SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE NAME :19IT401 COMPUTER NETWORKS

II YEAR /IV SEMESTER

Unit 4-Transport layer

Topics 3,4 and 5 : Error control-flow control-congestion control

Flow control

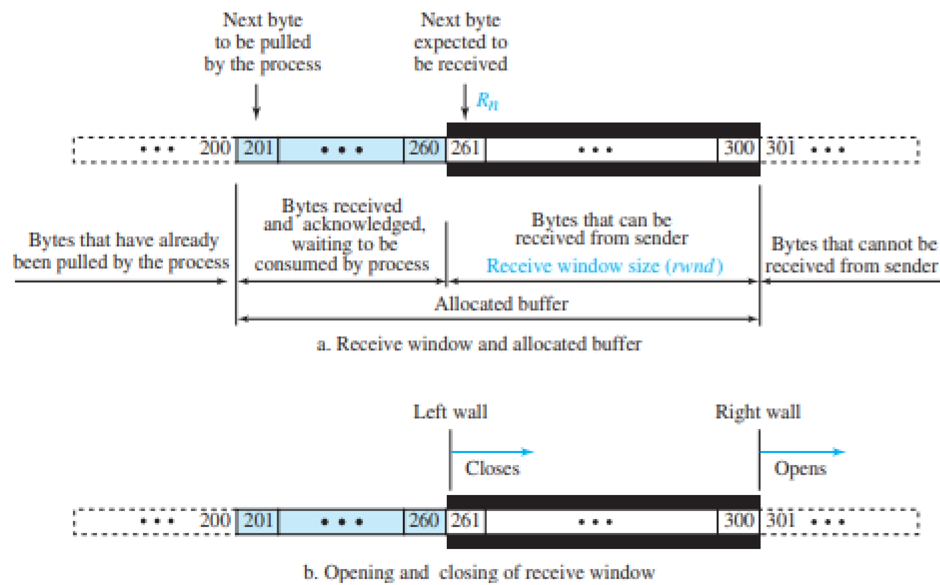
- Flow control is **a technique that allows two stations working at different speeds to communicate with each other**. It is a set of measures taken to regulate the amount of data that a sender sends so that a fast sender does not overwhelm a slow receiver.
- *Flow control balances the rate a producer creates data with the rate* a consumer can use the data.

- Send Window Figure 24.17 shows an example of a send window.
- The window size is 100 bytes, but later we see that the send window size is dictated by the receiver (flow control) and the congestion in the underlying network (congestion control).
- The figure shows how a send window opens, closes, or shrinks.

The send window in TCP is similar to the one used with the Selective-Repeat protocol, but with some differences: 1. One difference is the nature of entities related to the window. The window size in SR is the number of packets, but the window size in TCP is the number of bytes. Although actual transmission in TCP occurs segment by segment, the variables that control the window are expressed in bytes. 2. The second difference is that, in some implementations, TCP can store data received from the process and send them later, but we assume that the sending TCP is capable of sending segments of data as soon as it receives them from its process. 3. Another difference is the number of timers. The theoretical Selective-Repeat protocol may use several timers for each packet sent, but as mentioned before, the TCP protocol uses only one timer.

Receive Window Figure 24.18 shows an example of a receive window. The window size is 100 bytes. The figure also shows how the receive window opens and closes; in practice, the window should never shrink

Figure 24.18 *Receive window in TCP*



There are two differences between the receive window in TCP and the one we used for SR.

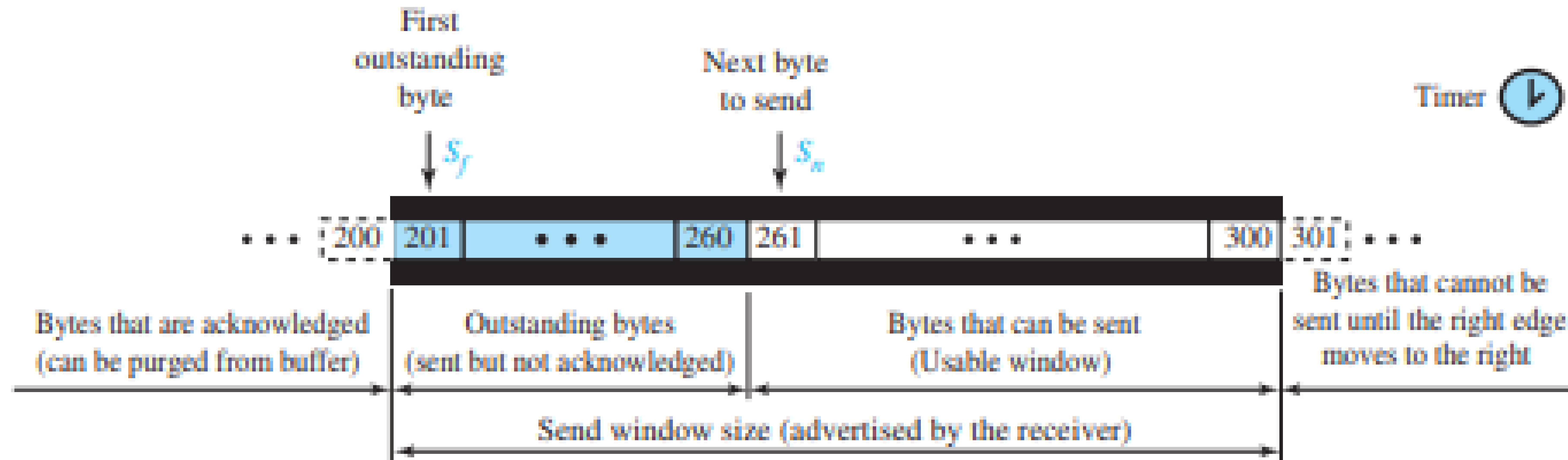
1. The first difference is that TCP allows the receiving process to pull data at its own pace. This means that part of the allocated buffer at the receiver may be occupied by bytes that have been received and acknowledged, but are waiting to be pulled by the receiving process. The receive window size is then always smaller than or equal to the buffer size, as shown in Figure 24.18. The receive window size determines the number of bytes that the receive window can accept from the sender before being overwhelmed (flow control). In other words, the receive window size, normally called *rwnd*, can be determined as:

