Data Communication
(DC)

Lecture 9b

Overview of the contents

- **TCP features**
- **TCP segments**
- **TCP connection**
- **TCP state transition diagram**
- **Windows in TCP**
- **Flow control in TCP**
- **Error control in TCP**

TCP services

Transport Layer

Transmission Control Protocol (TCP) [protocol 6]

Like UDP, this protocol is process-to-process, but unlike UDP, TCP is a connection-oriented protocol.

TCP creates a logical connection between two processes that want to communicate.

TCP applies flow and error controls at the transport layer level, which makes TCP a reliable protocol.

TCP adds connection-oriented and reliable features to the IP service.

Transport Layer

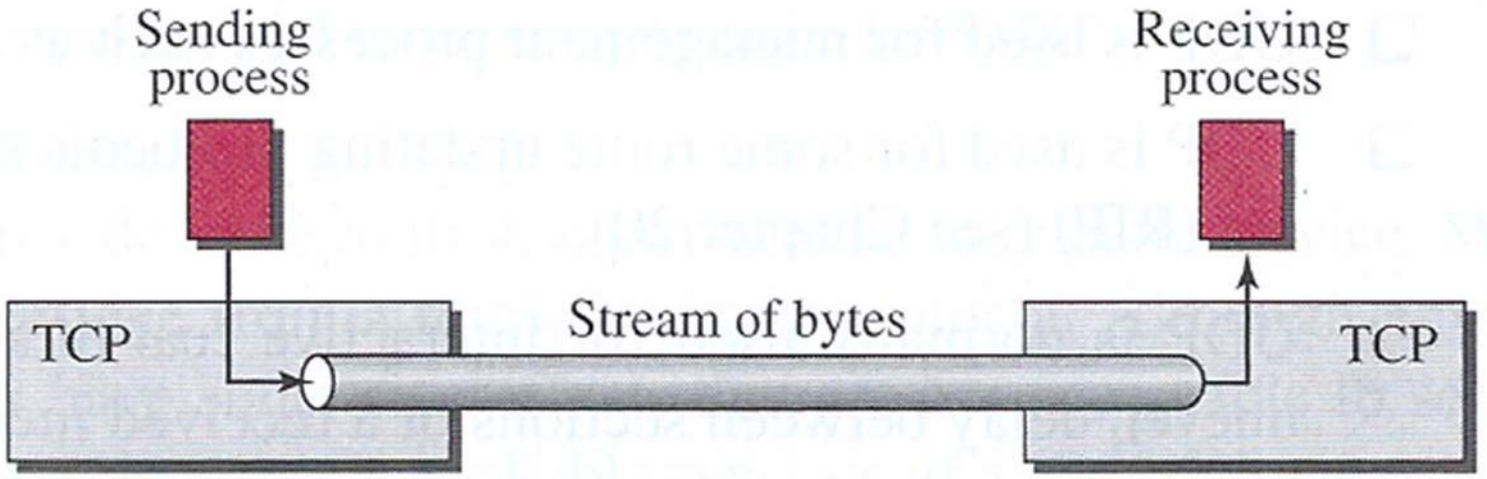
Transmission Control Protocol (TCP): Stream delivery service

TCP is, unlike UDP, a stream-oriented protocol.

In UDP, a process sends its messages with some predefined sizes. UDP adds a header and forwards them to IP for encapsulation and transmission. Neither UDP nor IP perceives a connection between datagrams.

However, TCP allows a sending process to deliver a data stream, and the protocol also allows a receiving process to receive a data stream.

TCP creates an environment in which the two processes seem to be connected by an imaginary "tube" that carries their bytes across the Internet.



