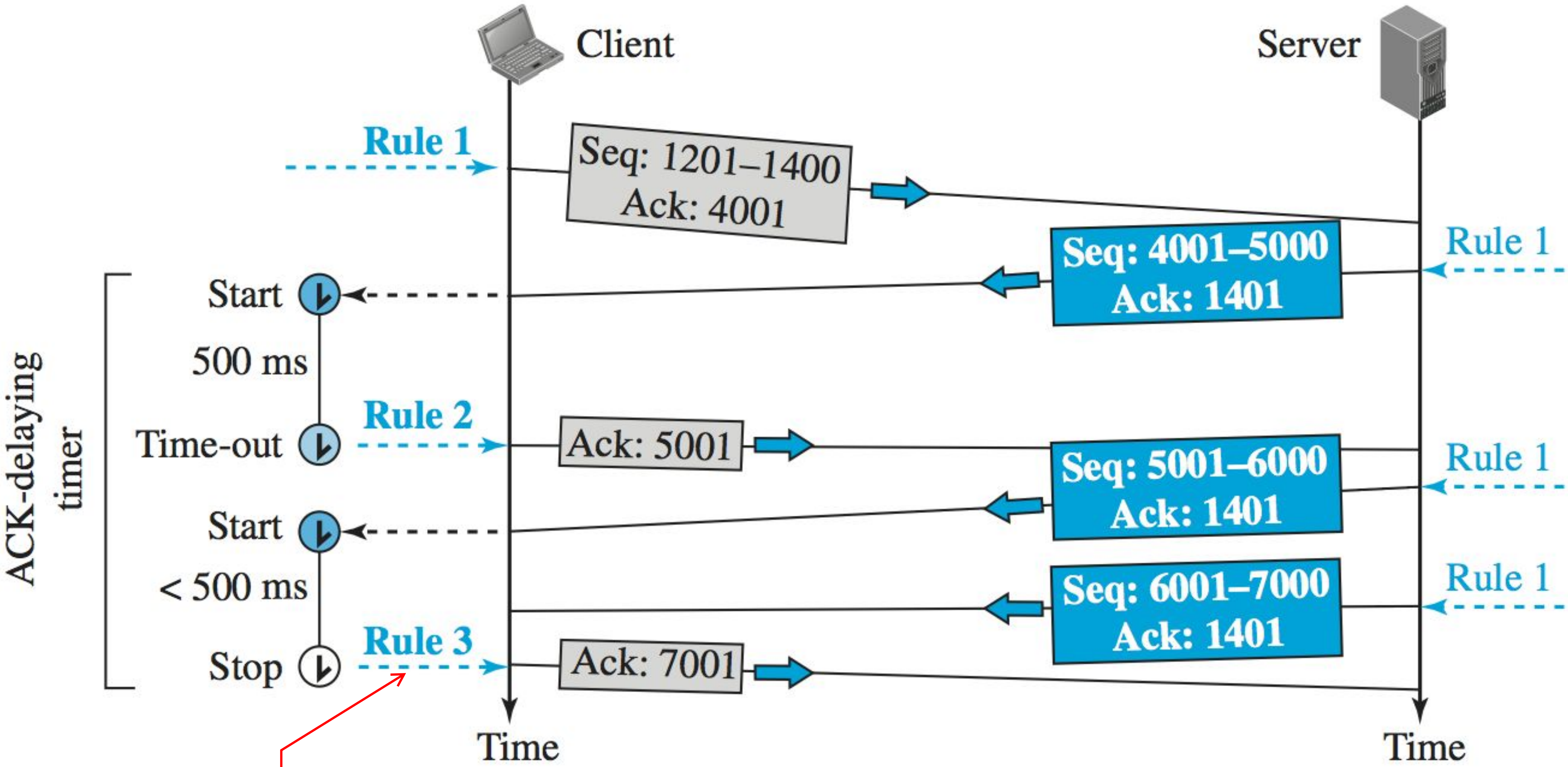
   can be disabled
3. There should not be more than two in-order unacknowledged segments
4. Out-of-order segments producing gaps are immediately acknowledged (this leads to fast retransmission, discussed later)
5. Out-of-order segments filling gaps are immediately acknowledged
6. Duplicate segments are acknowledged to avoid problems with lost ACKs

Optional

- Selective acknowledgement (SACK) of out-of-order segments
  - Don’t replace ACK, they are informative for the sender

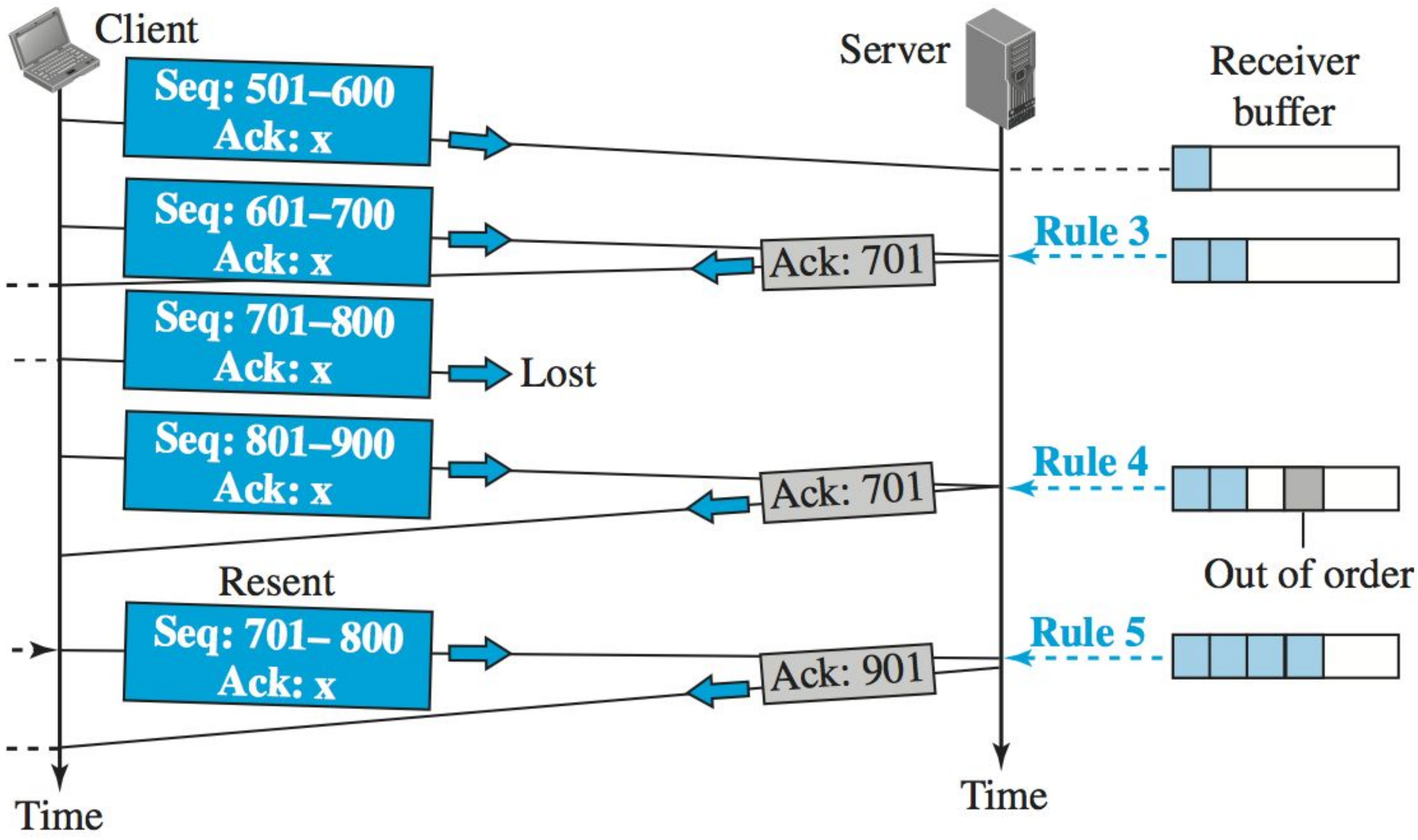
    to avoid the retransmission of out-of-order segments
  - Implemented as an TCP option