   $$rwnd = \textbf{buffer size} - \textbf{number of waiting bytes to be pulled}$$

2. The second difference is the way acknowledgments are used in the TCP protocol. Remember that an acknowledgement in SR is selective, defining the uncorrupted packets that have been received. The major acknowledgment mechanism in TCP is a cumulative acknowledgment announcing the next expected byte to receive (in this way TCP looks like GBN, discussed earlier). The new version of TCP, however, uses both cumulative and selective acknowledgments; we will discuss these options on the book website.

There are two differences between the receive window in TCP and the one we used for SR. 1. The first difference is that TCP allows the receiving process to pull data at its own pace. This means that part of the allocated buffer at the receiver may be occupied by bytes that have been received and acknowledged, but are waiting to be pulled by the receiving process. The receive window size is then always smaller than or equal to the buffer size, as shown in Figure 24.18. The receive window size determines the number of bytes that the receive window can accept from the sender before being overwhelmed (flow control). In other words, the receive window size, normally called rwnd, can be determined as: 2. The second difference is the way acknowledgments are used in the TCP protocol. Remember that an acknowledgement in SR is selective, defining the uncorrupted packets that have been received. The major acknowledgment mechanism in TCP is a cumulative acknowledgment announcing the next expected byte to receive (in this way TCP looks like GBN, discussed earlier). The new version of TCP, however, uses both cumulative and selective acknowledgments; we will discuss these options on the book website.

Figure 24.17 *Send window in TCP*



a. Send window

b. Opening, closing, and shrinking send window

Data flow and flow control feedbacks in TCP

Data flow
Flow control feedback

Sender
Application layer
Producer

Receiver
Consumer
Application layer

Messages are pushed (1)