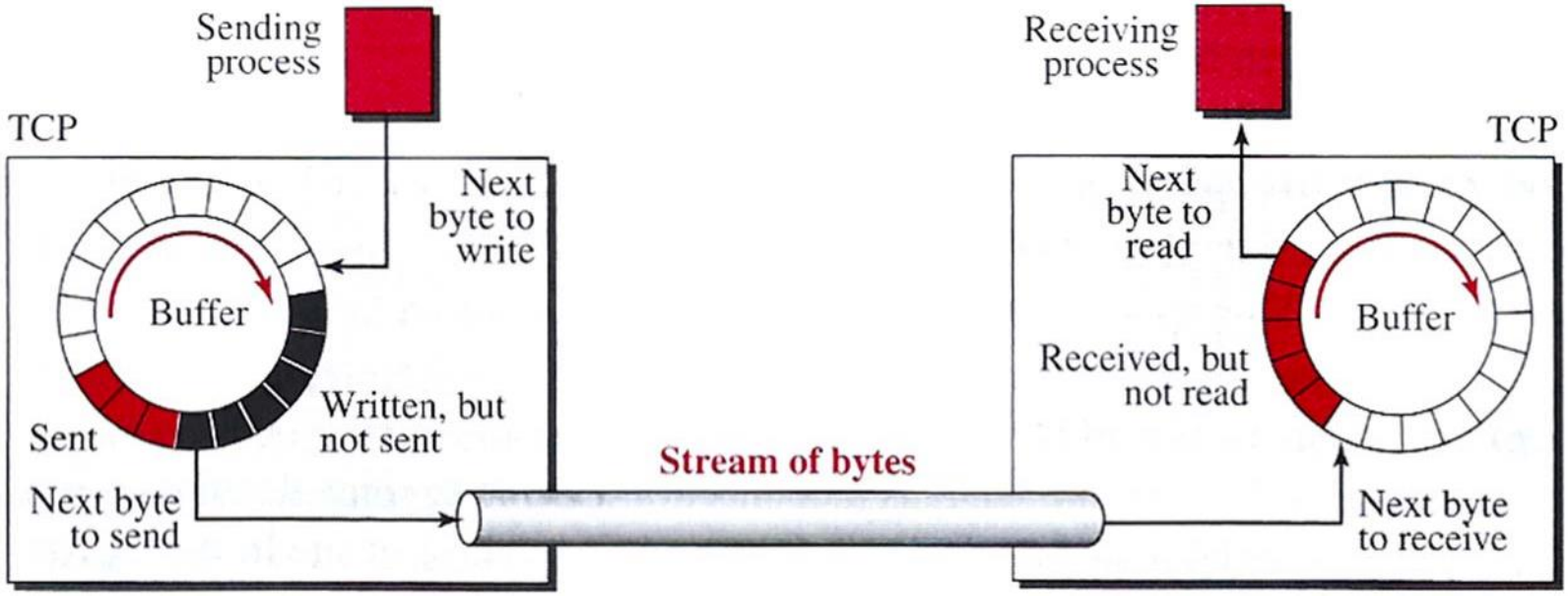
Transport Layer

Transmission Control Protocol (TCP): Sending and receiving buffers

Since the sending and receiving processes may not write and read at the same speed, buffers are needed.
There are two buffers, a sending buffer and a receiving buffer.

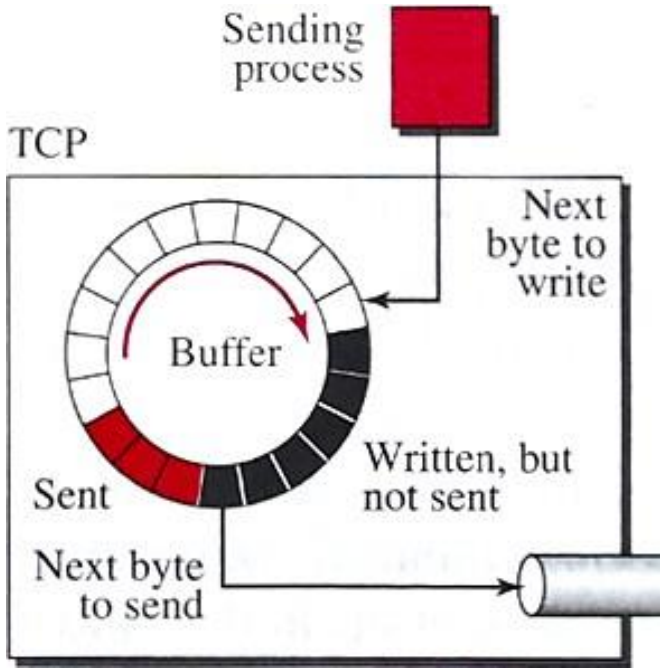
As we will see later, these buffers are also used for flow- and error-control.

