CS321: Computer Networks

Error and Flow Control in TCP

Dr. Manas Khatua
Assistant Professor
Dept. of CSE
IIT Jodhpur
E-mail: manaskhatua@iitj.ac.in

SEQ and ACK numbers in TCP

- TCP views data as an unstructured, but ordered, stream of bytes
- The SEQ number for a segment is the byte-stream number of the first byte in the segment.
- The ACK number that Host A puts in its segment is the SEQ number of the next byte Host A is expecting from Host B.

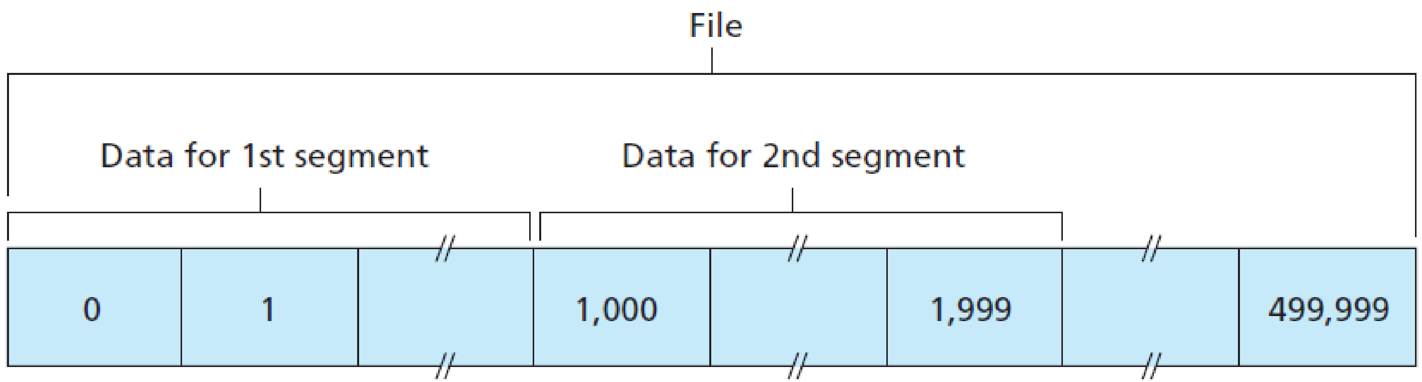


Figure 3.30 ♦ Dividing file data into TCP segments

Error Control in TCP

- Error control in TCP is done by :
  - checksum, ACK, time-out

- By default TCP uses cumulative ACK

- When does a receiver generate ACK?

  - Rule-1: when node A sends data to node B, it piggybacks ACK
  - Rule-2: the receiver has no data to send and it receives an in-order segment, it delays sending ACK

  - Rule-3: there should not be more than two in-order unACKed segments at any time (it is delayed ACK)
  - Rule-4: when a segment arrives with an out-of-order sequence number; or, it is *fast retransmission* of missing segments

  - Rule-5: when a missing segment arrives
  - Rule-6: If a duplicate segment arrives

Cont…

- When retransmission happens?
  - After time-out: sending TCP maintains one retransmission time-out (RTO) for each connection
  - Three duplicate ACK rule: if three duplicate ACK (i.e., an original ACK + three exactly identical copies) arrive for a segment, the next segment is retransmitted without waiting for the time-out.
  - Why three duplicate ACK?
    - Since TCP does not know whether a duplicate ACK is caused by a lost segment or just a reordering of segments, it waits for at max two duplicate ACK. If three or more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost.
- Data may arrive out of order and be temporarily stored by the receiving TCP, but TCP guarantees that no out-of-order data are delivered to the process.

RTT Estimation and Timeout

- TCP uses a timeout/retransmit mechanism to recover from lost segments.
- The timeout should be larger than the connection's round-trip time (RTT)

- Few Questions:
  - How much larger? How should the RTT be estimated in the first place?
  - Should a timer be associated with each and every unACKed segment?

- The sample RTT, denoted *SampleRTT*, for a segment is
  - the amount of time between when the segment is sent (that is, passed to IP) and when an ACK for the segment is received.

- But, the *SampleRTT* values will fluctuate from segment to segment due to congestion in the routers and to the varying load on the end systems.
- Solution:
  - TCP maintains an average, called *EstimatedRTT*, of the *SampleRTT* values.
  - Exponentially weighted moving average (EWMA)

$$\text{EstimatedRTT} = (1 - \alpha) \times \text{EstimatedRTT} + \alpha \times \text{SampleRTT}$$

The recommended value of α is $= 1/8$

Cont…

- It is also valuable to have a measure of the variability of the RTT.

$$DevRTT = (1 - \beta) \times DevRTT + \beta \times | SampleRTT - EstimatedRTT |$$

The recommended value of β is 1/4.

- For little fluctuation, the DevRTT will be small; on the other hand, for lot of fluctuation, DevRTT will be large.