Error Control: Transmission without Errors



You can delay the acknoledgement of the first segment but not the second, that's why the ack of the second one is done immediately

Error Control: Out-of-order Reception

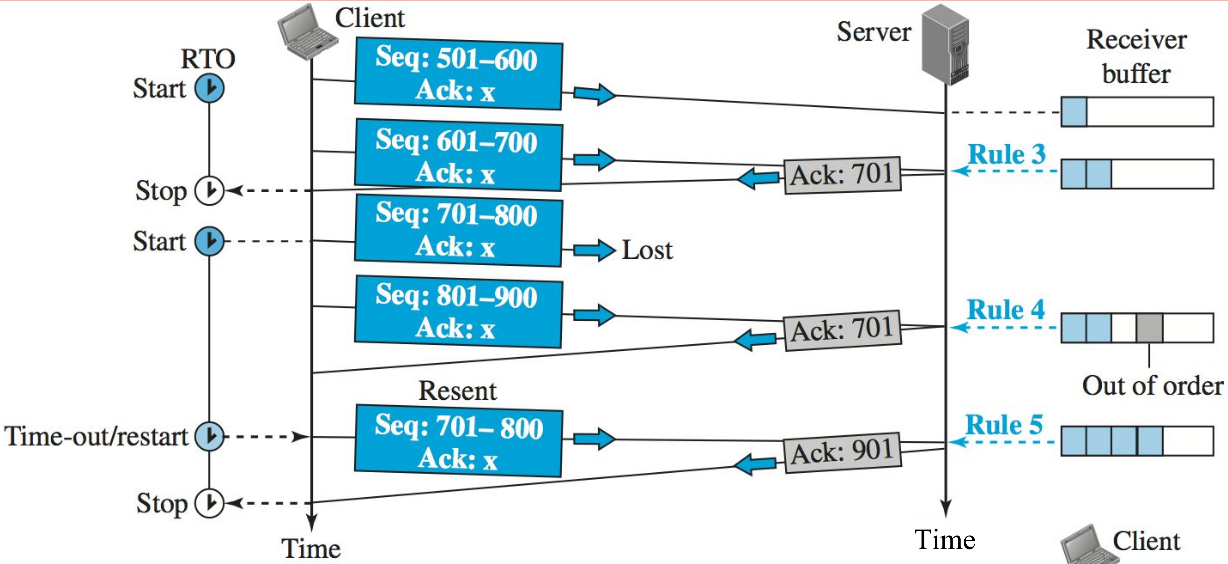


bc of the lost one, it's not the expected segment

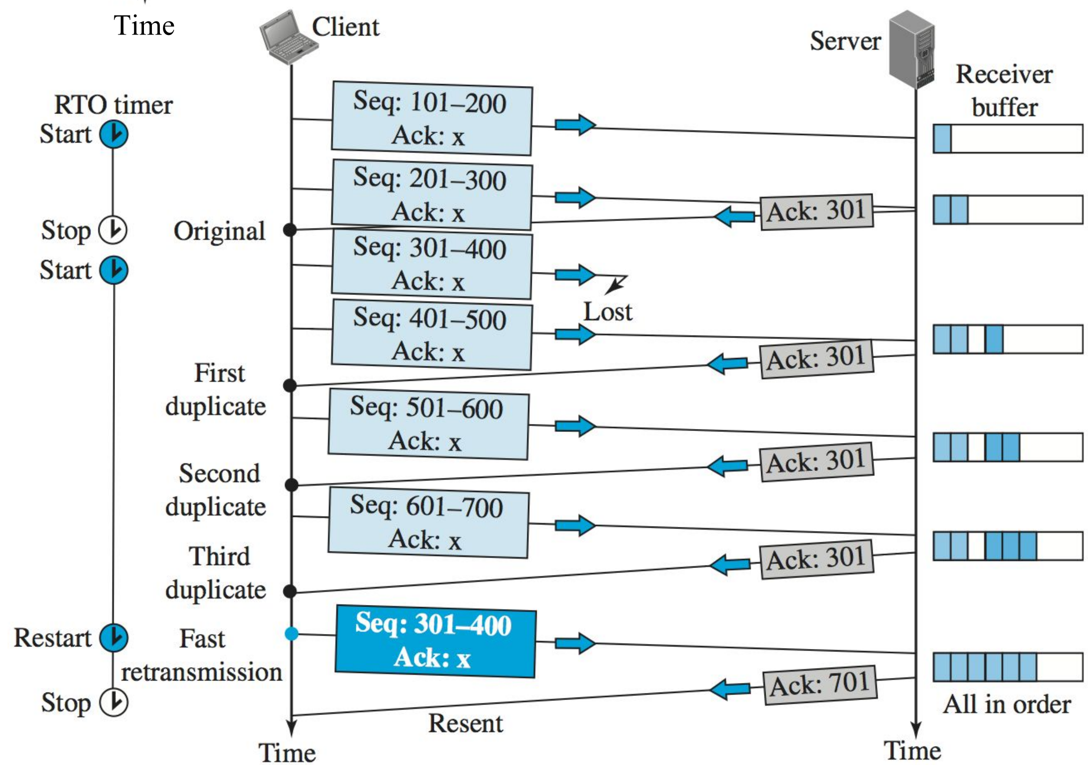
Error Control: Retransmission

- The capacity to retransmit bad or lost segments is the core of error control
- TCP provides two retransmission mechanisms:
  - **Retransmission Time-Out (RTO)** enabled when a segment is sent
    - TCP implementations should only use one retransmission timer (see RFC 6298)
    - If the timer times out, the first unacknowledged segment is retransmitted
    - There are different algorithms to set the RTO, which is dynamic and should be higher than RTT (Round-Trip Time)
  - **Fast retransmit**
    - Retransmission after receiving 3 duplicate ACKs