These buffers are often implemented as circular buffers.
(shown here with 20 bytes, but in practice they will be much larger)



Transport Layer

Transmission Control Protocol (TCP): Sending and receiving buffers



On the sender side, the buffer is divided into three areas:

- The white area is empty and can be filled with data from the sending process.
- The red area contains data that has been sent but has not been acknowledged.
- The black area contains data that TCP has not yet sent.

The size of the black area does not always depend on the TCP on the sender side, it can also be controlled by the receiver or connection (or only part of this area is to be sent).

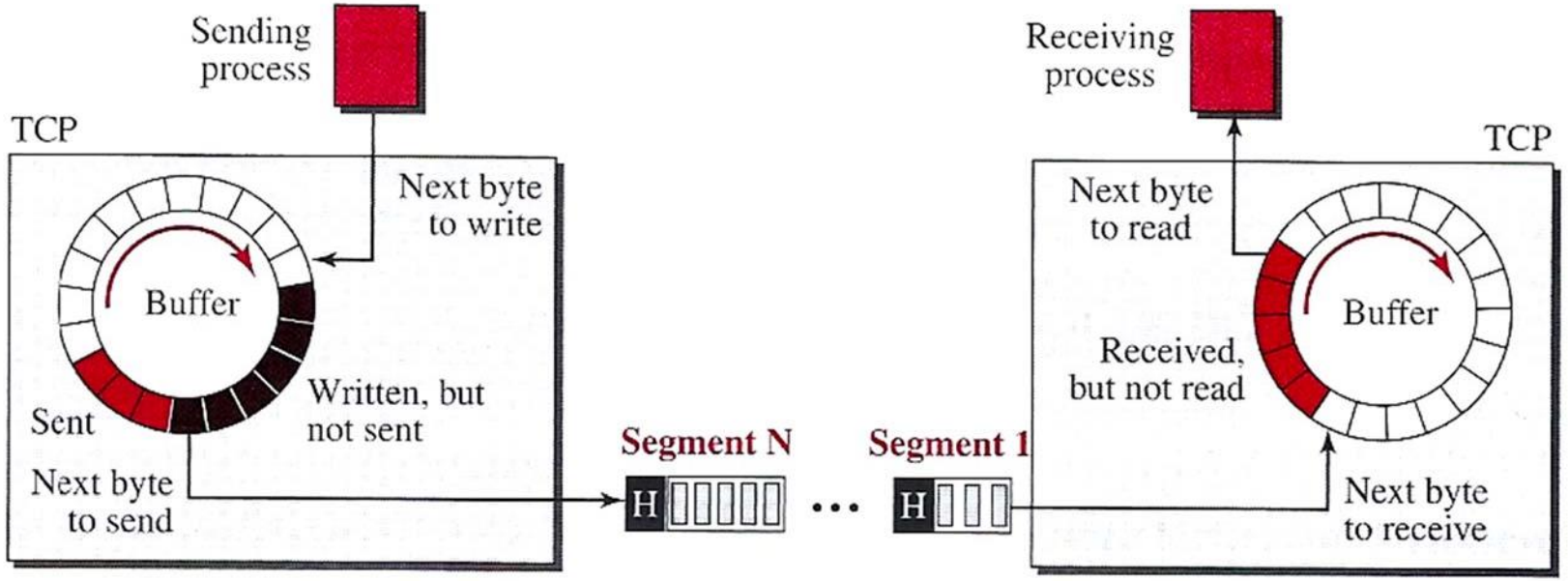
Transport Layer

Transmission Control Protokol (TCP): Segments

Although buffering can handle processing speed differences between producer and consumer, it takes one more step before we can send data.

The network layer, as a service provider for TCP, needs to send data in packets, not as a stream of bytes.

At the transport layer, TCP groups a number of bytes together into a packet called a **Segment**. TCP adds a header to each segment (for control purposes) and delivers the segment to the network layer for transmission.



Note that segments do not have to be the same size

Transport Layer

Transmission Control Protocol (TCP): Properties

Full-Duplex communication

TCP offers Full-Duplex service, which means that data can flow in both directions at the same time (e.g., Telephone).

Each process that uses TCP has both sending and receiving buffers.

