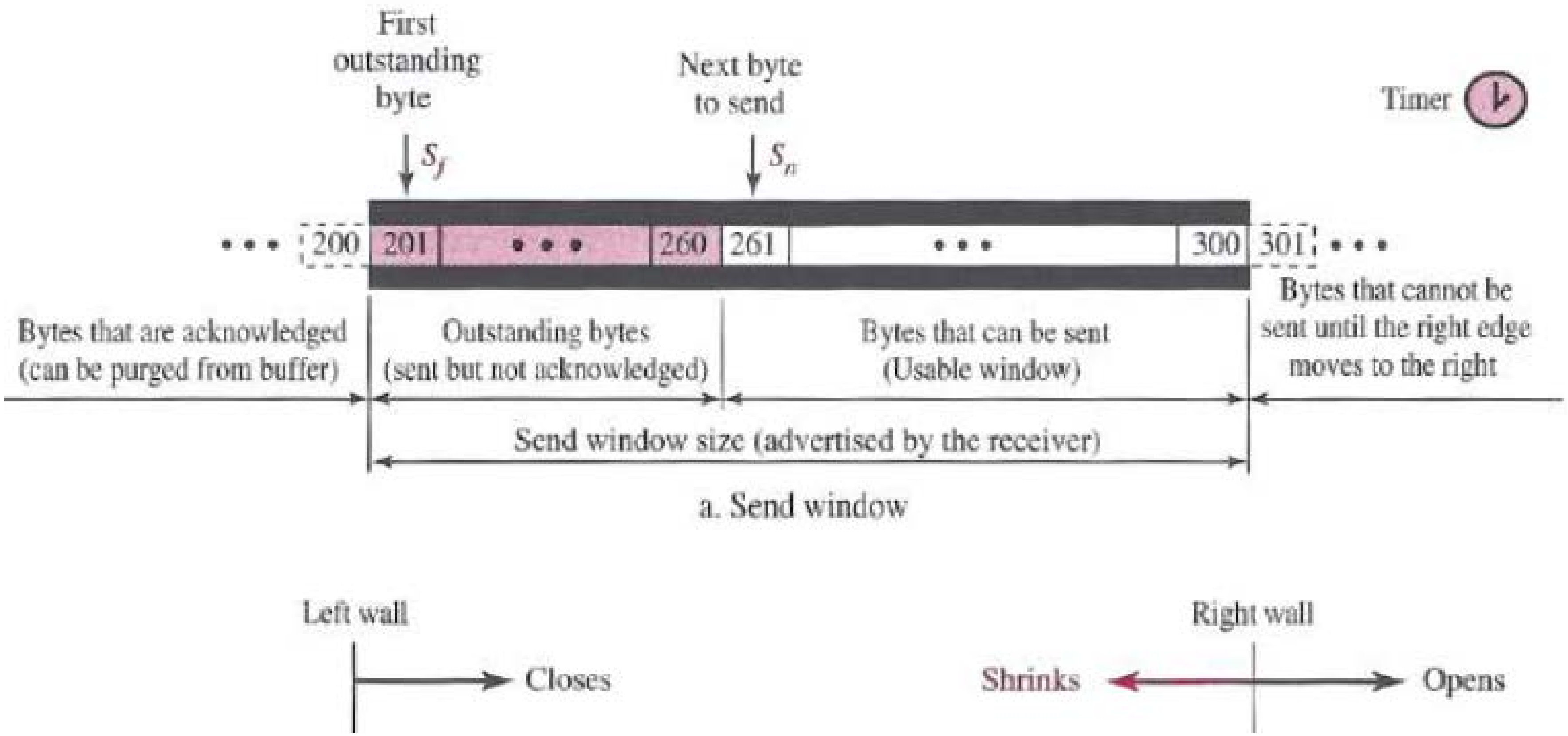
CS321: Computer Networks

Flow, Error, and Congestion Control in TCP

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

Send Window in TCP

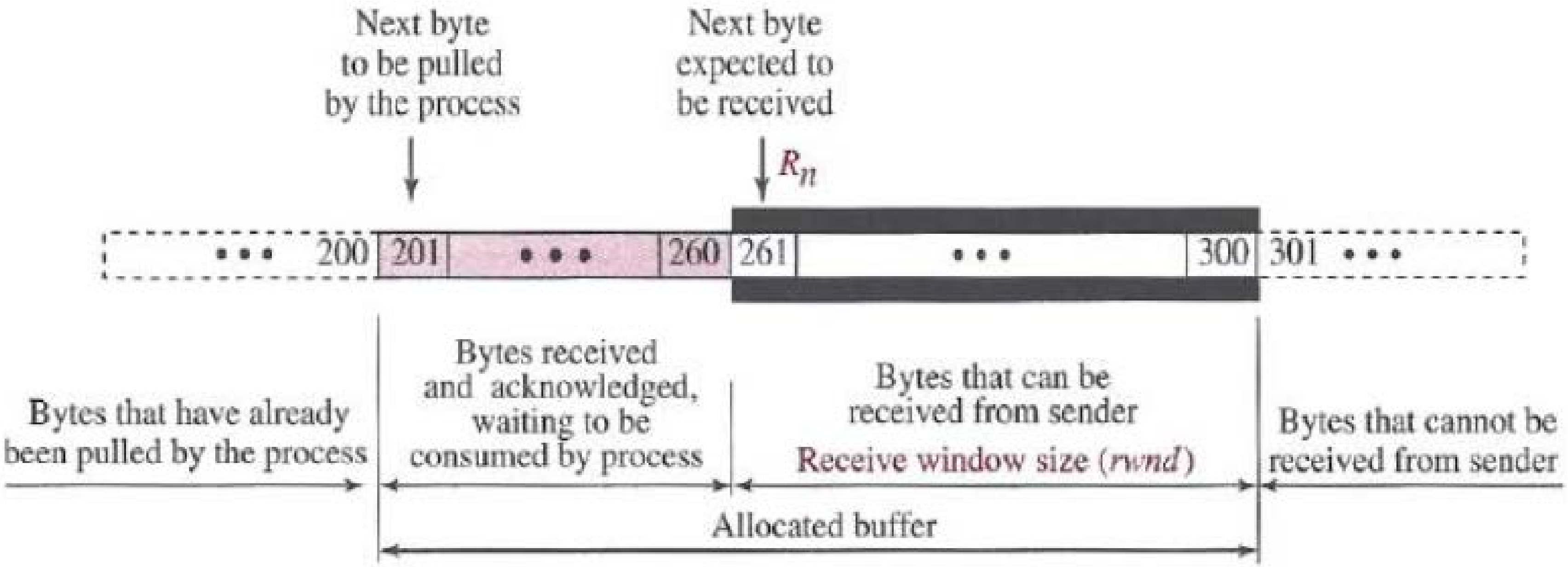
- TCP uses Send window & Receive window
- Let send window size = 100