Error Control: Segment Loss

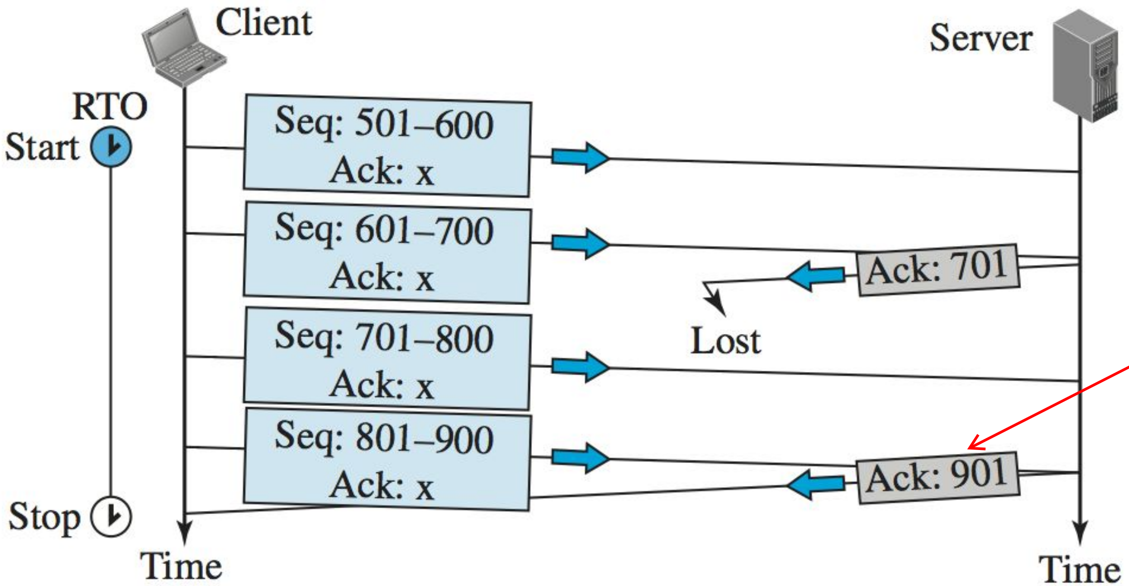


Retransmission time-out



Fast retransmit

Error Control: ACK Loss

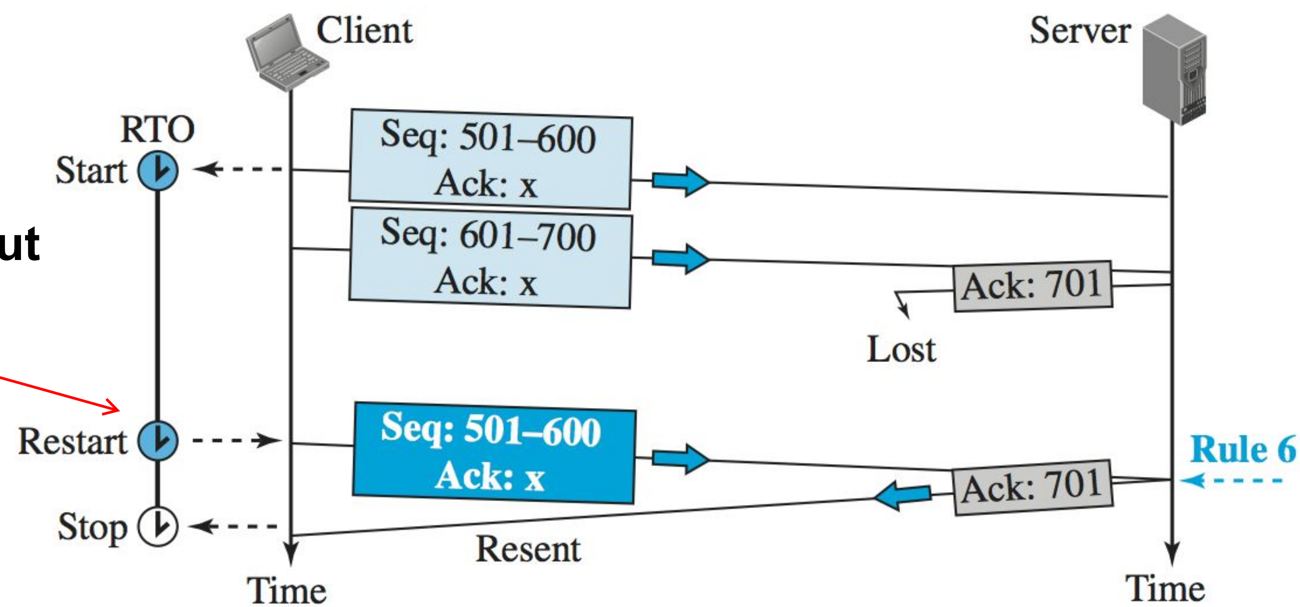


No retransmission time-out

if there is another ack, the second one will acknowledge the previous data

Retransmission time-out

The RTO times-out, so the first segment is sent



TCP Timers

- 4 timers are used to control the connection
  - **Retransmission**
  - **Keepalive** prevents a long idle connection with a death end to remain open
    - A connection can be silent for `tcp_keepalive_time` seconds with no traffic in any direction
    - After this time, a maximum of `tcp_keepalive_probes` probes are sent every `tcp_keepalive_intvl` seconds
    - If no ACK is received for the probes, the connection is closed
    - Example: 2 hours, 10 probes, 75 seconds
  - **TIME-WAIT** is useful in two situations:
    - To resend the final ACK in case it is lost (remote FIN will be retransmitted)
    - To prevent collisions of sequence/acknowledge numbers from two different connections (port and sequence numbers cannot be reused)
    - Usually, 2*MSL (Maximum Segment Lifetime, the time any segment can exist in a network before being discarded). Example: 30, 60, 120 seconds
  - **Persistency timer** deals with a zero-window-size advertisement
    - Recovering from the loss of the ACK segment announcing the nonzero size of the receive window
    - A probe is sent to force its acknowledgement
    - Example: 60 seconds

Retransmission Timer

- The value of RTO is based on the observed Round-Trip Time (RTT) of the network
- RTT can vary dynamically, therefore RTO should adapt to this situation
- Main techniques used to set RTO are:
  - Jacobson’s algorithm (smoothed RTT)
  - Jacobson/Karels’s algorithm (also considers RTT deviation)
  - Karn’s algorithm (does not consider RTT of retransmitted segments)

Retransmission Timer

Measured RTT (RTT_M)

- Time required for the segment to reach the destination and be acknowledged
- Only one RTT measurement can be in progress at any time
- The value of RTT_M can experience high fluctuations

problem --> that's why we use the smoothed RTT

Smoothed RTT (RTT_S)

- Weighted average of RTT_M and the previous RTT_S:

  First measurement: $RTT_S = RTT_M$

  Next ones: $RTT_S = (1 - \alpha)\times RTT_S + \alpha\times RTT_M$, $\alpha<1$ (usually, $\alpha=\frac{1}{8}$)
- Avoids fluctuations in RTT_M

RTT Deviation (RTT_D)

- Most implementations also calculate the RTT deviation:

  First measurement: $RTT_D = RTT_M/2$

  Next ones: $RTT_D = (1 - \beta)\times RTT_D + \beta\times|RTT_S - RTT_M|$, $\beta<1$ (usually, $\beta=\frac{1}{4}$)