Connection-oriented service

TCP offers a connection-oriented service. It behaves as follows in three phases:

1. Two processes with access to TCP establish a connection between them.
2. Data is exchanged in both directions simultaneously (Full-Duplex).
3. The connection is terminated.

Transport Layer

Transmission Control Protocol (TCP): Properties

Remember the connection established in the first phase is a logical connection, not a physical one.

The TCP segment is encapsulated in an IP datagram and can be sent out of order, or lost or corrupted, and then resent. Each may be routed over a different path to reach the destination.

Connection-oriented service means TCP creates a stream-oriented environment in which it accepts the responsibility of delivering the bytes in order to the other site.

Reliable service

TCP is a reliable transport protocol.

It uses an acknowledgement mechanism to check that data reaches its receiver safely without errors. (we'll look at that later)

TCP features

Transport Layer

Transmission Control Protocol (TCP): Features

In order to offer the service that we have just mentioned, TCP needs to have some features that we will now look at.

Numbering system

TCP keeps track of the segments sent and received, even though there is no field in the segment header where a segment number can be specified.

Instead, there are two other fields in the segment header:

- **Sequence number.**
- **Acknowledgment number.**

These fields refer to the byte number, not the segment number.

Transport Layer

Transmission Control Protocol (TCP): Features

Byte number

- TCP numbers all bytes that are transmitted via the connection.
- The numbering is independent of the direction.
- When TCP receives data (bytes) from a process, they are stored in the sending buffer and numbered.
- The numbering does not necessarily start from 0
- TCP generates a random number between **0** and $\mathbf{2^{32}-1}$ for the number of the first byte.
- We will see later that this byte numbering is used for flow- and error-control.

Transport Layer

Transmission Control Protocol (TCP): Features

Sequence number

After the bytes have been numbered, TCP assigns a sequence number to each segment that is being sent.
This sequence number is the same as the byte number of the first data byte of in the segment.
When a segment carries both data and control information (called **piggybacking**), the sequence number is used as a reference.

If a segment does not contain a data part, then it does not logically define a sequence number as it corresponds to the first data byte number in the sequence.
(the field is in the header, but the value is not valid!)

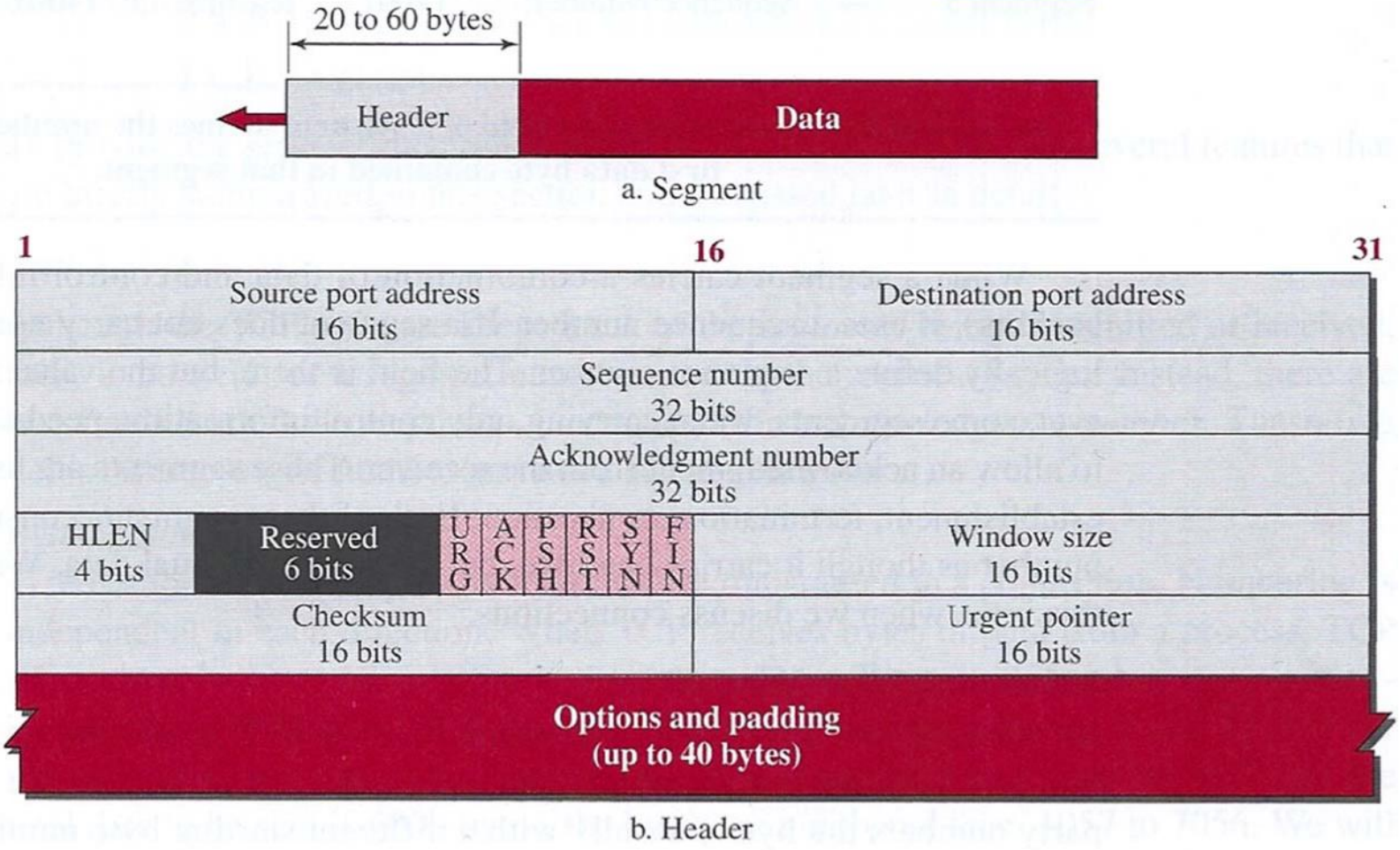
However, some segments, when carrying only control information, need a sequence number to allow an acknowledgment from the receiver. These segments are used for connection establishment, termination, or abortion. Each of these segments consume one sequence number **as though it carries one byte, but there are no actual data**.