Sending Window : TCP vs SR

- 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 protocol may use several timers, but TCP protocol uses only one timer.
  - Window size can be changed dynamically in TCP

Receive Window in TCP

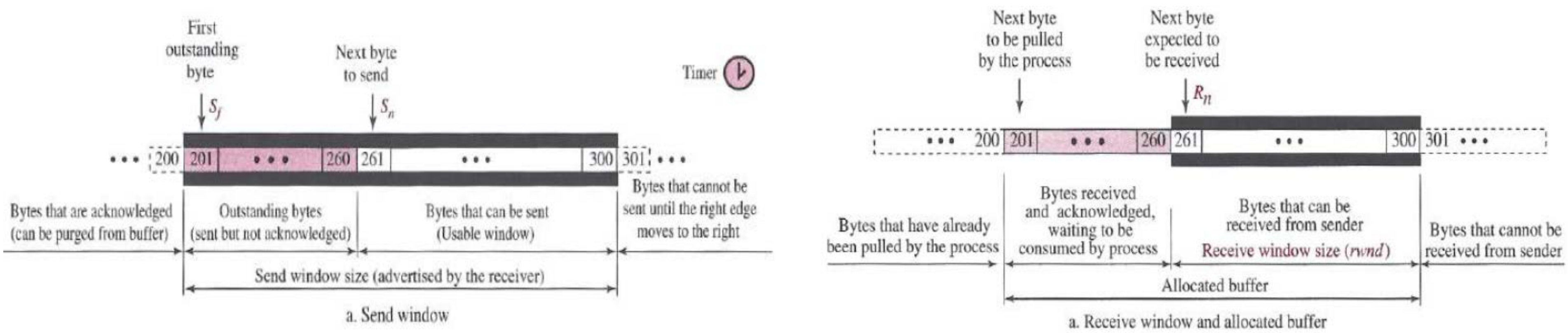


a. Receive window and allocated buffer

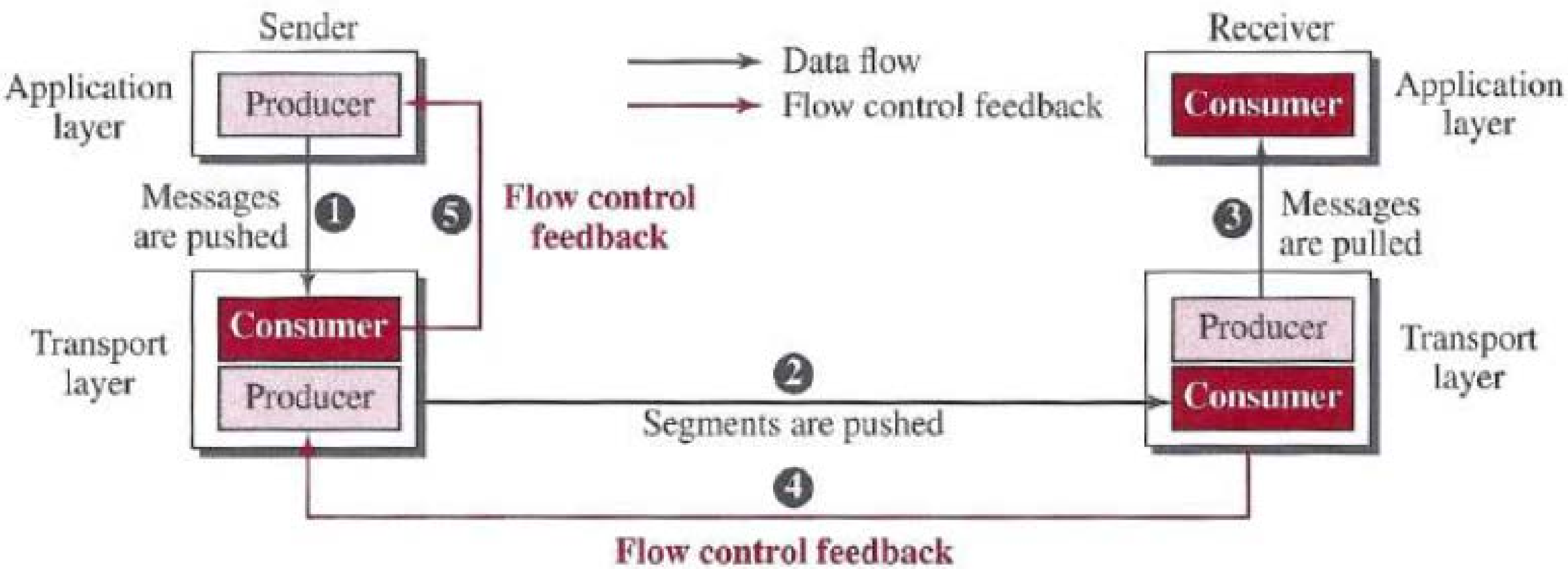
Receive Window : TCP vs SR

- 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 waiting bytes to be pulled
  - ACK in SR is selective, but in TCP is cumulative