Retransmission Timer

Jacobson's Algorithm

- Only considers RTT_S to calculate RTO after each RTT measurement

  $RTO = \gamma \times RTT_S$ (usually, $\gamma=2$)

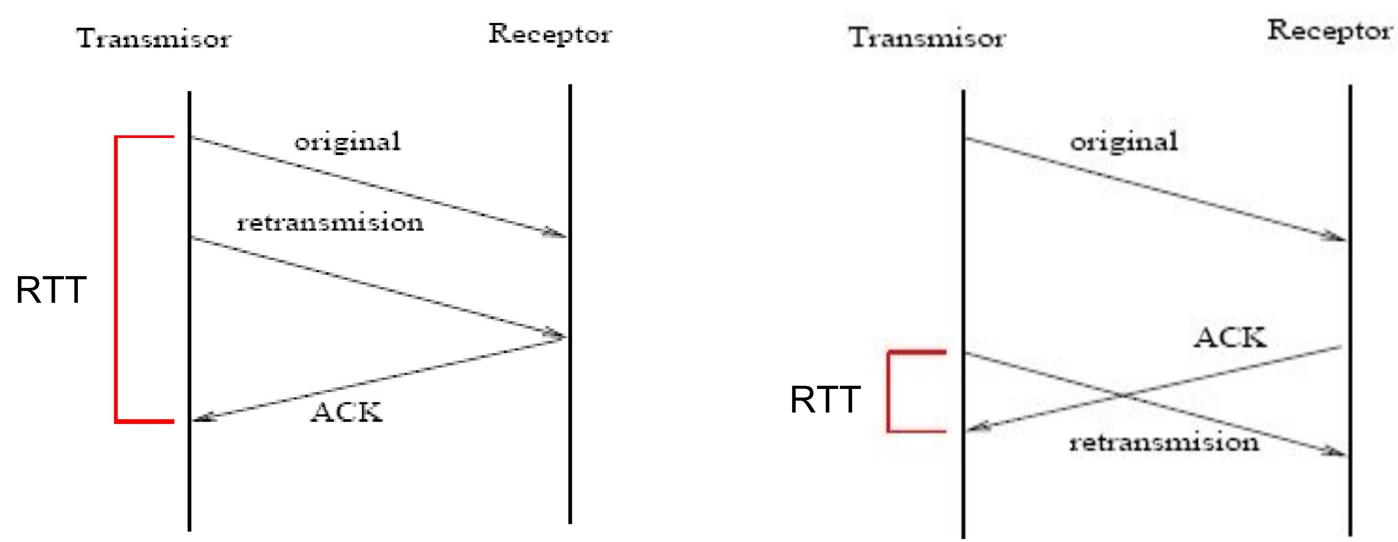
Jacobson/Karels's Algorithm

- Combines RTT_S and RTT_D

  $RTO = RTT_S + 4 \times RTT_D$

Karn's Algorithm

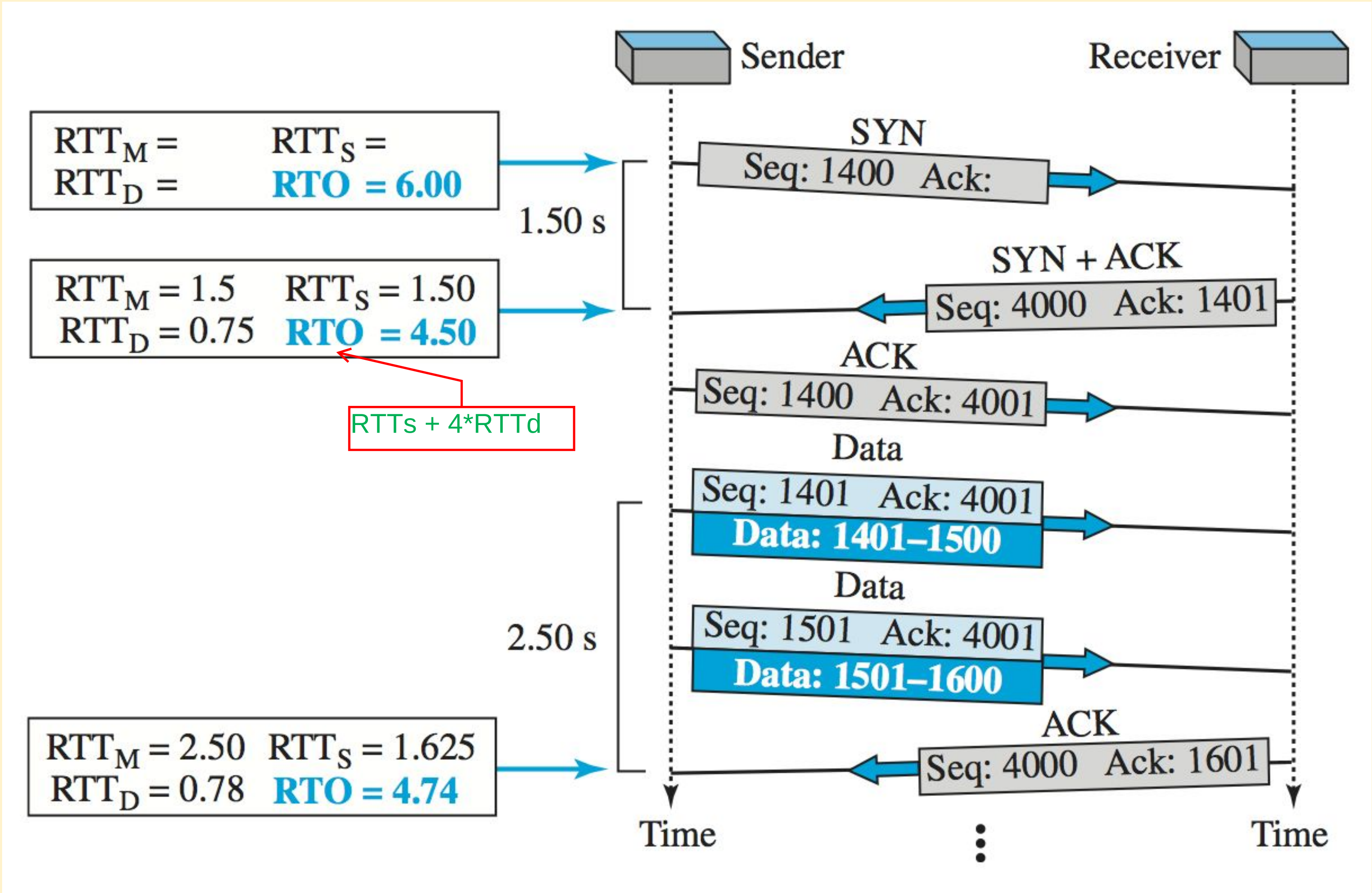
- To avoid ambiguity, retransmitted segments are not considered in RTT_S calculation
- The value of RTO is doubled for each retransmission (**exponential backoff**)



Retransmission Timer

Example: Calculate the value of timers.