(5) Flow control feedback

(3) Messages are pulled

Transport layer
Consumer
Producer

(2) Segments are pushed

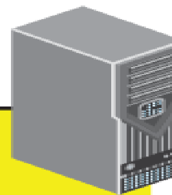
Producer
Consumer
Transport layer

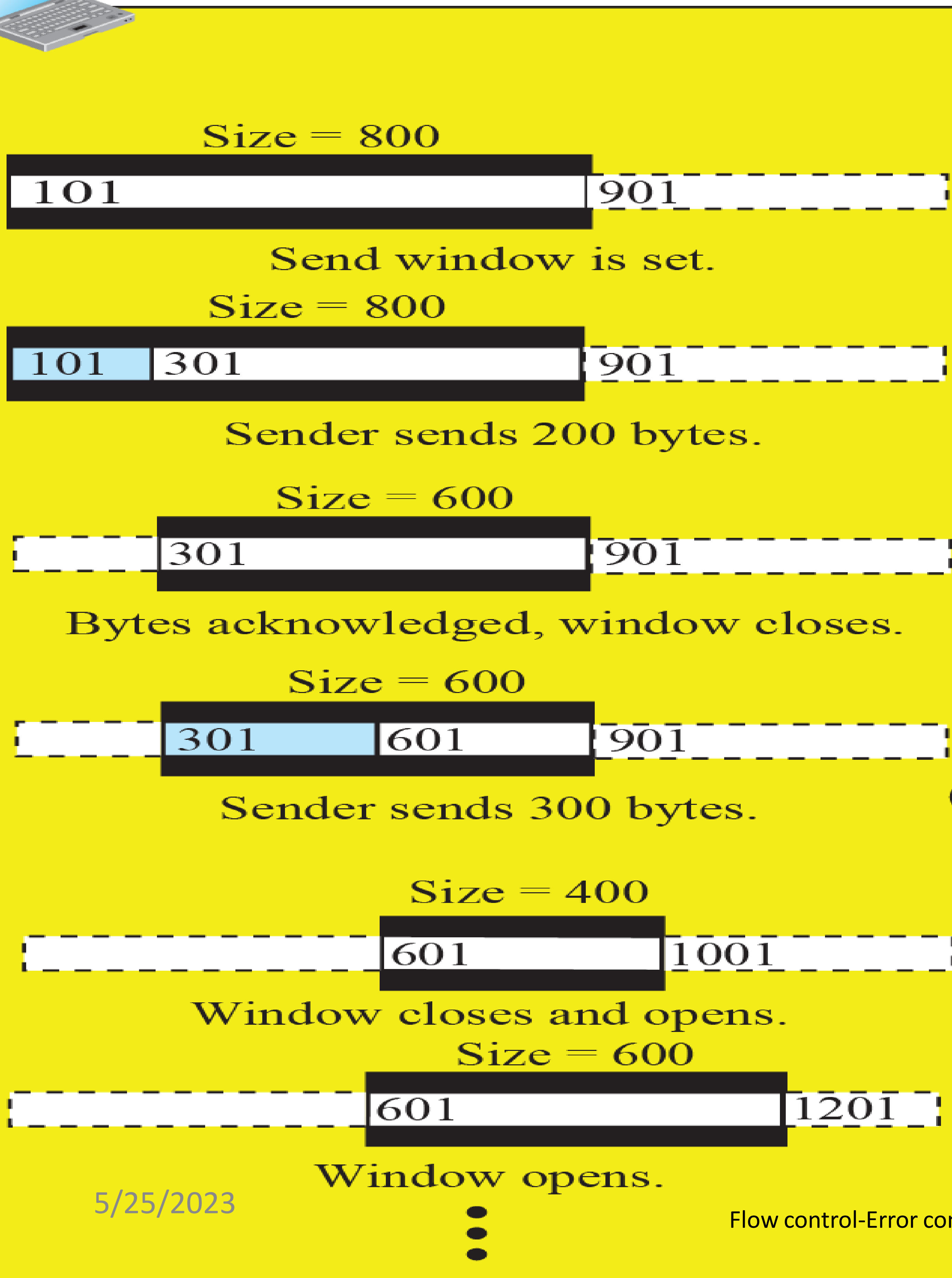
(4) Flow control feedback

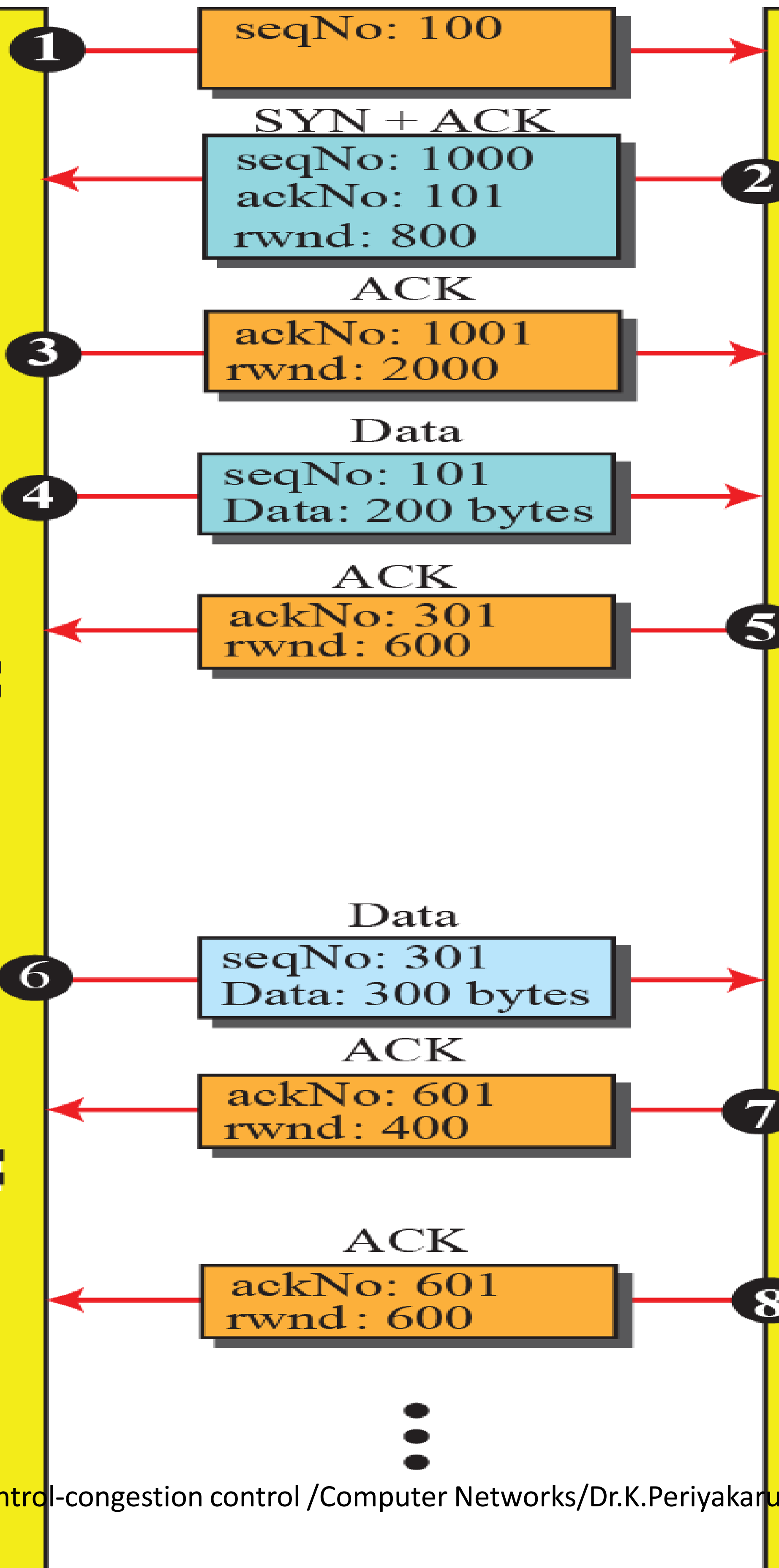
Eight segments are exchanged between the client and server:

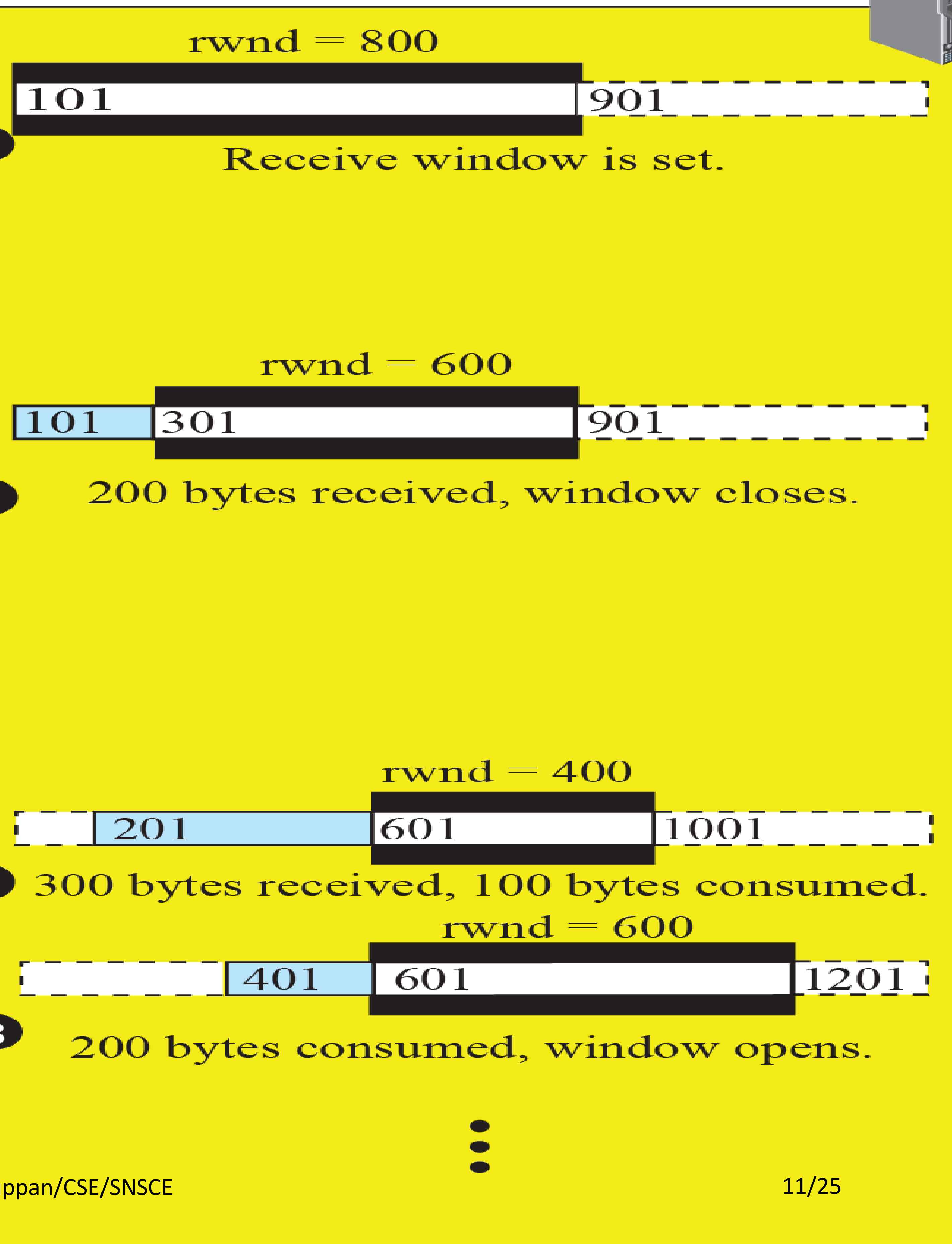
1. The first segment is from the client to the server (a SYN segment) to request connection.
The client announces its initial seqNo = 100. When this segment arrives at the server, it allocates a buffer size of 800 (an assumption) and sets its window to cover the whole buffer (*rwnd = 800). Note that the number of the next byte to arrive is 101.*
2. The second segment is from the server to the client. This is an ACK + SYN segment.
The segment uses ackNo = 101 to show that it expects to receive bytes starting from 101. It also announces that the client can set a buffer size of 800 bytes.
3. The third segment is the ACK segment from the client to the server. Note that the client has defined a *rwnd of size 2000, but we do not use this value* because the communication is only in one direction.
4. Segment 4: the process pushes 200 bytes of data. The TCP client numbers these bytes 101 to 300. It then creates a segment and sends it to the server. When this segment is received at the server, the bytes are stored, and the receive window closes to show that the next byte expected is byte 301; the stored bytes occupy 200 bytes of buffer
5. Segment 5:The server acknowledges bytes up to and including 300 (expecting to receive byte 301). The segment also carries the size of the receive window after decrease (600). The client, after receiving this segment, purges the acknowledged bytes from its window and closes its window to show that the next byte to send is byte 301.
6.Segment 6 is sent by the client after its process pushes 300 more bytes. The segment defines seqNo as 301 and contains 300 bytes. When this segment arrives at the server, the server stores them, but it has to reduce its window size. After its process has pulled 100 bytes of data, the window closes from the left for the amount of 300 bytes, but opens from the right for the amount of 100 bytes. The result is that the size is only reduced 200 bytes. The receiver window size is now 400 bytes.
7.In segment 7, the server acknowledges the receipt of data, and announces that its window size is 400 The send window closes from the left by 300 bytes, and opens from the right by 100 bytes.
8.Segment 8 is also from the server after its process has pulled another 200 bytes. Its window size increases. The new *rwnd value is now 600* After this segment arrives at the client, the client opens its window by 200 bytes without closing it. The result is that its window size increases to 600 bytes.