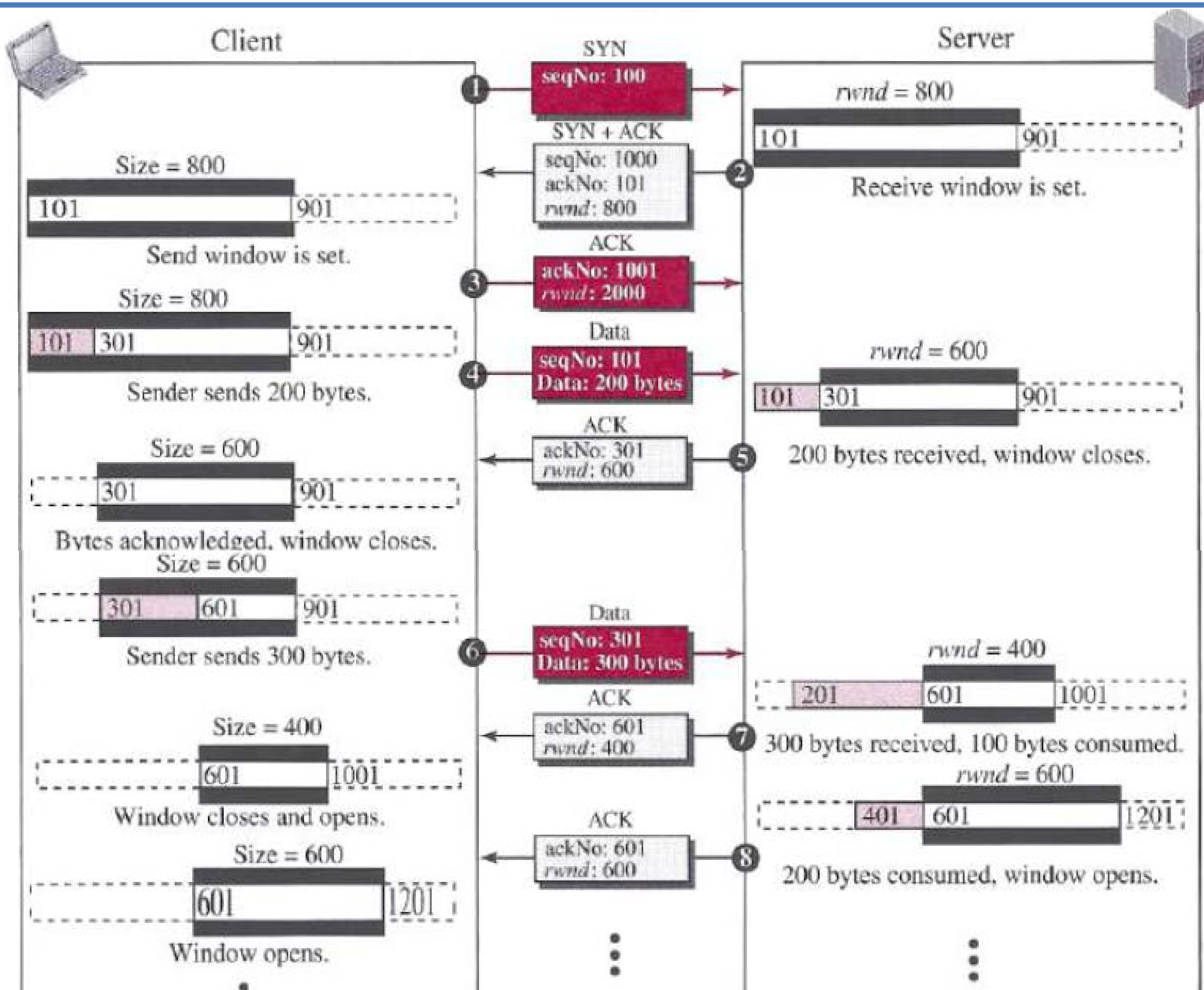


Flow Control in TCP



- 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.

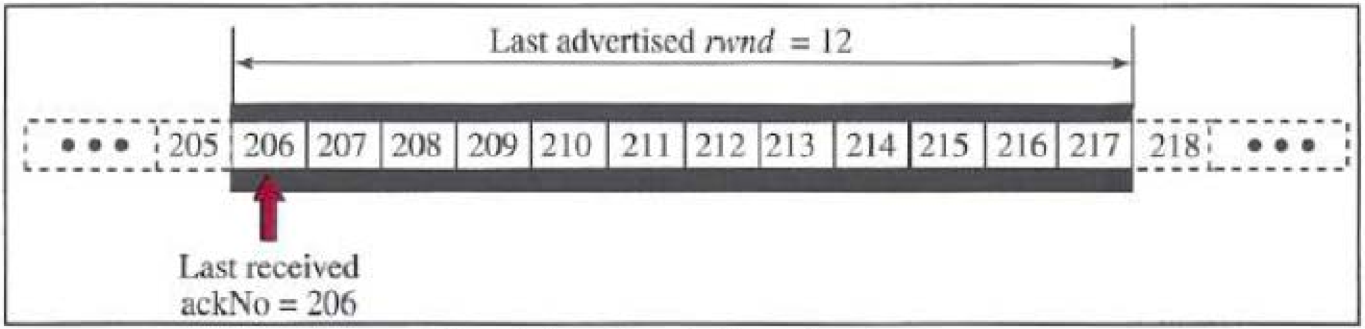
Example (from client to server)



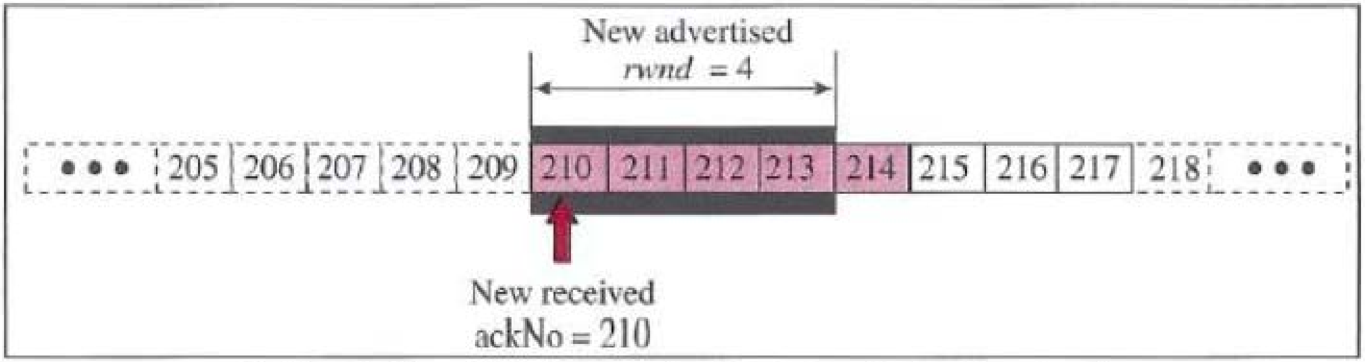
Shrinking of Windows

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
  - Receiving application program receives data slowly
  - 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
  - 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

Cont…

- Better Solution for receiver: Clark's two algorithms
- First,
  - 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,
  - delay sending the ACK.
  - The receiver waits until there is a decent amount of space in its incoming buffer before acknowledging the arrived segments.

Error Control in TCP

- Error control in TCP is done by :
  - checksum, ACK, time-out
- By default TCP uses cumulative ACK