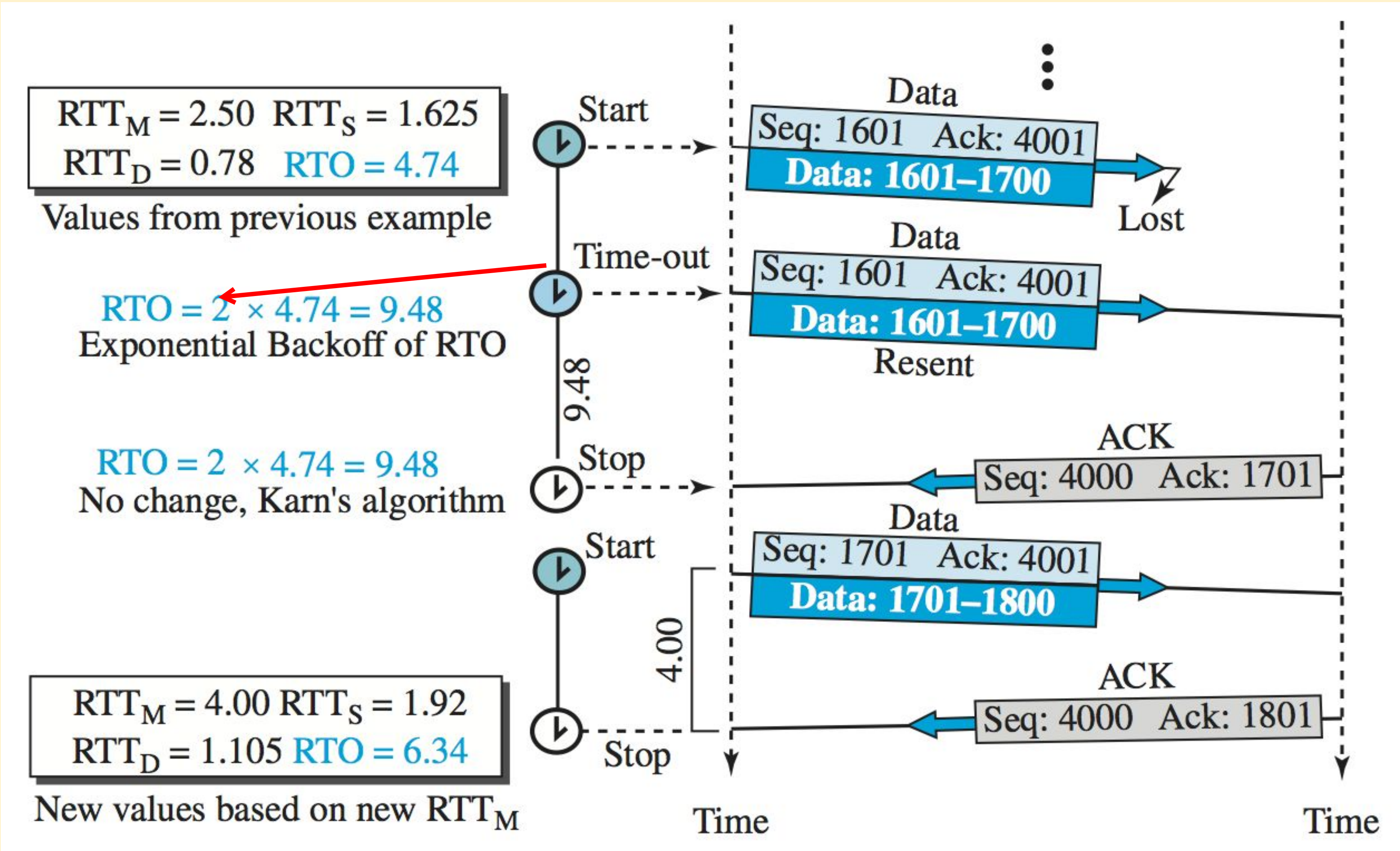
- α=1/8, β=1/4



Retransmission Timer

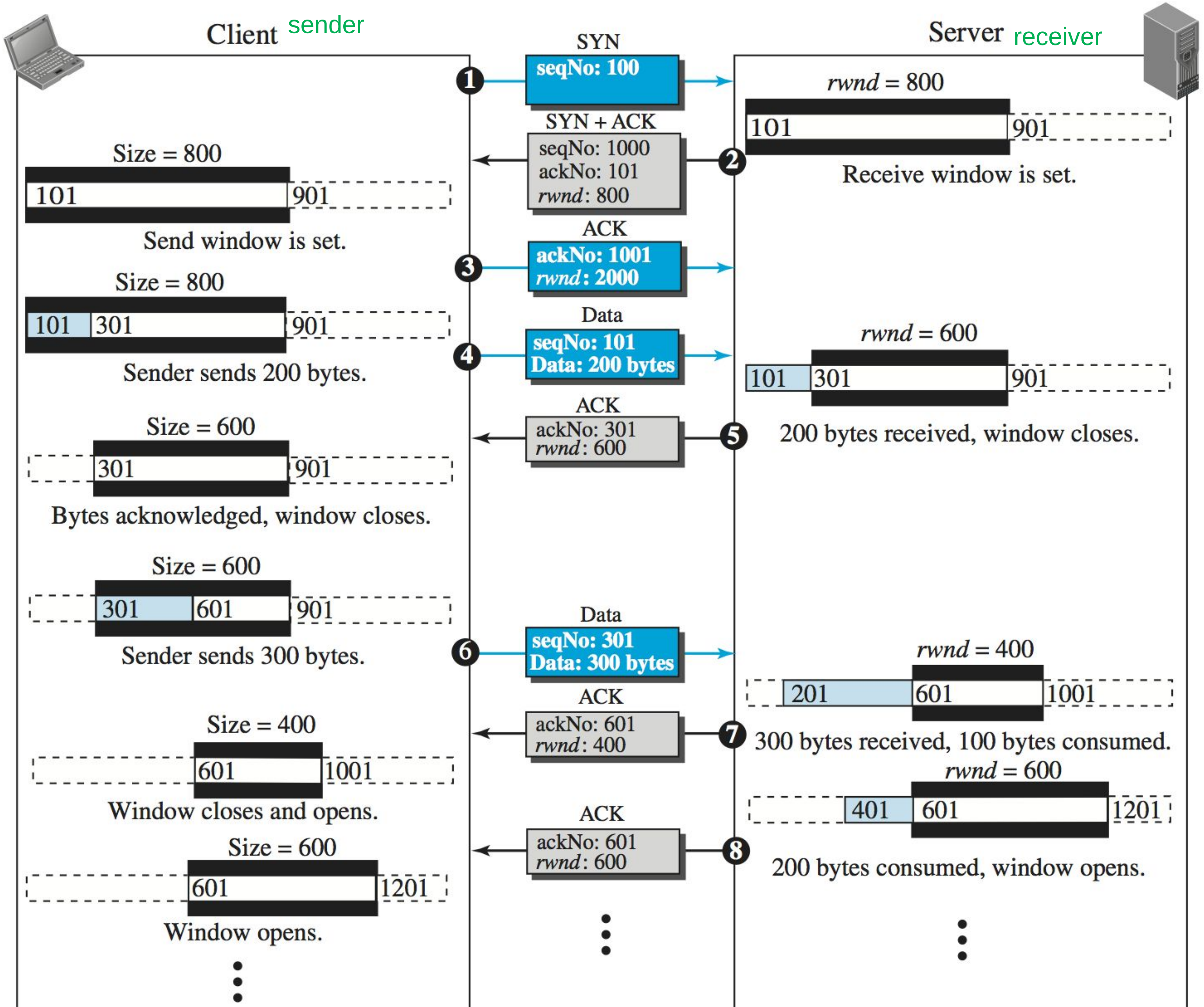
Example: Calculate the value of timers (cont.).