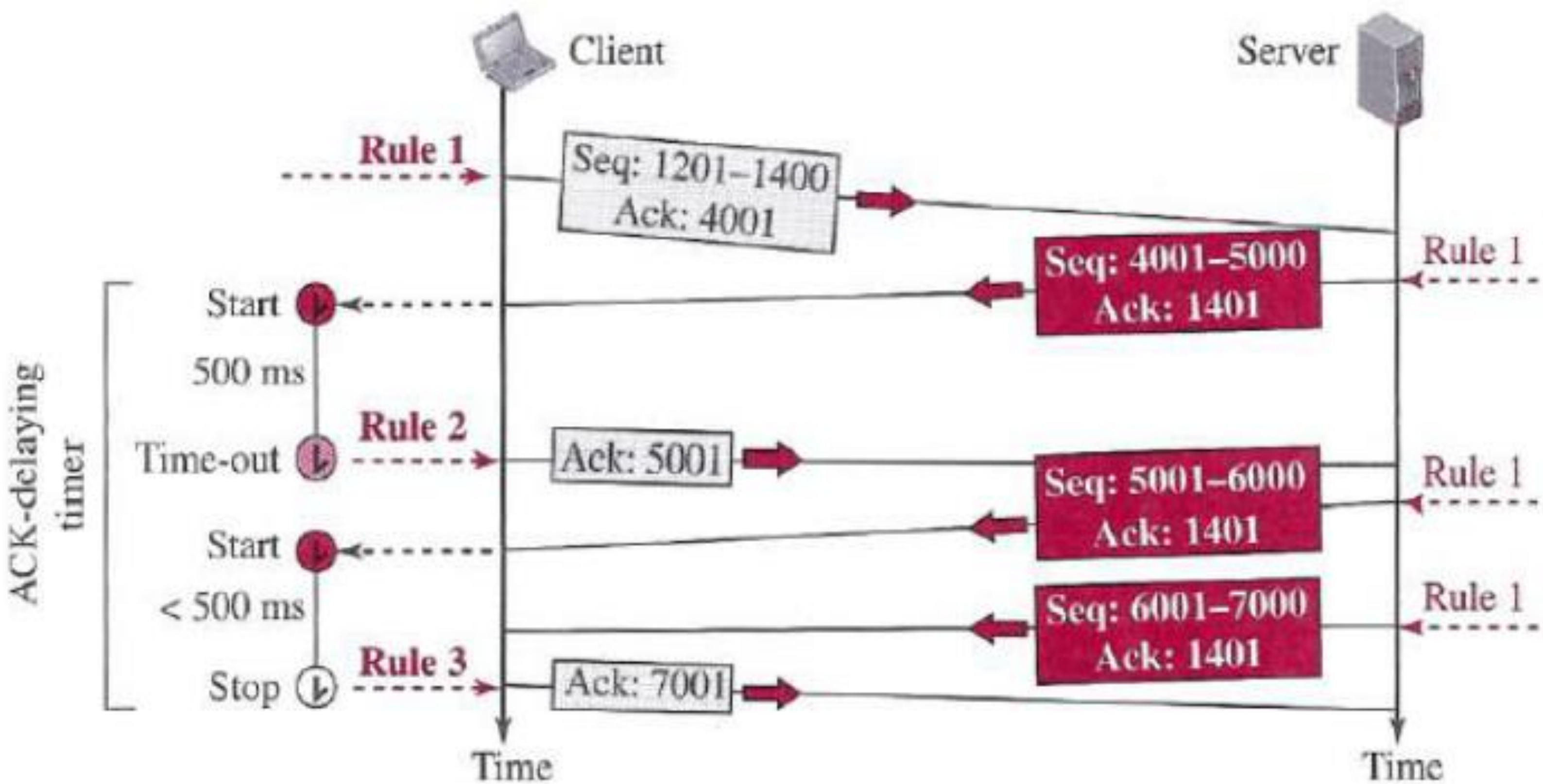
- It is desirable to set the timeout equal to the *EstimatedRTT* plus some margin.
- The margin should be large when there is a lot of fluctuation in the *SampleRTT* values; it should be small when there is little fluctuation.

$$TimeoutInterval = EstimatedRTT + 4 \times DevRTT$$

- *Another Question:* How long the receiver waits before sending a stand-alone ACK to acknowledge data on a TCP socket?
  – *Delayed ACK* was invented to reduce the number of *ACKs* required to acknowledge the segments, so protocol overhead is reduced.
  – A host may delay sending an ACK response by up to 500 ms.
  – However, a stand-alone ACK is sent if two full packets worth of data arrive before the delayed ACK timer expires.