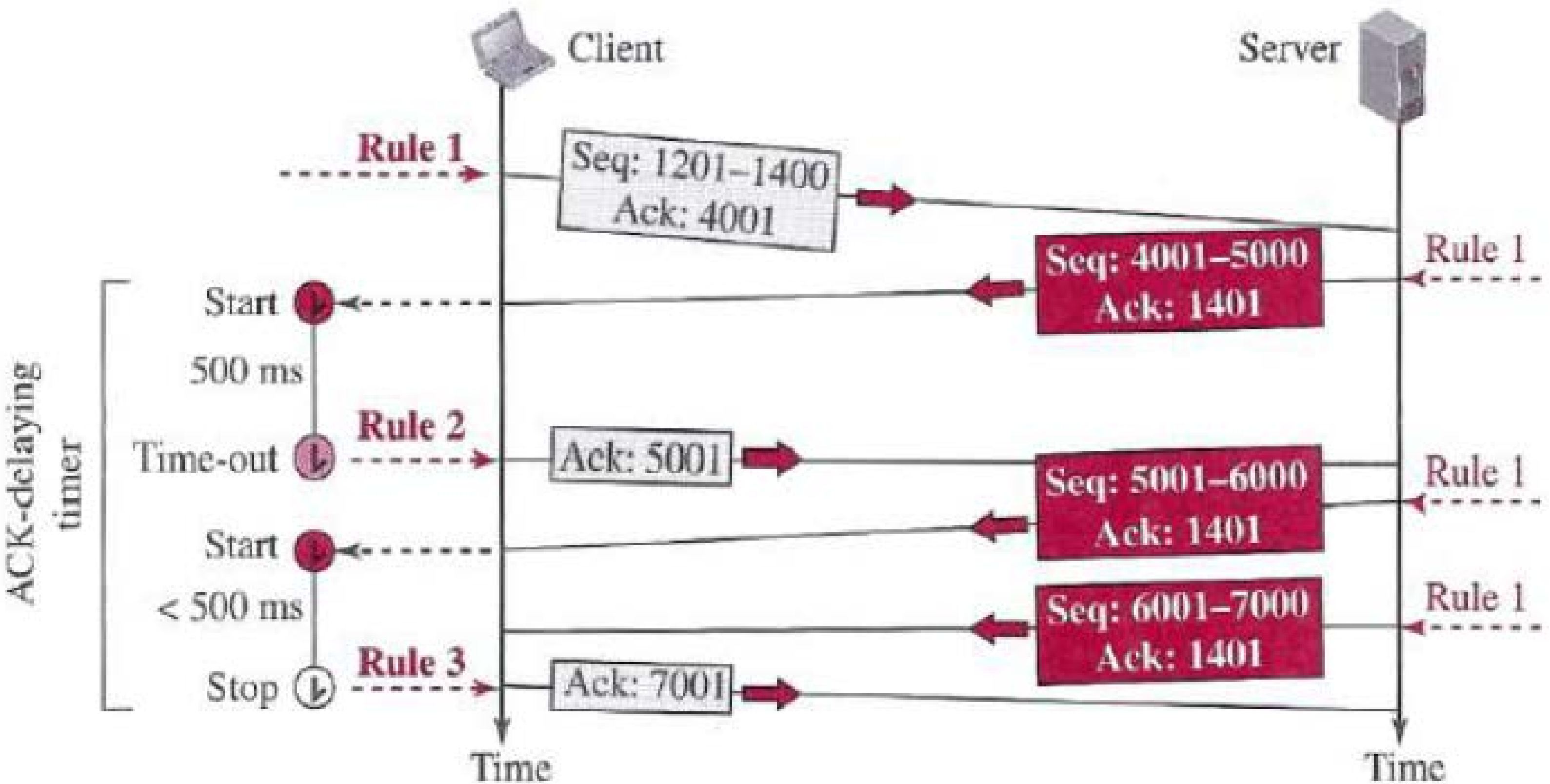
- When does a receiver generate ACK?
  - Rule-1: when node A sends data to node B, it piggybacks ACK
  - Rule-2: the receiver has no data to send and it receives an in-order segment, it delays sending ACK
  - Rule-3: there should not be more than two in-order unacknowledged segments at any time
  - Rule-4: 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…

- Retransmission
  - 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.
- 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.