Note: We assume only unidirectional communication from client to server. Therefore, only one window at each side is shown.









Error Control

TCP is a reliable transport-layer protocol. This means that an application program that delivers a stream of data to the application program on the other end in order, without error, and without any part lost or duplicated.

TCP provides reliability using error control.

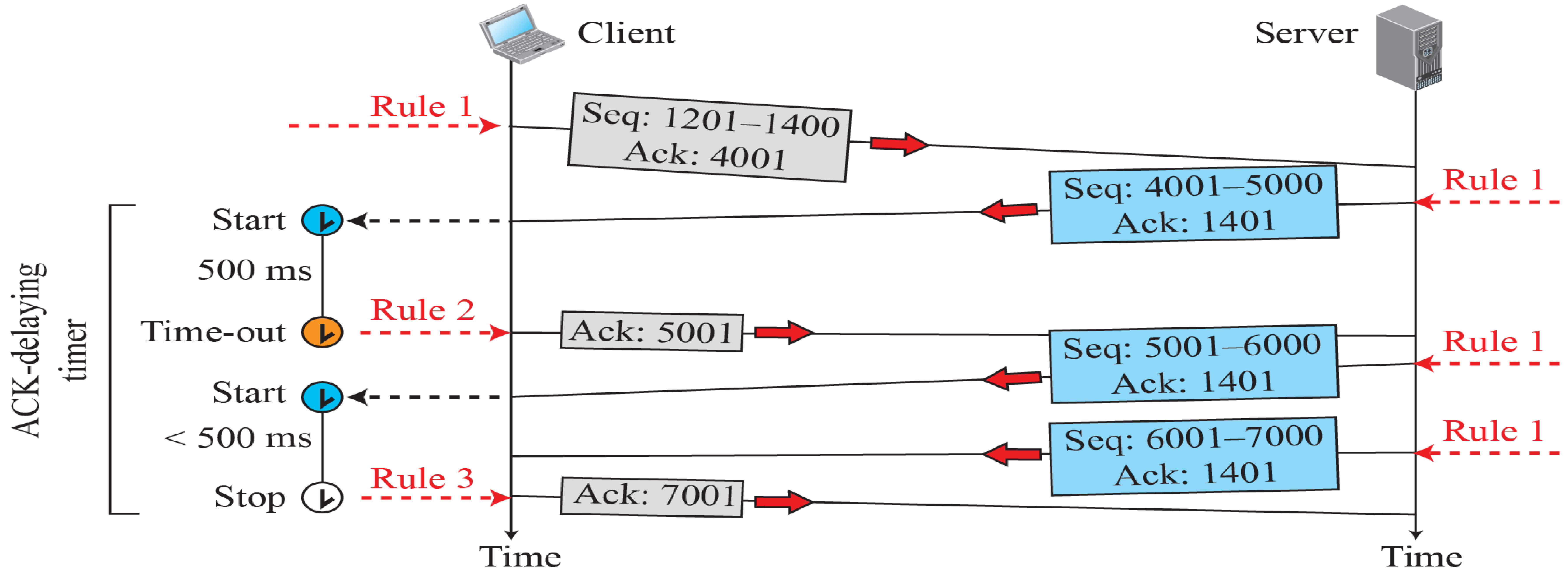
Error control includes mechanisms for detecting and resending corrupted segments, resending lost segments, storing out-of order segments until missing segments arrive, and detecting and discarding duplicated segments.