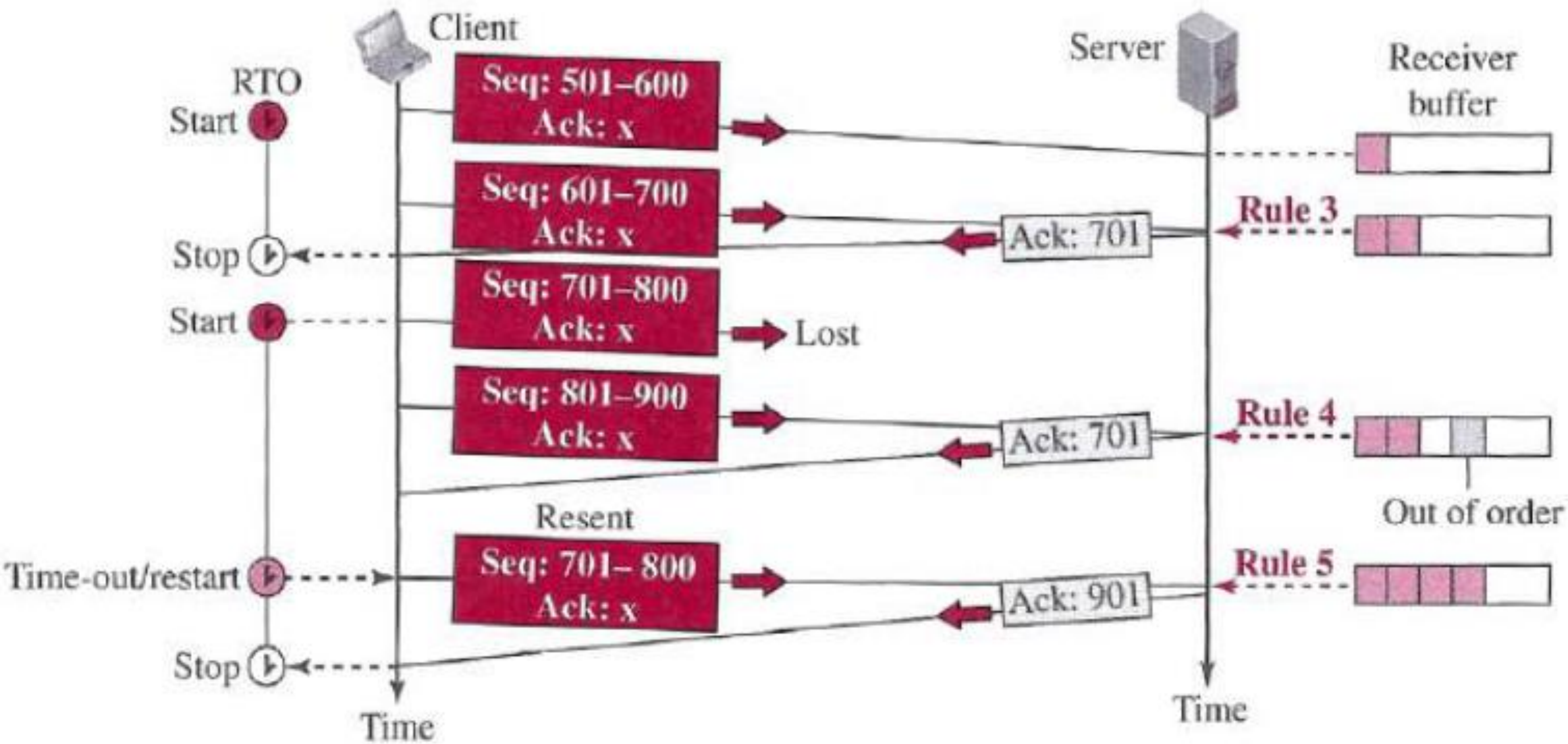
Example : Normal Scenario

- The client TCP sends one segment (2000 byte);
- server TCP sends three segments (3 x 1000 byte).