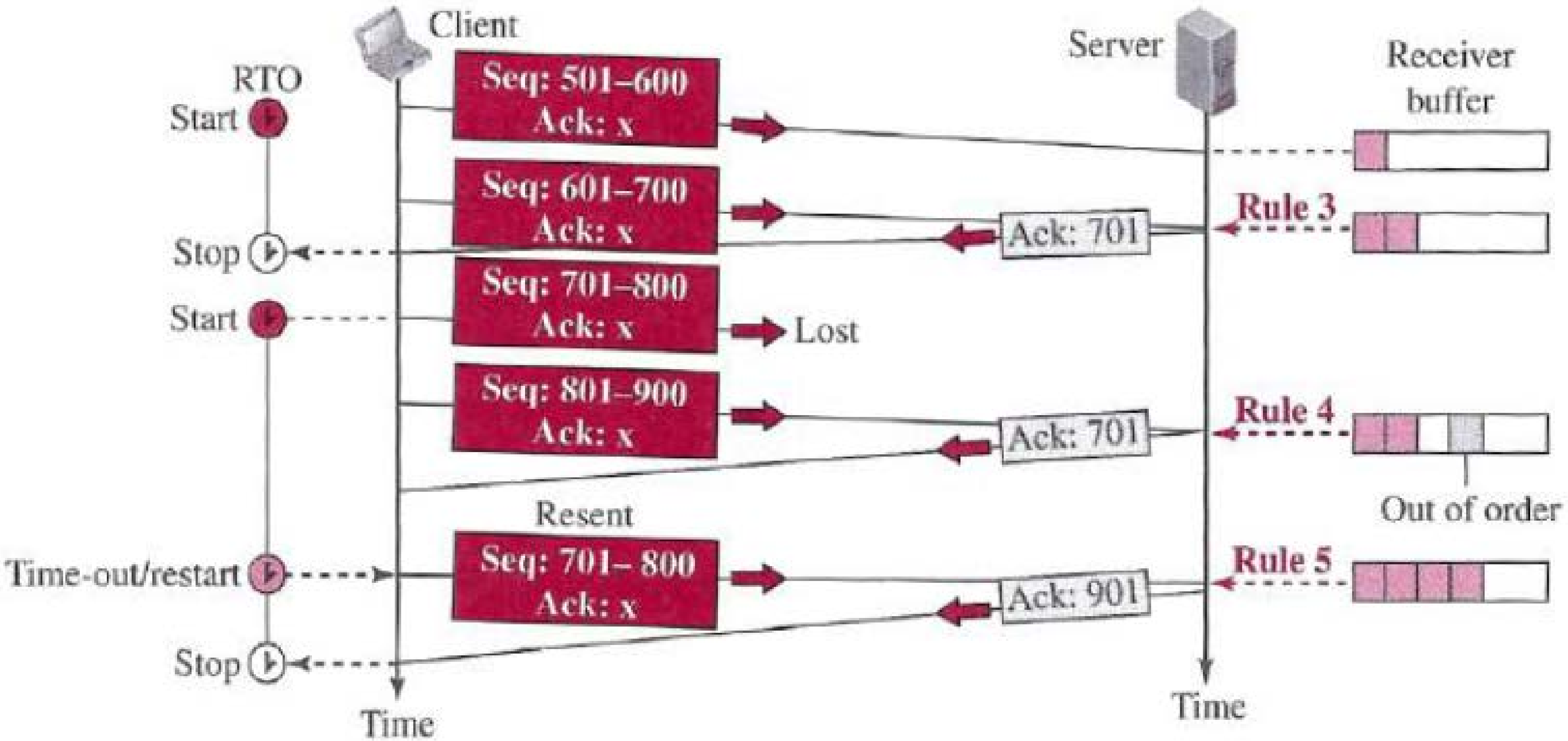
Example : Normal Scenario

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

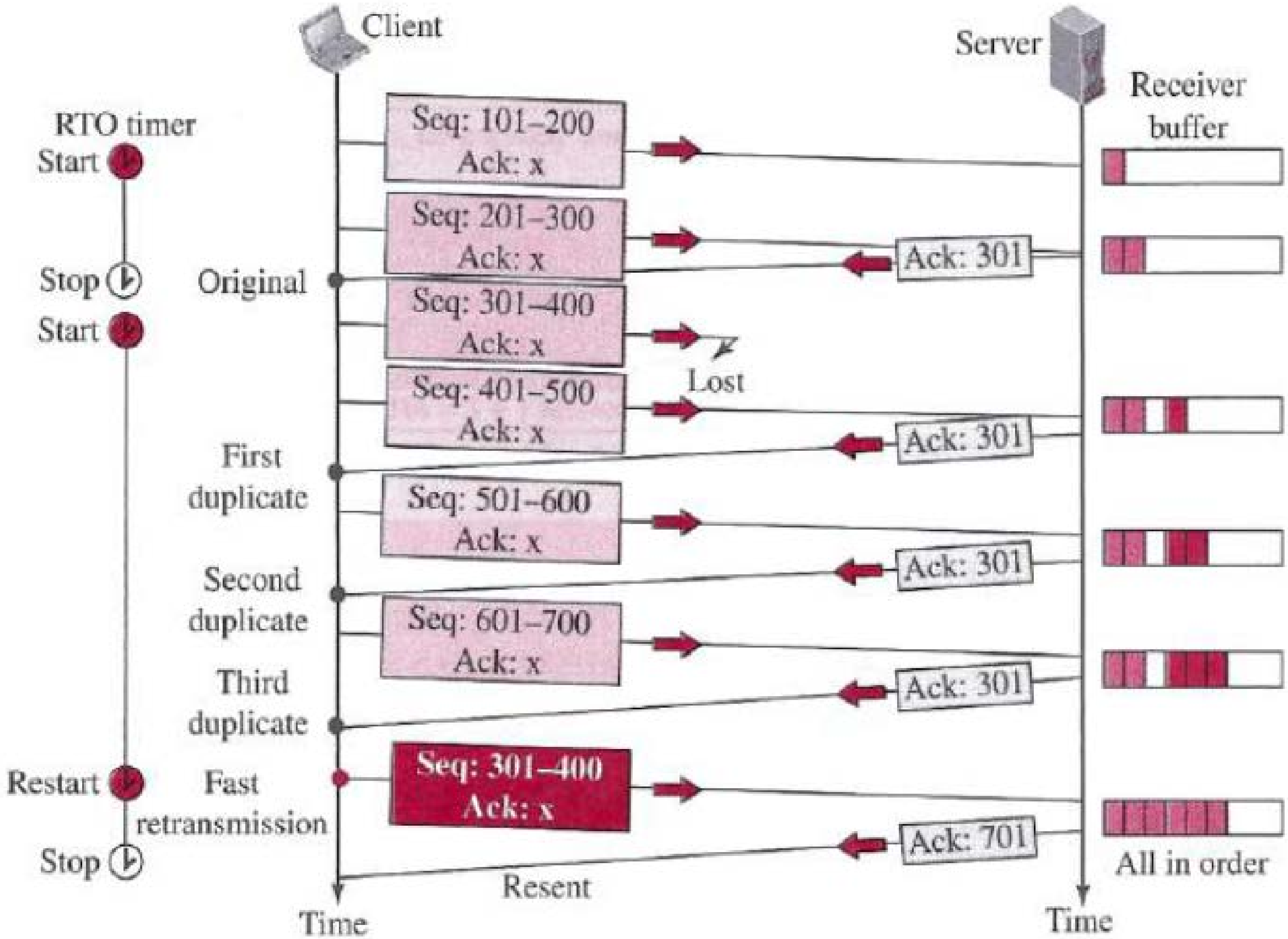


Example : Lost Segment Scenario

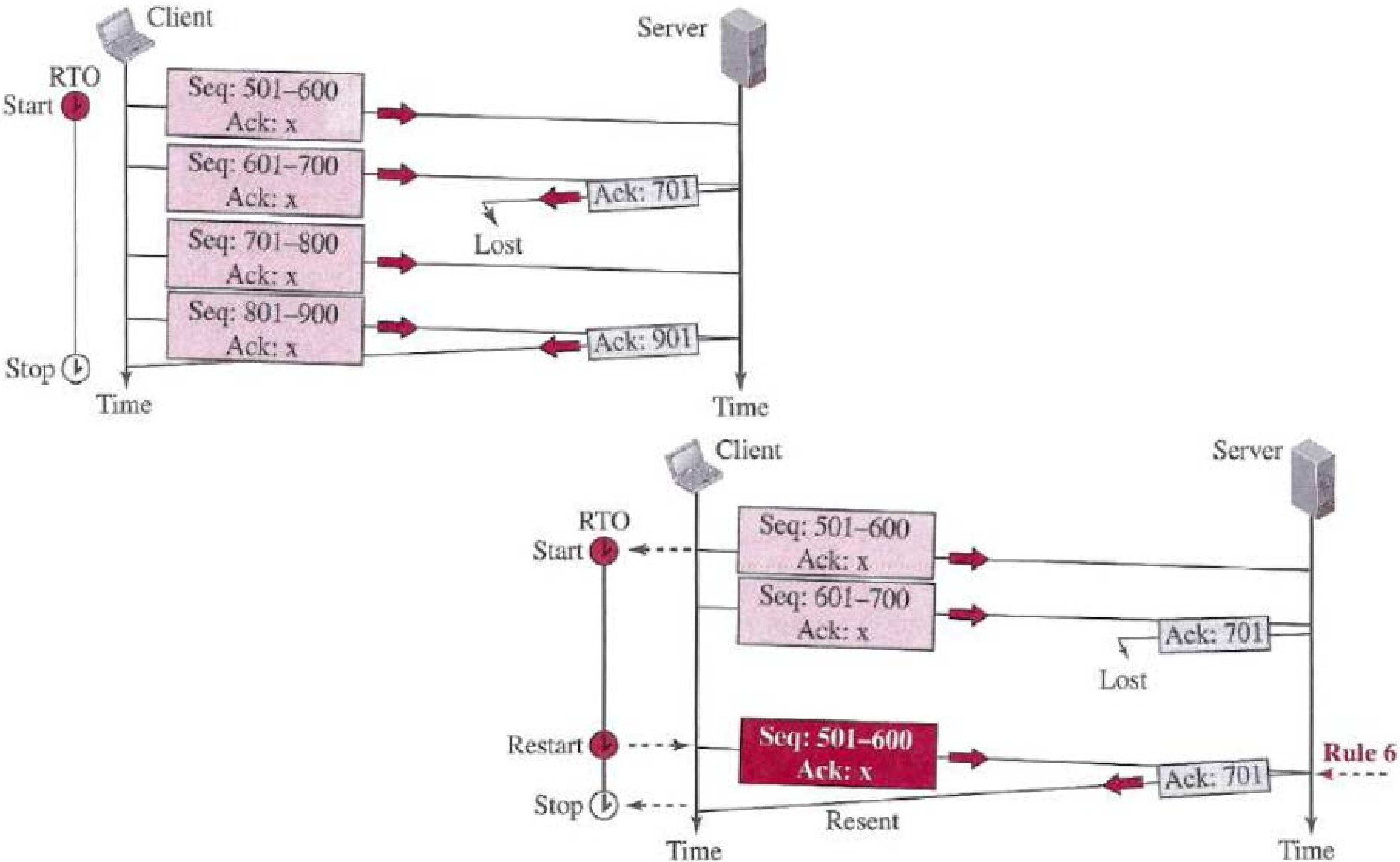
- assuming that data transfer is unidirectional
  - from client to server