- α=1/8, β=1/4



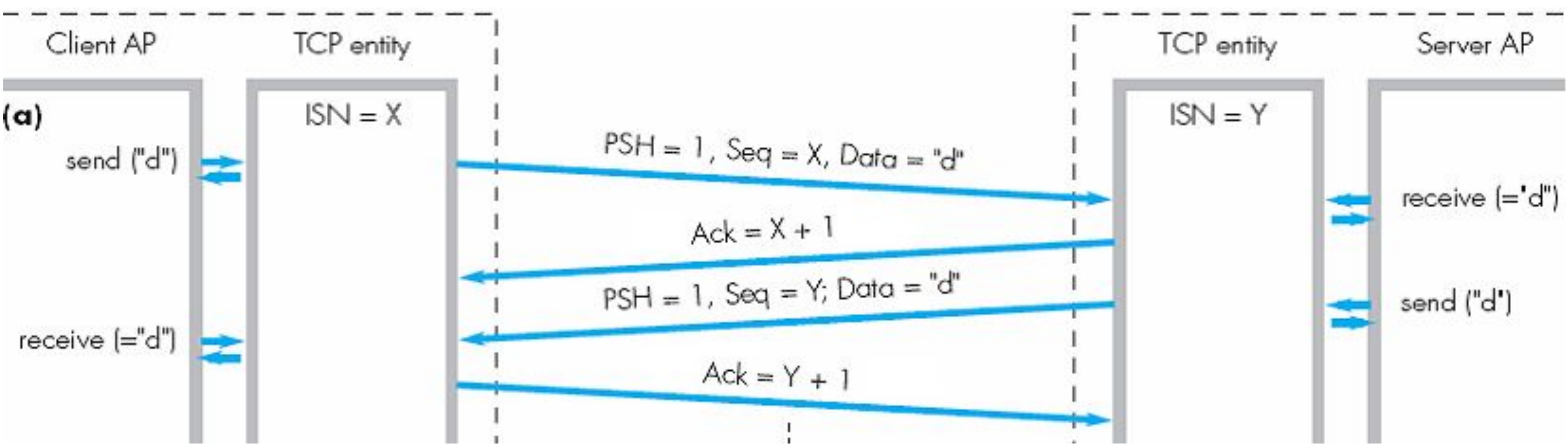
Flow Control

- Control of sending data rate to prevent overwhelming the receiver with data
- Performed by means of the receive window, announced on each ACK



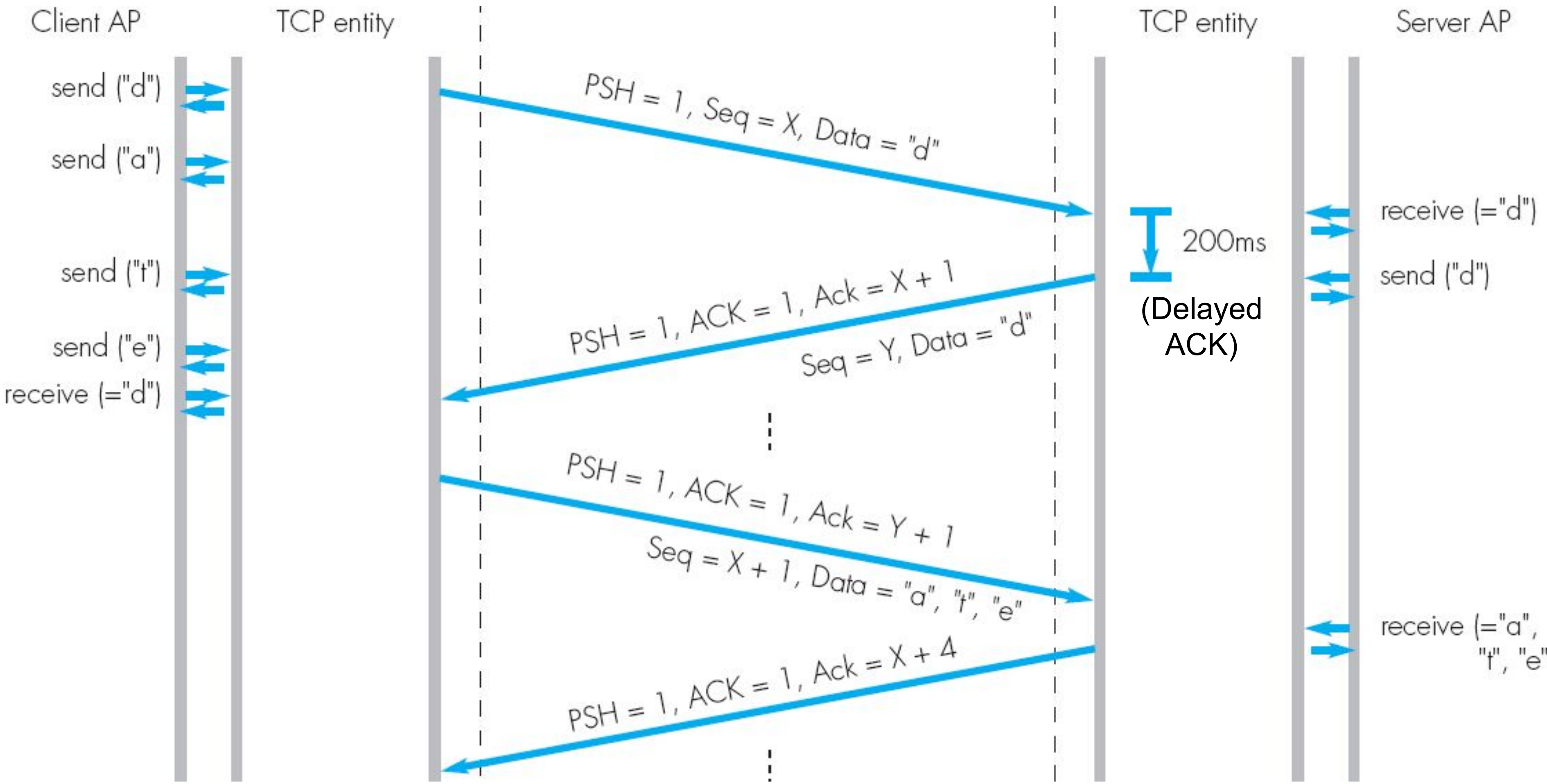
Flow Control: Silly Window Syndrome

- The silly window syndrome occurs when:
  - The sending application sends data slowly (e.g. byte by byte)
  - The receiving application consumes data slowly
- Any of these situations results in the sending of data in very small segments, which reduces the efficiency of the operation
- **Silly window in the sender** (e.g. interactive applications)
  - Each character needs 4 messages (40 bytes in headers)
  - A character (1 byte) uses more than 160 bytes