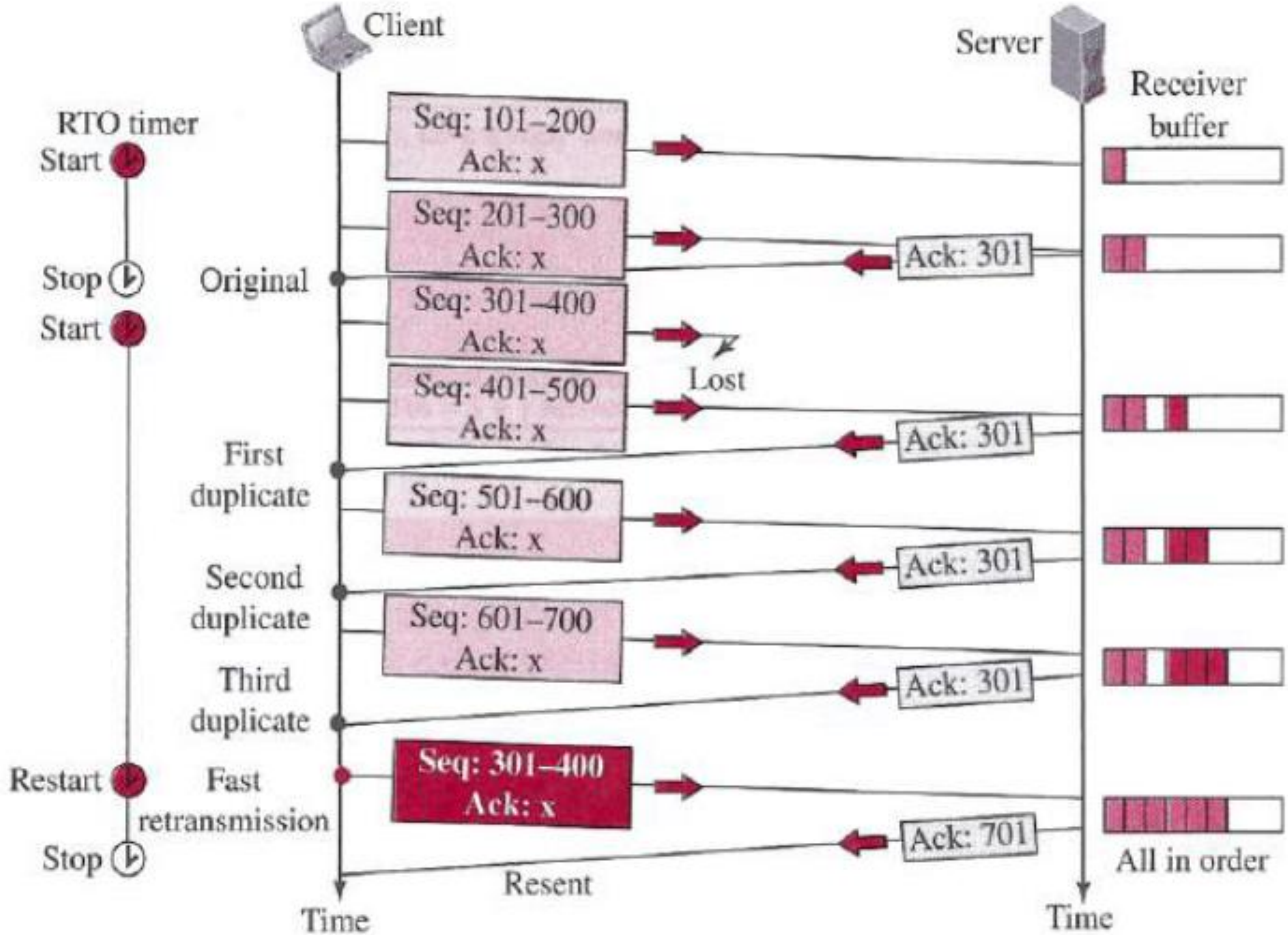


Example : Lost Segment Scenario

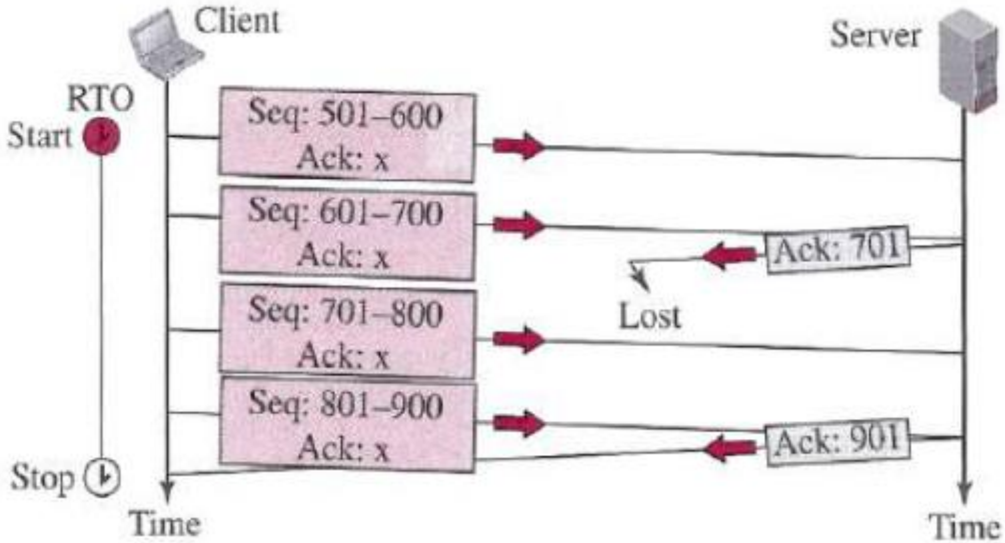
- assuming that data transfer is unidirectional
  - from client to server