Fast Retransmission



Lost ACK


Client
Server
RTO
Start
Seq: 501–600
Ack: x
Seq: 601–700
Ack: x
Ack: 701
Lost
Seq: 701–800
Ack: x
Seq: 801–900
Ack: x
Ack: 901
Stop
Time
Time
Client
Server
RTO
Start
Seq: 501–600
Ack: x
Seq: 601–700
Ack: x
Ack: 701
Lost
Restart
Seq: 501–600
Ack: x
Rule 6
Ack: 701
Stop
Resent
Time
Time

TCP Congestion Control

- The use of flow control in TCP cannot avoid congestion in intermediate routers as a router may receive data from more than one sender
- there is no congestion at the either end, but there may be congestion in the middle.

- TCP cannot ignore the congestion in network although it is an end-to-end protocol; it cannot aggressively send segments to the network; it cannot be very conservative, either, sending a small number of segments in each time interval;

- To control the number of segments to transmit, TCP uses another variable called Congestion Window (*cwnd*)
- The *cwnd* variable and the *rwnd* variable together define the size of the send window in TCP
  - Actual send window size = minimum *(rwnd, cwnd)*

Congestion Detection

- How a TCP sender can detect the possible existence of congestion in the network?

- TCP sender uses the occurrence of two events as signs of congestion:
  - time-out
  - receiving three duplicate ACKs

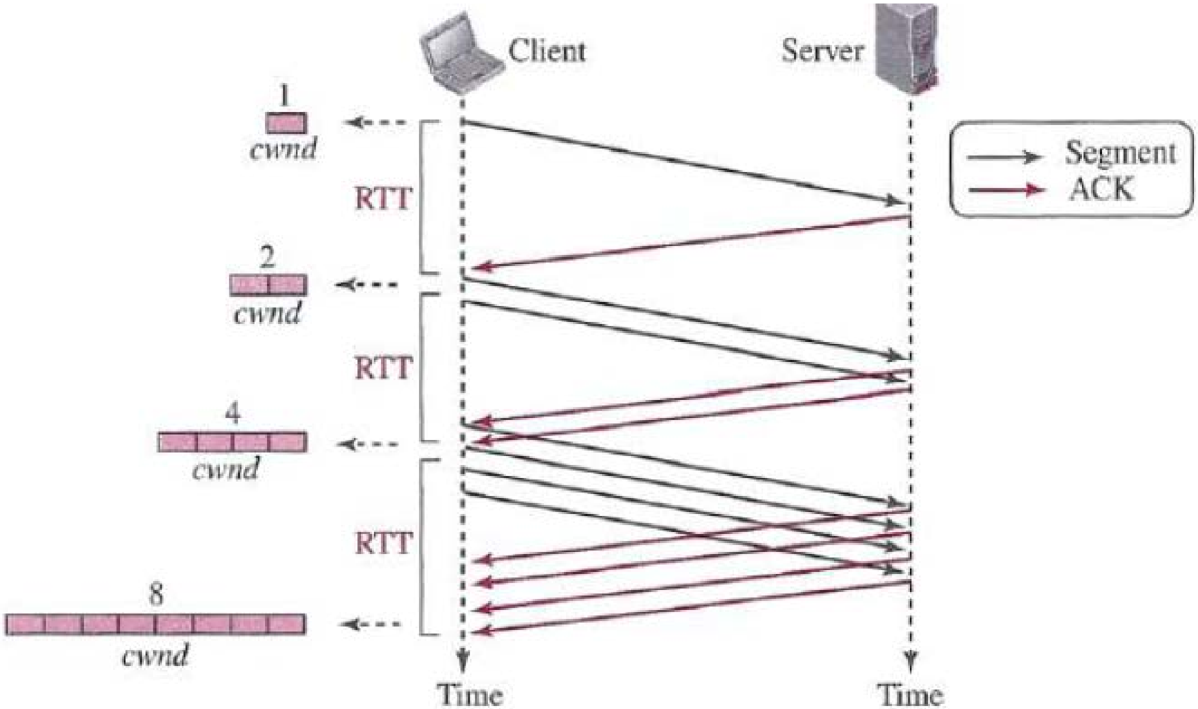
Congestion Control Policy

- Modified TCP with congestion control
  - Taho TCP: both signs of occurrence are treated equally
  - Reno TCP: both signs of occurrence are treated differently
  - New Reno TCP: TCP checks to see if more than one segment is lost in the current window when three duplicate ACKs arrive

- TCP uses three policies:
  - Slow start
  - Congestion avoidance
  - Fast recovery

Slow-start

- the size of the congestion window increases exponentially until it reaches a threshold
- the size of the congestion window is determined as follows: If an ACK arrives, *cwnd* = *cwnd* + 1.