Flow Control: Silly Window Syndrome

- **Nagle Algorithm (RFC 1122, Sec. 4.2.3.4)**
  - *"If there is unacknowledged data, then the sending TCP buffers all user data (regardless of the PSH bit), until the outstanding data has been acknowledged or until the TCP can send a full-sized segment (MSS bytes)"*

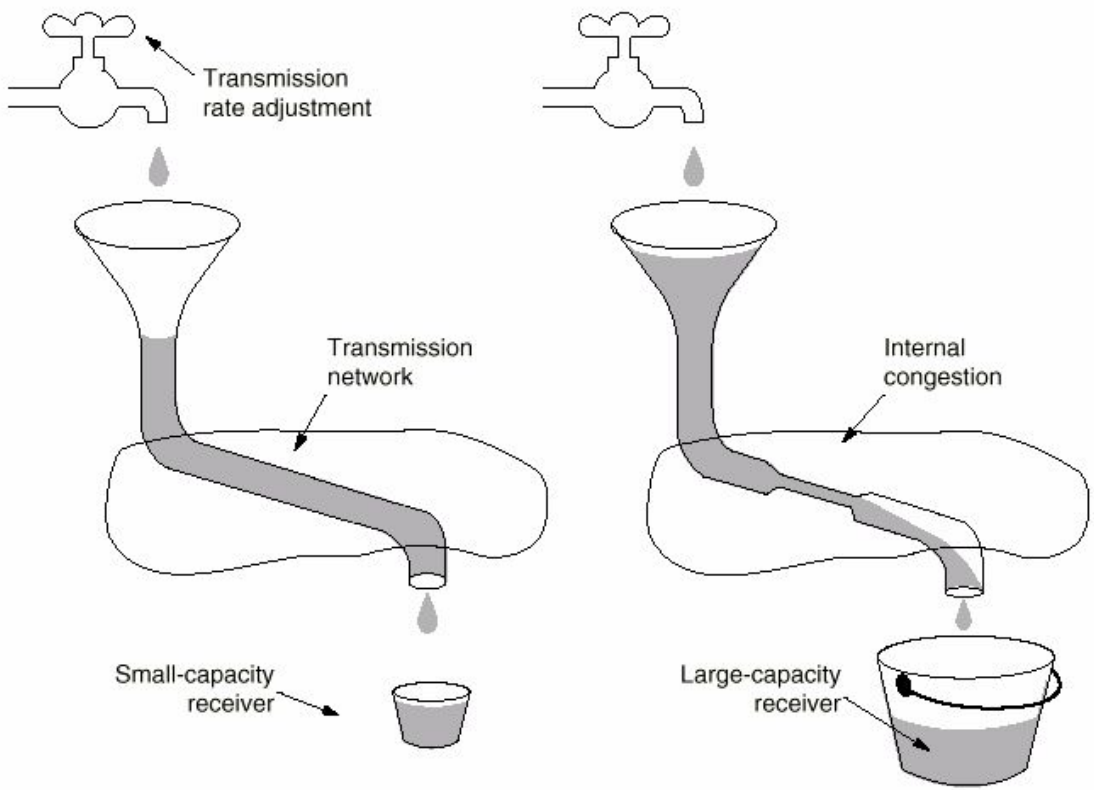


Flow Control: Silly Window Syndrome

- **Silly window in the receiver**
  - Application program that consumes data slowly
  - Windows of reduced size are advertised, generating the previous effect
- **Clark's Algorithm**
  - Receiver announces a window size of zero until either:
    - there is enough space to accommodate a full-sized segment (MSS)
    - at least half of the receive buffer is empty
- **Delayed ACKs**
  - To prevent the sender from sliding its window
  - This reduces traffic (number of ACKs) but may result in the sender unnecessarily retransmitting the unacknowledged segments
  - Acknowledgments must not be delayed by more than 500 ms, as TCP states

Congestion Control

- Packets loss in the Internet is mostly due to a congestion problem at some point of the network:
  - A router cannot process and forward packets at the same rate at which they arrive (IP does not provide congestion control)
  - Therefore, it starts dropping packets (including acknowledgements)
- Congestion control and flow control are two different mechanisms:



Congestion Control

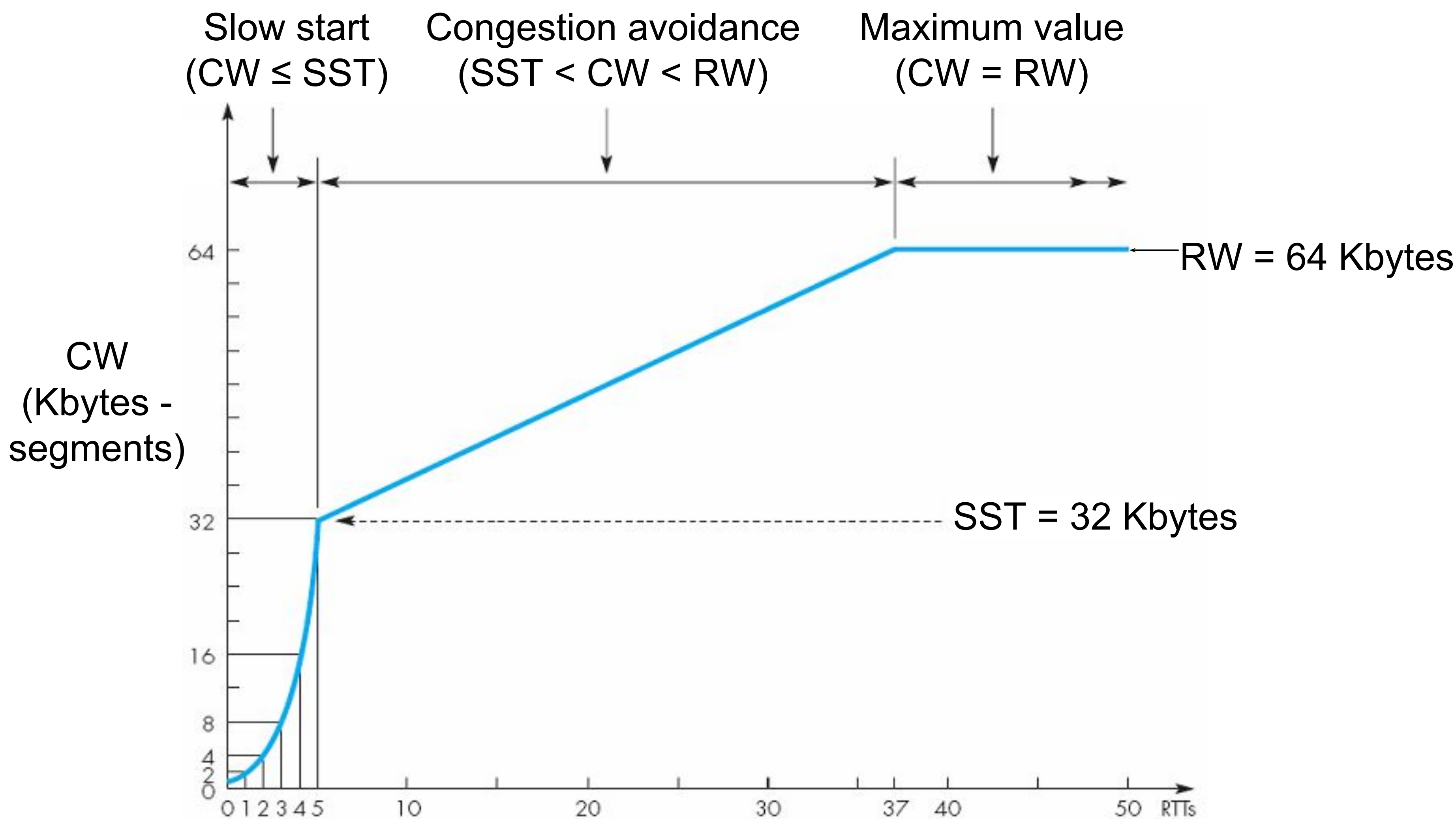
- The sender uses the rate at which acknowledgements arrive to regulate the rate at which data segments are sent
- This is implemented with the **Congestion Window** (CW size)
  - It is complementary to the receive window (RW size) used for flow control
  - With no congestion (no segment loss nor delay) the congestion window reaches the same size as the receive window (CW=RW)
  - Under congestion, CW is progressively reduced
  - When the congestion condition disappear, CW is progressively increased
  - The maximum number of bytes that can be sent (Allowed Window) is the minimum of both window sizes:

$$AW = \min \{ RW, CW \}$$

Congestion Control

- When congestion exists, segments are lost or delayed
- The size of the congestion window starts with one MSS (CW = 1)
  - A single segment of size MSS is sent
- Then, CW increases in two different phases:
  - **Slow start**
    - CW increases by one MSS each time an acknowledgment arrives
    - This causes an exponential growth (CW = 1, 2, 4, 8, 16, 32…)
    - This phase stops when CW reaches the Slow Start Threshold (SST)
    - Usually, the initial value of SST is 64 Kbytes
  - **Congestion avoidance**
    - From SST, CW is increased by one MSS each time the whole “window” of segments is acknowledged (i.e., CW segments)
      - A window is the number of segments transmitted during RTT
      - Actually, after each acknowledgement, CW increases by 1/CW
    - This causes a linear growth

Congestion Control

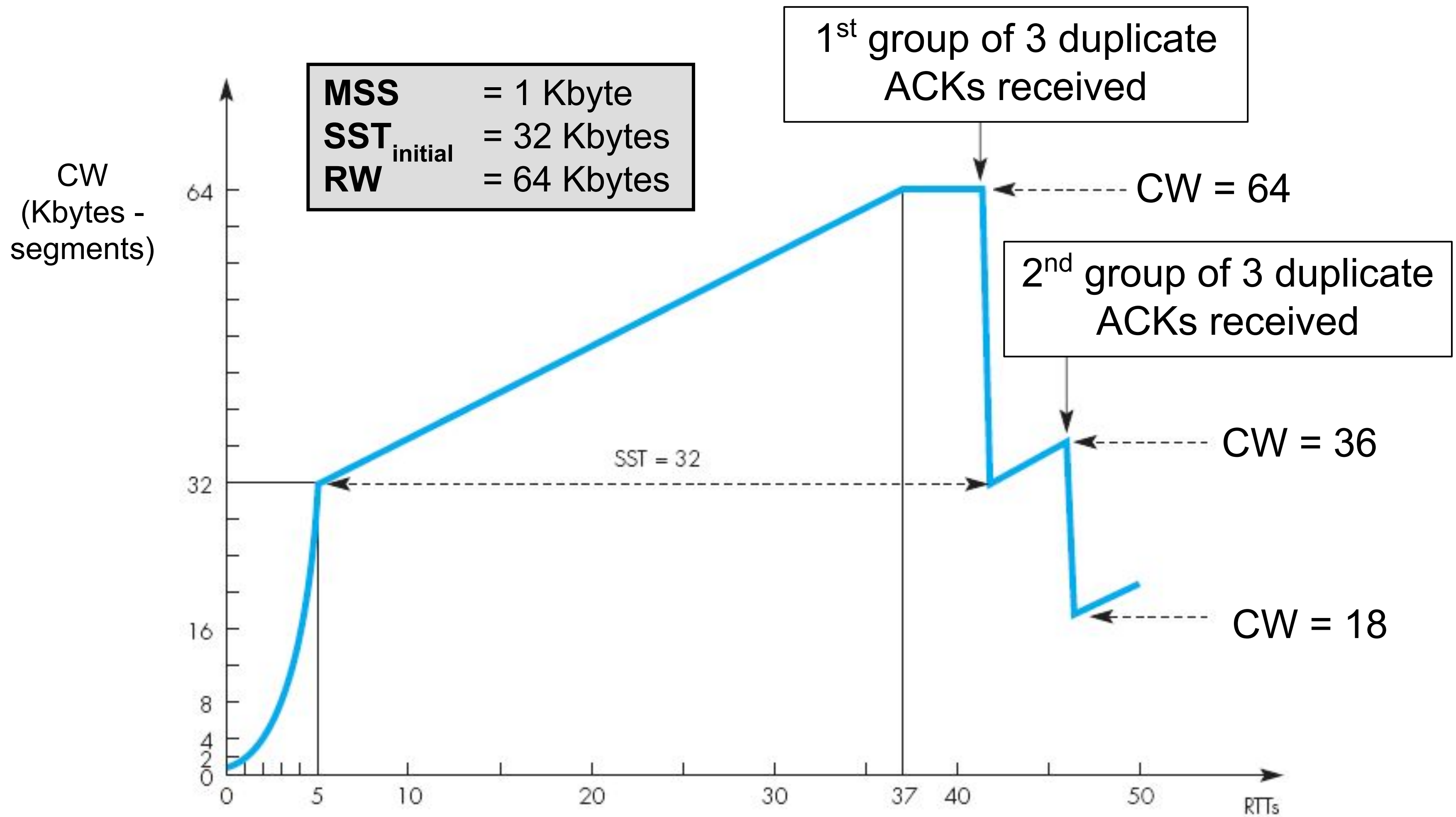


Congestion Control

- Network congestion situations are detected in an indirect way
- **Fast recovery**
  - Three duplicate ACKs are interpreted as light congestion in the network (there is still traffic on the network, since acknowledgements arrive)
  - The following actions are done:
    - Reduce CW and SST to half of the previous value of CW
    - Perform congestion avoidance
- **Retransmission time-out**
  - A retransmission time-out is interpreted as severe congestion (traffic on the network is interrupted, since no acknowledgement arrived)
  - The following actions are done:
    - Reset CW to 1
    - Reduce SST to half of the previous value of CW
    - Perform slow start

Congestion Control

- **Fast recovery**



Congestion Control

- **Retransmission time-out**