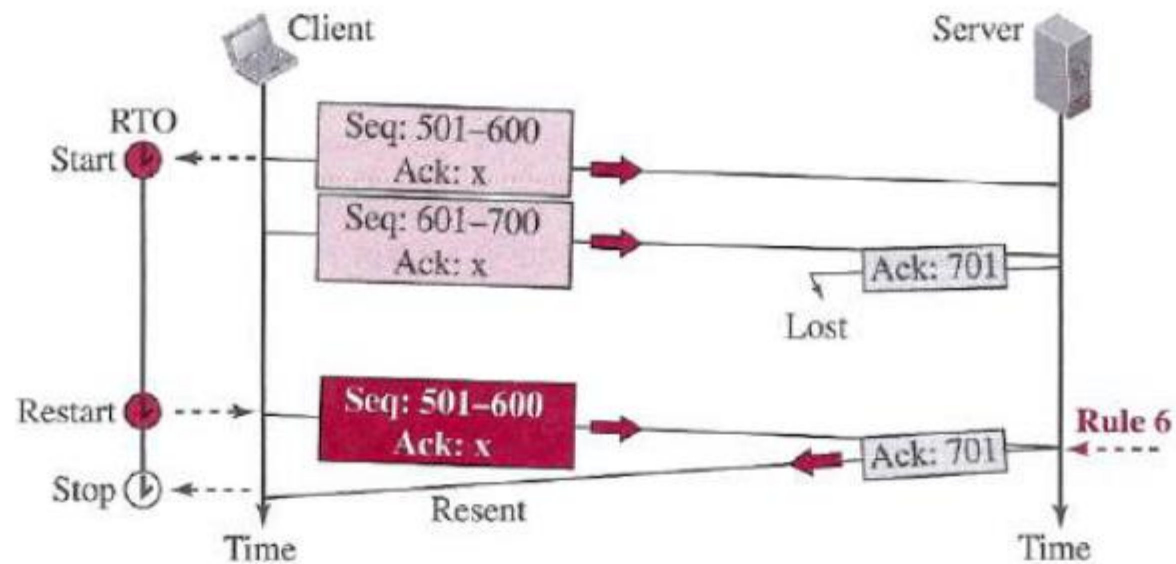


Example : Fast Retransmission

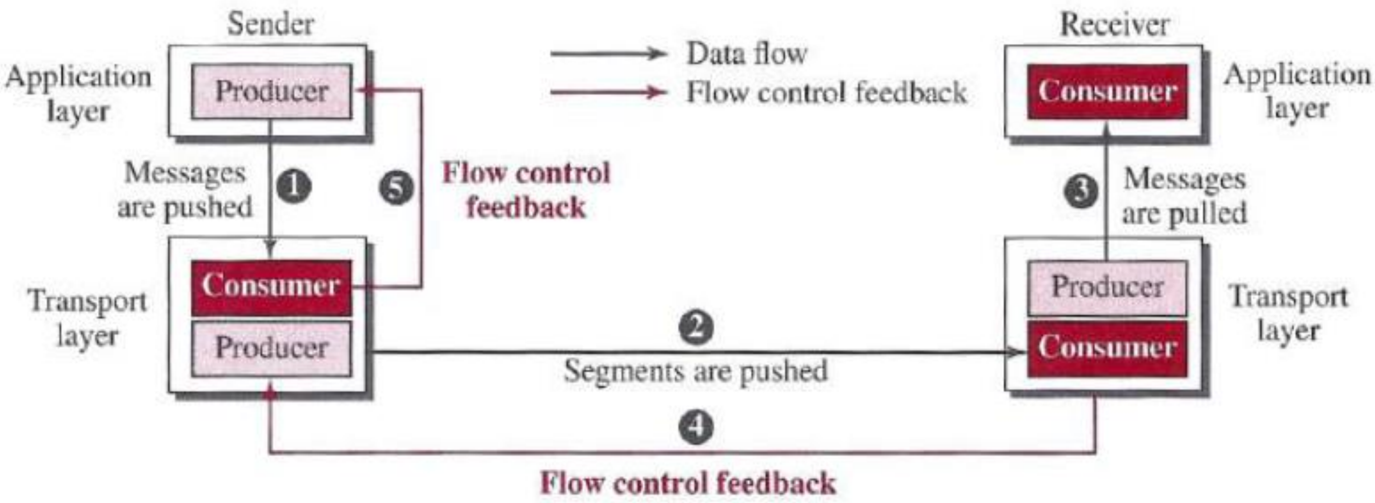


Example : Lost ACK





Flow Control in TCP

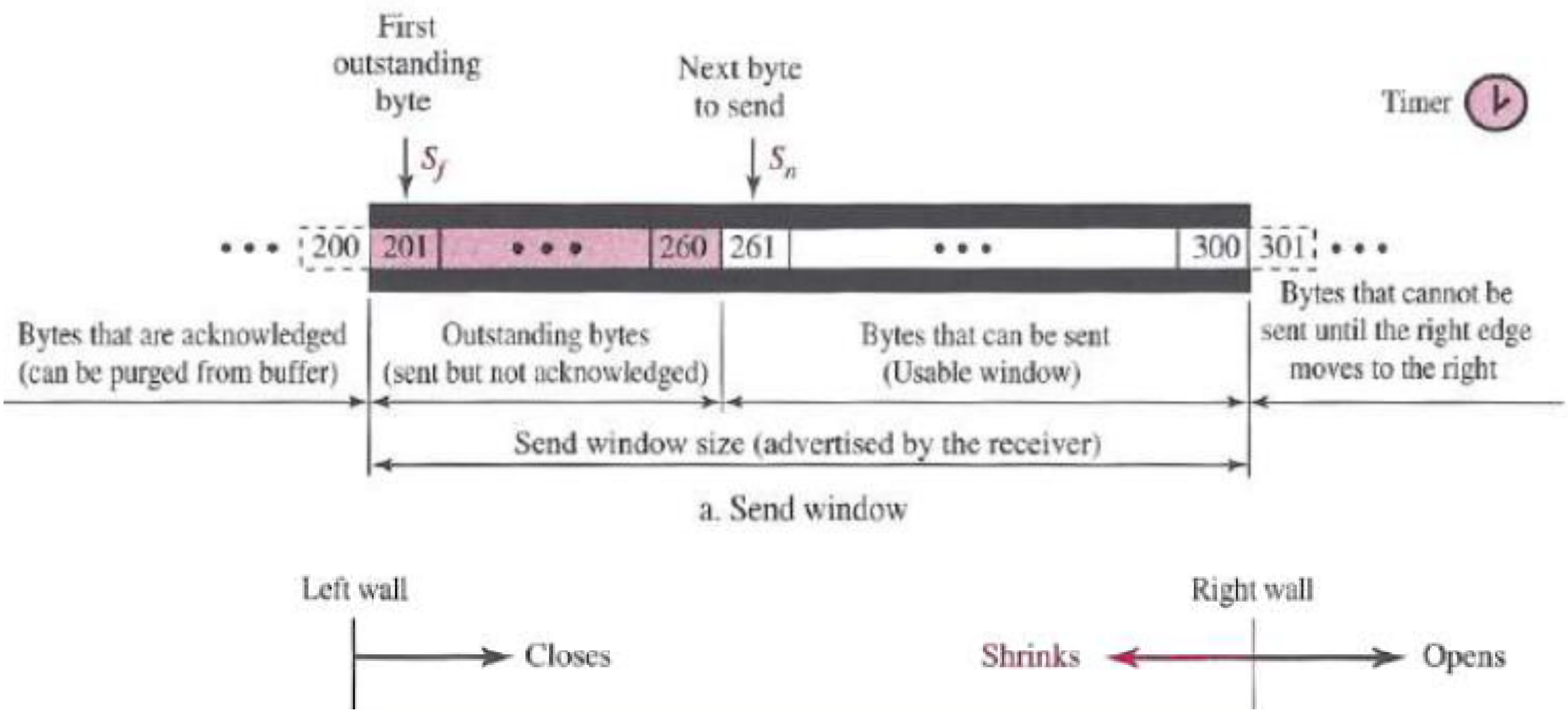


- The receiving TCP controls the sending TCP; the sending TCP controls the sending process.
- No flow control between receiving TCP and receiving process.

- To achieve flow control:
  - TCP forces the sender and the receiver to adjust their window sizes, although the size of the buffer for both parties is fixed when the connection is established.
  - The opening, closing, and shrinking of the send window is controlled by the receiver.

Send Window in TCP

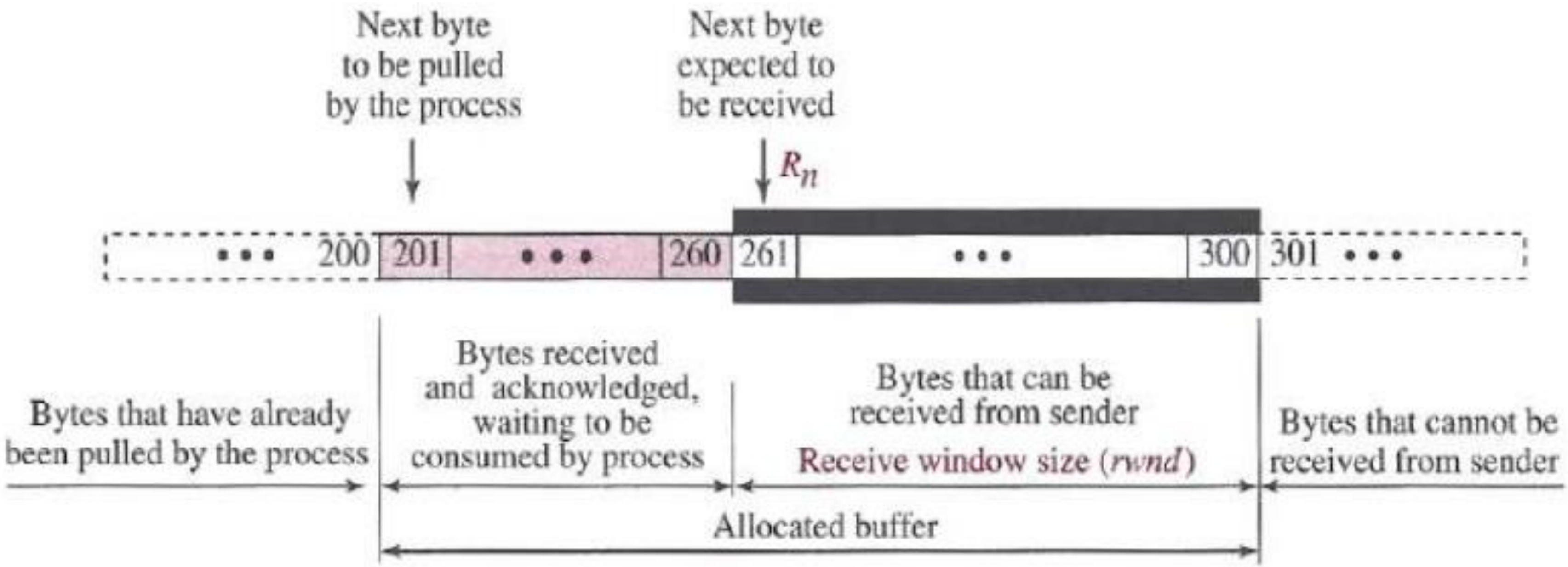
- TCP uses Send window & Receive window
- Let send window size = 100



Modified SR (for Send Window)

- Sending window in TCP follows Selective-Repeat (SR) protocol with few modifications
  - The window size in SR is the number of packets, but the window size in TCP is the number of bytes.
  - TCP can store data received from the process and send them later
  - SR requires individual ACK of each packet that was sent; but TCP sends ACK for the next packet that it is expecting (like Go-back-N)
  - SR protocol may use several timers for general transmission and selected retransmission, but TCP protocol uses only one timer.
  - Window size can be changed dynamically in TCP