Cont…

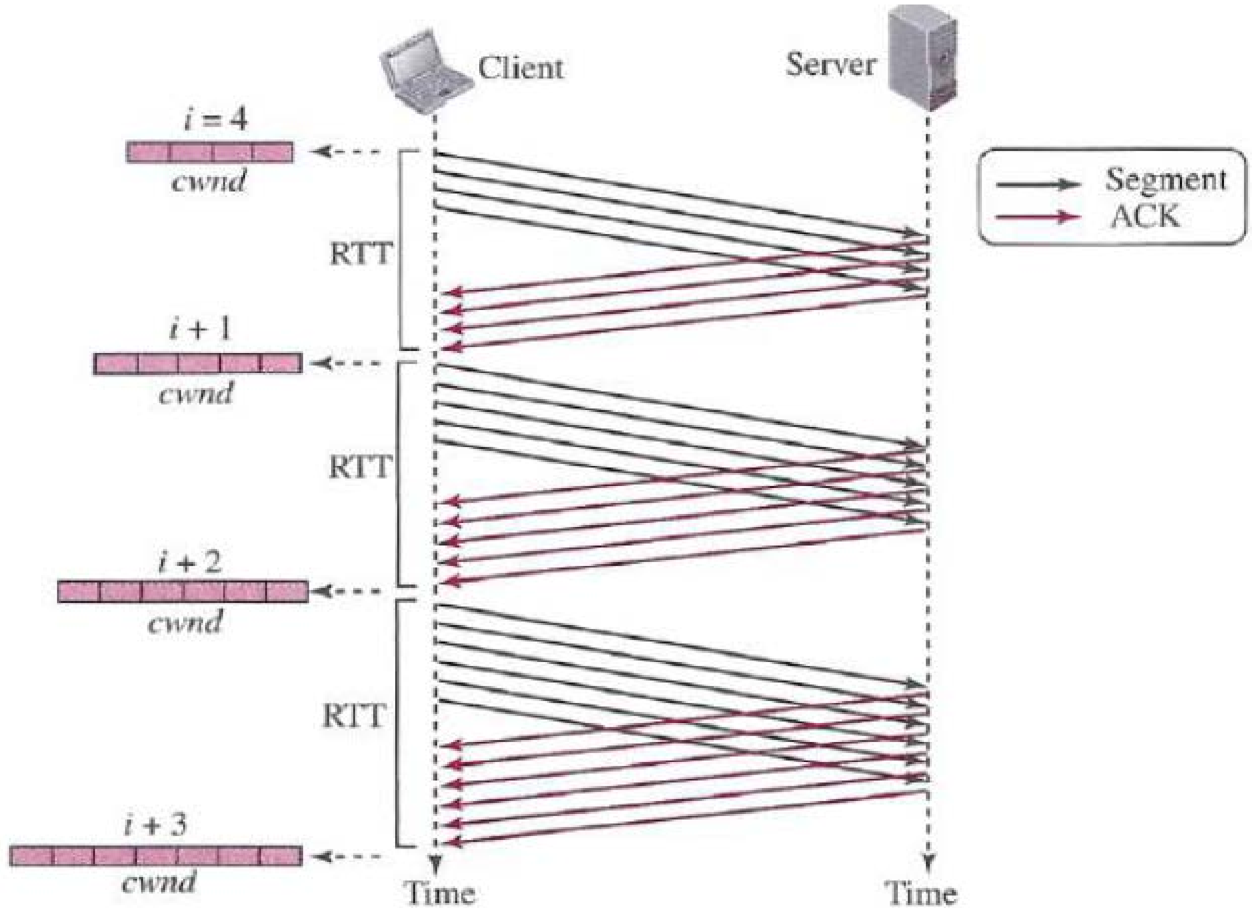
- slow-start strategy is slower in the case of delayed ACK.
- if two segments are acknowledged cumulatively, the size of the *cwnd* increases by only 1, not 2.
- With one ACK for every two segments, the growth is a power of 1.5, but still exponential

Congestion Avoidance

- To avoid congestion before it happens, we must slow down the exponential growth of *cwnd*
- increase the *cwnd* additively instead of exponentially.
- When the size of the *cwnd* reaches the slow-start threshold, the slow-start phase stops and the additive phase begins.

Cont…

- The size of the congestion window increases additively.
- If an ACK arrives, cwnd = cwnd + (1/ cwnd)



Fast Recovery

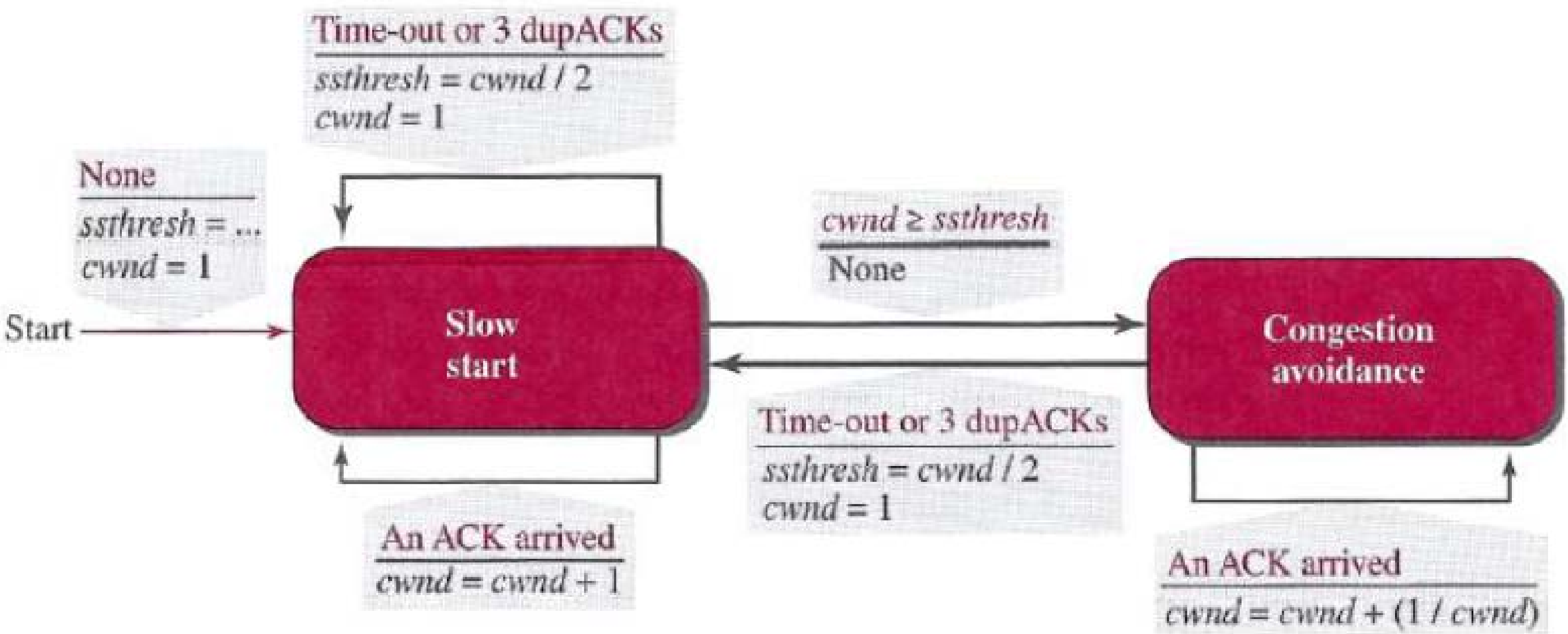
- this algorithm is also an additive increase, but it starts when three duplicate ACK arrives