Error control in TCP is achieved through the use of three simple tools: checksum, acknowledgment, and time-out.

Generating Acknowledgments

1. When end A sends a data segment to end B, it must include (piggyback) an acknowledgment that gives the next sequence number it expects to receive. This rule decreases the number of ACKs needed and therefore reduces traffic.
2. When the receiver has no data to send and it receives an in-order segment (with expected sequence number) and the previous segment has already been acknowledged, the receiver delays sending an ACK segment until another segment arrives or until a period of time (normally 500 ms) has passed. In other words, the receiver needs to delay sending an ACK segment if there is only one outstanding in-order segment. This rule reduces ACK segments.
3. When a segment arrives with a sequence number that is expected by the receiver, and the previous in-order segment has not been acknowledged, the receiver immediately sends an ACK segment. In other words, **there should not be more than two in-order unacknowledged segments at any time**. This prevents the unnecessary retransmission of segments that may create congestion in the network.
4. When a segment arrives with an out-of-order sequence number that is higher than expected, the receiver immediately sends an ACK segment announcing the sequence number of the next expected segment. This leads to the *fast retransmission* of missing segments
5. When a missing segment arrives, the receiver sends an ACK segment to announce the next sequence number expected. This informs the receiver that segments reported missing have been received.
6. If a duplicate segment arrives, the receiver discards the segment, but immediately sends an acknowledgment indicating the next in-order segment expected. This solves some problems when an ACK segment itself is lost.

Normal operation



Lost segment

A lost or corrupted segment is treated the same way by the receiver.
A lost segment is discarded somewhere in the network; a corrupted segment is discarded by the receiver itself.

Lost segment

Client

Server

Receiver buffer

RTO

Start

Seq: 501–600
Ack: x

Seq: 601–700
Ack: x

Rule 3

Ack: 701

Stop

Start

Seq: 701–800
Ack: x

Lost

Seq: 801–900
Ack: x

Rule 4

Ack: 701

Out of order

Resent

Time-out/restart

Seq: 701– 800
Ack: x

Rule 5

Ack: 901

Stop

Time

Time

Fast Retransmission

- ✓Each time the receiver receives a subsequent segment, it triggers an acknowledgment
- ✓The sender receives four acknowledgments with the same value (three duplicates).
- ✓the segment that is expected by all of these duplicate acknowledgments, be resent immediately.
- ✓ After resending this segment, the timer is restarted.