Receive Window in TCP



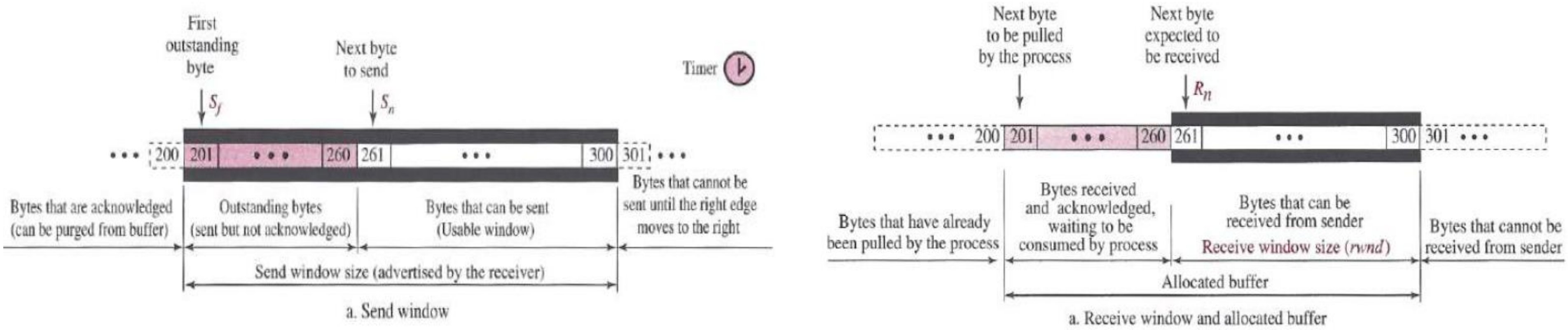
a. Receive window and allocated buffer

Modified SR (for Receive Window)

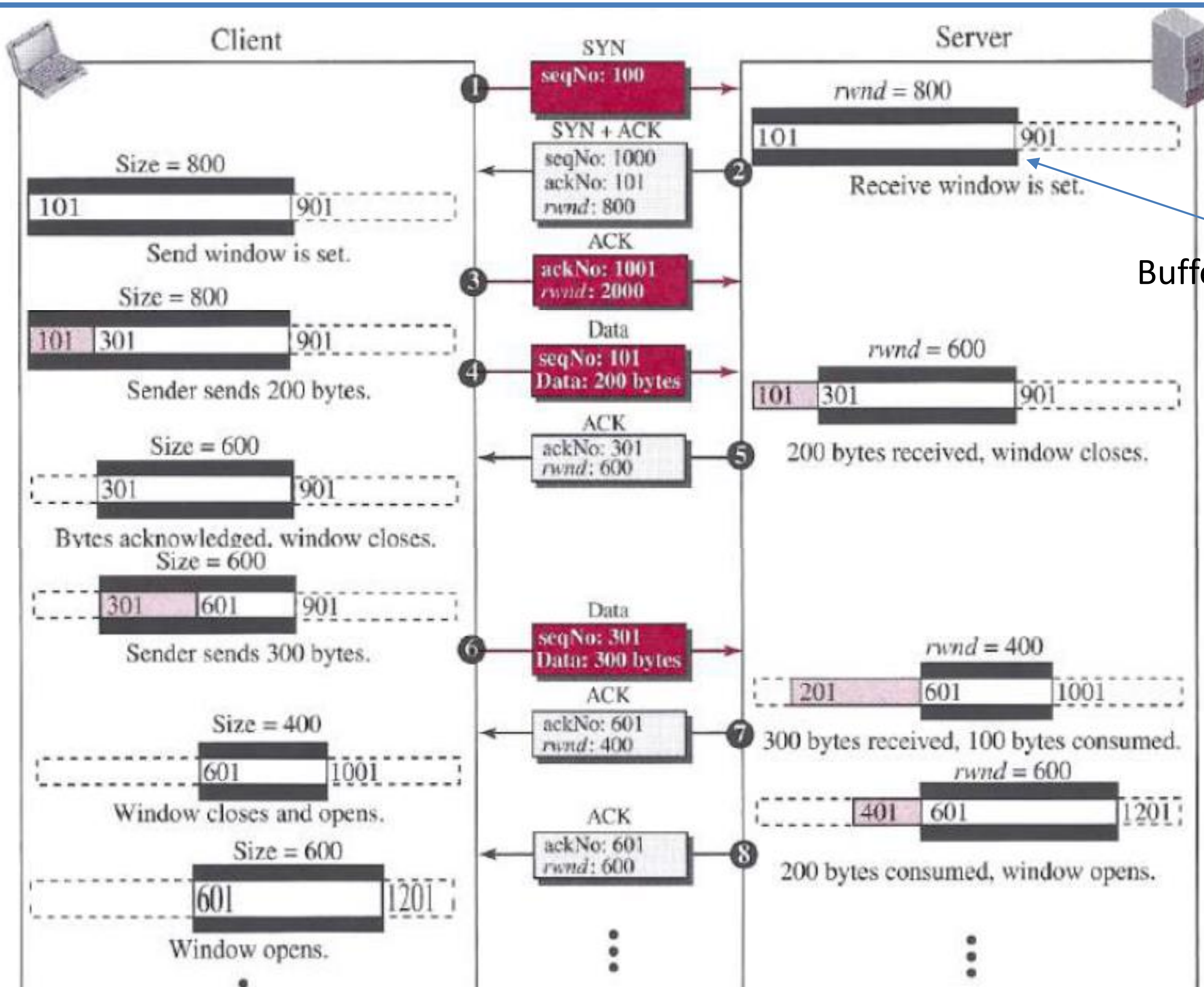
- Receive window in TCP is little different than that in SR
  - TCP allows the receiving process to pull data at its own pace.
  - The receive window size (*rwnd*) determines the number of bytes that the receive window can accept from the sender before being overwhelmed (**flow control**).