- If a duplicate ACK arrives (after the three duplicate ACK which triggers the recovery)
  - $cwnd = cwnd + (1/ cwnd)$

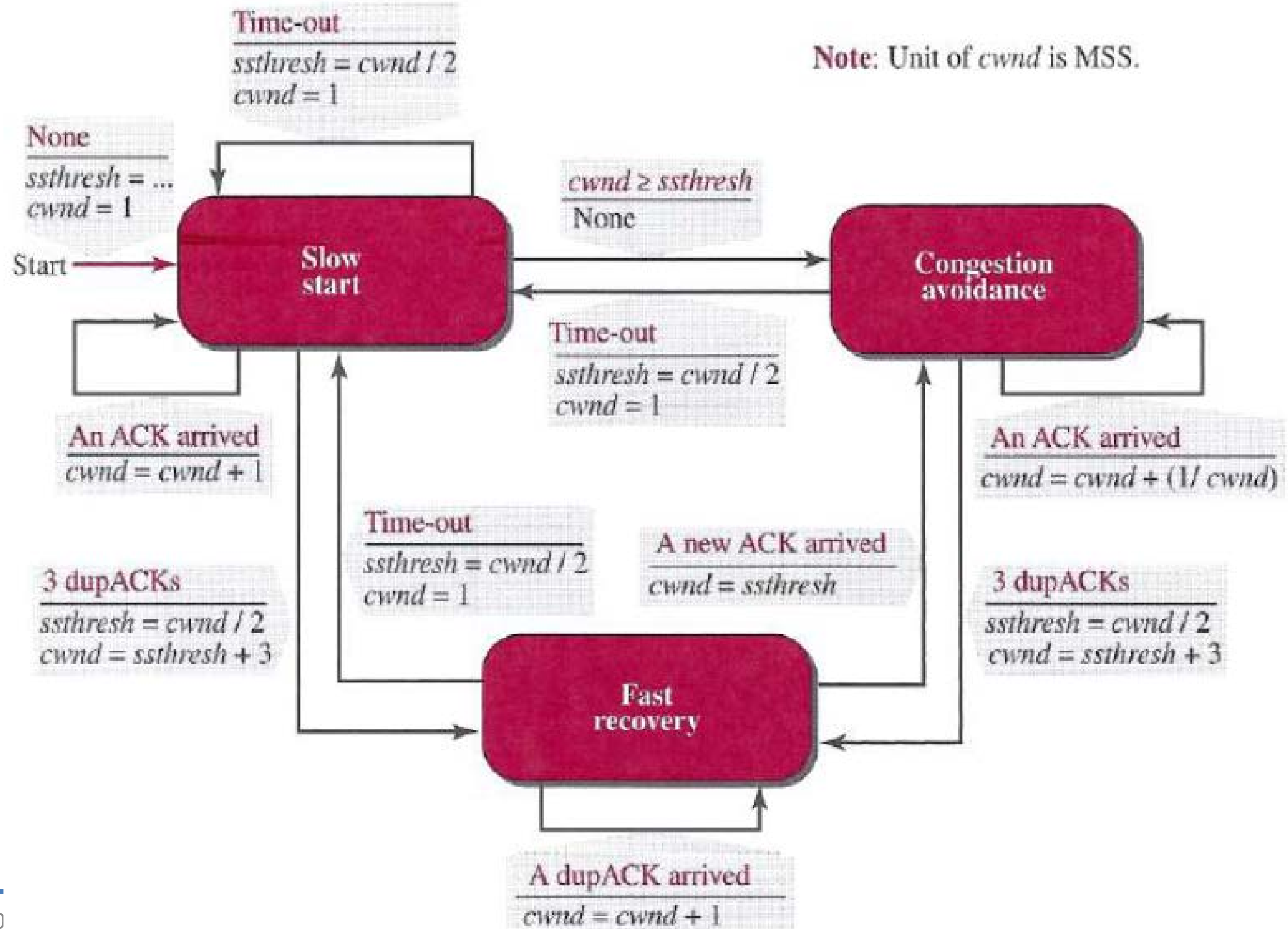
- This feature is optional in TCP

Taho TCP

- In Taho TCP
  - both signs of congestion occurrence (time-out, 3 duplicate ACK) are treated equally
  - uses only *slow start* and *congestion avoidance*.



Reno TCP



Time-out
ssthresh = cwnd / 2
cwnd = 1
Note: Unit of cwnd is MSS.
None
ssthresh = ...
cwnd = 1
cwnd ≥ ssthresh
None
Start
Slow start
Congestion avoidance
Time-out
ssthresh = cwnd / 2
cwnd = 1
An ACK arrived
cwnd = cwnd + 1
An ACK arrived
cwnd = cwnd + (1/ cwnd)
Time-out
ssthresh = cwnd / 2
cwnd = 1
A new ACK arrived
cwnd = ssthresh
3 dupACKs
ssthresh = cwnd / 2
cwnd = ssthresh + 3
3 dupACKs
ssthresh = cwnd / 2
cwnd = ssthresh + 3
Fast recovery
A dupACK arrived
cwnd = cwnd + 1

New Reno TCP

- It checks to see 3 duplicate ACK indicates 1-segment / 3-segments
- Reno TCP version is most common today
- AIMD: Additive Increase Multiplicative Decrease
- if we ignore the slow-start states and short exponential growth during fast recovery,
  - the TCP congestion window is $cwnd = cwnd + (1/cwnd)$ when an ACK arrives (congestion avoidance),
  - $cwnd = cwnd\ /2$ when congestion is detected

TCP Timers

- TCP uses at least four timers:
  - Retransmission : To retransmit lost segments,
  - Persistence: To deal with a zero-window-size advertisement
  - keepalive: to prevent a long idle connection between two TCPs.
  - TIME-WAIT: is used during connection termination.

TCP Throughput

- we'll ignore the slow-start phases that occur after timeout events. (These phases are typically very short)

- During a particular round-trip interval, the rate at which TCP sends data is a function of the congestion window (W) and the current *RTT*

- Assuming that *RTT* and *W* are approximately constant over the duration of the connection, the TCP transmission rate ranges from $W/(2 \cdot RTT)$ to W/RTT.

- Steady-state TCP throughput = 0.75*(W/RTT)

Thanks!