Fast retransmission

Client

Server

Receiver buffer

RTO timer

Start

Seq: 101–200
Ack: x

Seq: 201–300
Ack: x

Ack: 301

Stop

Original

Start

Seq: 301–400
Ack: x

Lost

Seq: 401–500
Ack: x

Ack: 301

First duplicate

Seq: 501–600
Ack: x

Ack: 301

Second duplicate

Seq: 601–700
Ack: x

Third duplicate

Ack: 301

Restart

Fast retransmission

Seq: 301–400
Ack: x

Resent

Ack: 701

All in order

Stop

Time

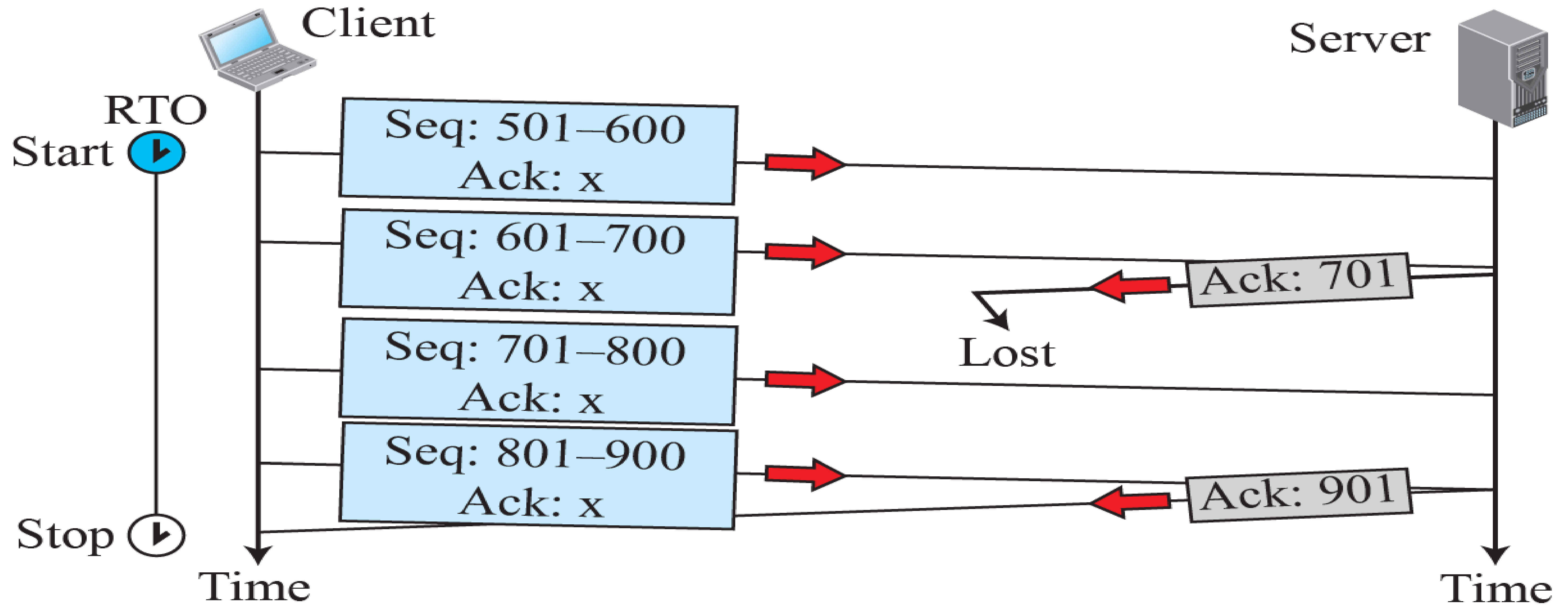
Time

Automatically Corrected Lost ACK

✓In the TCP acknowledgment mechanism, a lost acknowledgment may not even be noticed by the source TCP.

✓ TCP uses cumulative acknowledgment. We can say that the next acknowledgment automatically corrects the loss of the previous acknowledgment

Lost acknowledgment



Lost Acknowledgment Corrected by Resending a Segment

If the next acknowledgment is delayed for a long time or there is no next acknowledgment (the lost acknowledgment is the last one sent), the correction is triggered by the RTO timer.

A duplicate segment is the result. When the receiver receives a duplicate segment, it discards it and resends the last ACK immediately to inform the sender that the segment or segments have been received.

Lost acknowledgment corrected by resending a segment

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

In flow control the size of the send window is controlled by the receiver using the value of *rwnd, which is advertised in each* segment traveling in the opposite direction.
The use of this strategy guarantees that the receive window is never overflowed with the received bytes (no end congestion).
This, however, does not mean that the intermediate buffers, buffers in the routers, do not become congested. A router may receive data from more than one sender.
there is no congestion at the other end, but there may be congestion in the middle. TCP needs to worry about congestion in the middle because many segments lost may seriously affect the error control.
More segment loss means resending the same segments again, resulting in worsening the congestion, and finally the collapse of the communication.

Congestion Detection

The TCP sender uses the occurrence of two events as signs of congestion in the network:

1. time-out
2. receiving three duplicate ACKs.

The first is the *time-out. If a TCP sender does not receive an ACK for a segment or* a group of segments before the time-out occurs, it assumes that the corresponding segment or segments are lost and the loss is due to congestion.

Another event is the receiving of three duplicate ACKs (four ACKs with the same acknowledgment number

sending three duplicate ACKs is the sign of a missing segment, which can be due to congestion in the network.

Congestion Policies

TCP’s general policy for handling congestion is based on three algorithms:

1. slow start
2. congestion avoidance
3. fast recovery

Slow Start: Exponential Increase

The slow-start algorithm is based on the idea that the size of the congestion window (*cwnd) starts with one maximum segment size (MSS), but it increases one MSS each* time an acknowledgment arrives.

the size of the congestion window in this algorithm is a function of the number of ACKs arrived and can be determined as follows.

If an ACK arrives, *cwnd = cwnd + 1.*

If we look at the size of the *cwnd in terms of round-trip times (RTTs), we find that* the growth rate is exponential in terms of each round trip time, which is a very aggressive approach:

Start → *cwnd = 1 → 20*

After 1 RTT → *cwnd = cwnd + 1 = 1 + 1 = 2→ 21*

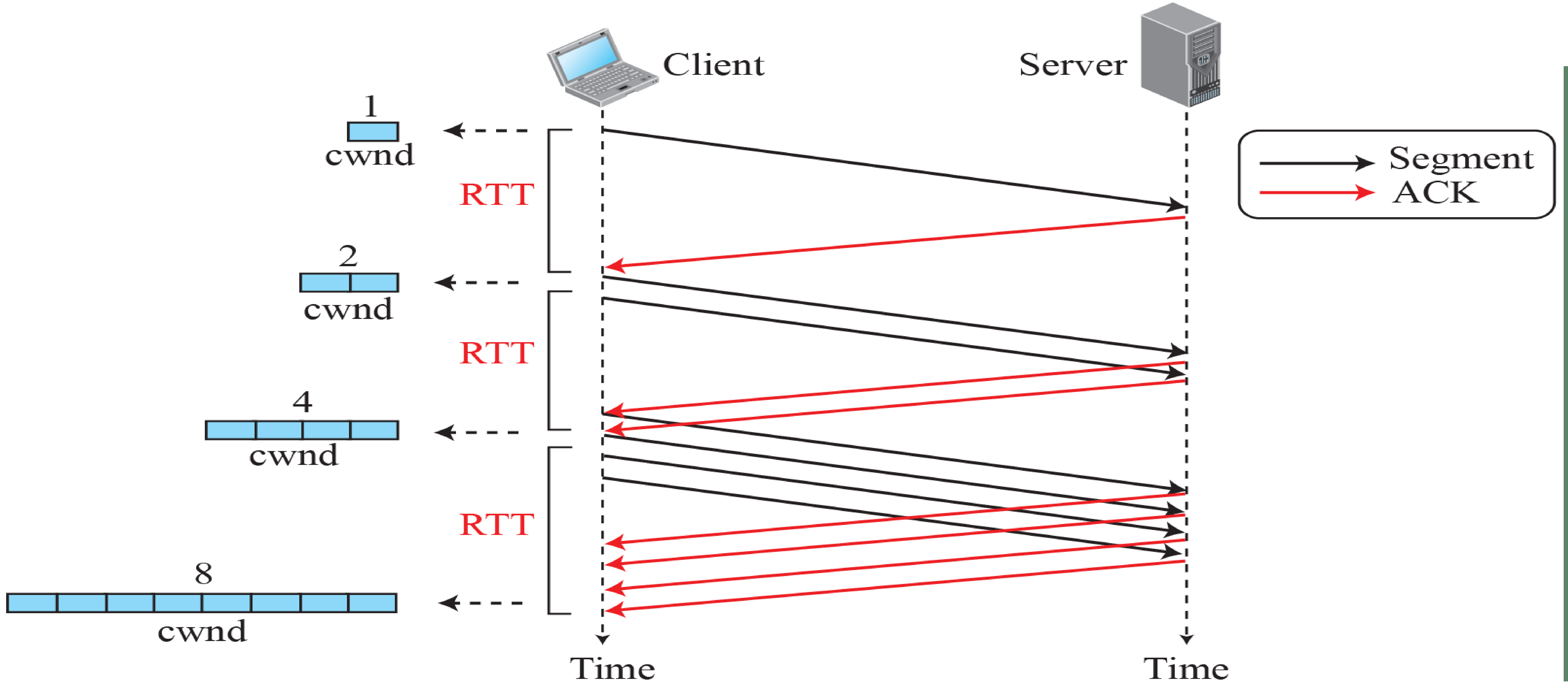
After 2 RTT → *cwnd = cwnd + 2 = 2 + 2 = 4→ 22*

After 3 RTT → *cwnd = cwnd + 4 = 4 + 4 = 8→ 23*

A slow start cannot continue indefinitely. There must be a threshold to stop this phase. The sender keeps track of a variable named *ssthresh (slow-start threshold).*

When the size of the window in bytes reaches this threshold, slow start stops and the next phase starts.

Slow start, exponential increase



Congestion Avoidance: Additive Increase

If we continue with the slow-start algorithm, the size of the congestion window increases exponentially. To avoid congestion before it happens, we must slow down this exponential growth. TCP defines another algorithm called *congestion avoidance,* which increases the *cwnd additively instead of exponentially.*

When the size of the congestion window reaches the slow-start threshold in the case where *cwnd = i,* the slow-start phase stops and the additive phase begins.

In this algorithm, each time the whole "window" of segments is acknowledged, the size of the congestion window is increased by one.

Start → *cwnd = i*

After 1 RTT → *cwnd = i + 1*

After 2 RTT → *cwnd = i + 2*

After 3 RTT → *cwnd = i + 3*

In the congestion-avoidance algorithm, the size of the congestion window increases additively until congestion is detected.

Congestion avoidance, additive increase

Assessment

a) What is flow controol?
b) What is Error control?
c) What is congestion control?

Reference

TEXT BOOKS

Behrouz A. Forouzan, Data Communications and Networking, Fifth Edition TMH, 2013.

REFERENCES

1. William Stallings, Data and Computer Communications, Tenth Edition, Pearson Education, 2013.
2. Andrew Tanenbaum, Computer Networks, Fifth Edition, Pearson (5th Edition) Education, 2013.
3. James F. Kurose, Keith W. Ross, Computer Networking, A Top-Down Approach Featuring the Internet, Sixth Edition, Pearson Education, 2013.
4. Larry L. Peterson, Bruce S. Davie, Computer Networks: A Systems Approach, Fifth Edition, Morgan Kaufmann Publishers Inc., 2012.