    rwnd = buffer size - number of bytes waiting to be pulled

  - ACK in SR is selective, but ACK in TCP is cumulative.
  - Similarly, retransmission is selective in SR, but oldest unACKed segment is retransmitted in TCP



Example (oneway from client to server)

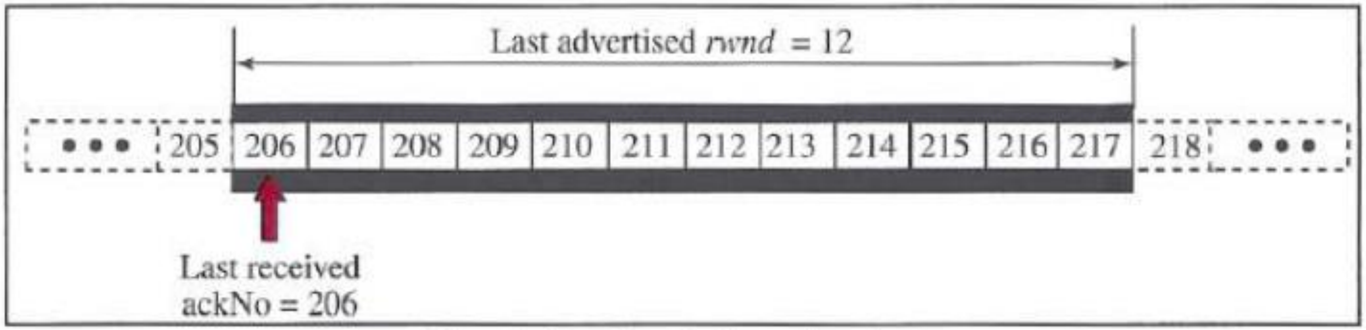


Buffer Size = 800

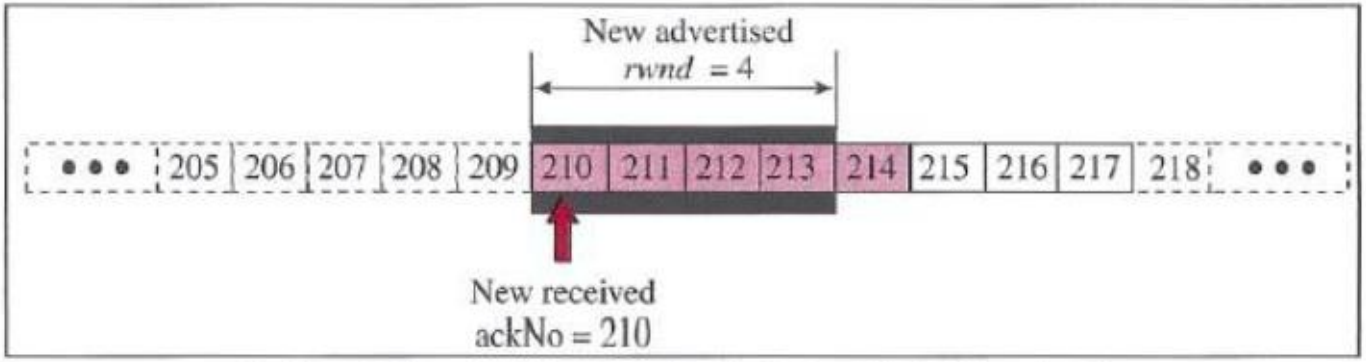
Shrinking of Windows

new ackNo + new *rwnd* >= last ackNo + last *rwnd*

If *rwnd*==0, it instructs for "window shutdown"



a. The window after the last advertisement



b. The window after the new advertisement; window has shrunk

Silly Window Syndrome

- Performance issue occurs when
  - Sending application program creates data slowly
  - OR, Receiving application program receives data slowly
  - OR, For both the above

- Example:
  - Sending process generating each byte very slowly
  - Sending TCP sends many 41 bytes segment (20 byte TCP header + 20 byte IP header + 1 byte data)

- Two types to address
  - Syndrome created by sender
  - Syndrome created by receiver

Solution

- Naïve solution faces a trade-off optimization
  - If TCP waits too long, it may delay the process
  - If TCP does not wait for long, it may end up sending small segment

- Better Solution for **sender**: Nagle's Algorithm
  - Sending TCP sends the 1st segment as it is
  - 2nd segment onwards, the sending TCP accumulates data in sending buffer and waits until
    - Either the receiving TCP sends an ACK
  - Or enough data have accumulated to fill the maximum-size segment

- Better Solution for **receiver**: Clark's two algorithms
- First algorithm:
  - send an ACK as soon as the data arrive,
  - but to announce a window size of zero until
    - either there is enough space to accommodate a segment of maximum size
    - or until at least half of the receive buffer is empty.
- Second algorithm:
  - delay sending the ACK.
  - The receiver waits until there is a decent amount of space in its incoming buffer before ACKing the arrived segments.

Thanks!