TCP segments

Transport Layer

Transmission Control Protocol (TCP): Segment format

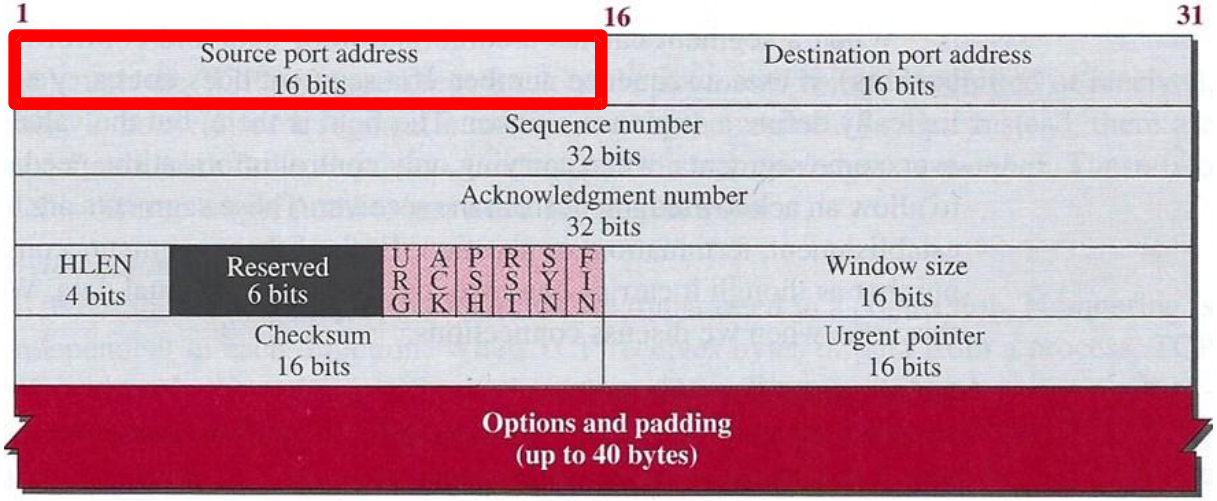
Let us now take a closer look at the TCP packet format which is called a **segment.**



The segment consists of a header of 20 to 60 bytes, followed by data from an application program. The header is 20 bytes if no options are specified.

Transport Layer

Transmission Control Protocol (TCP): Segment format

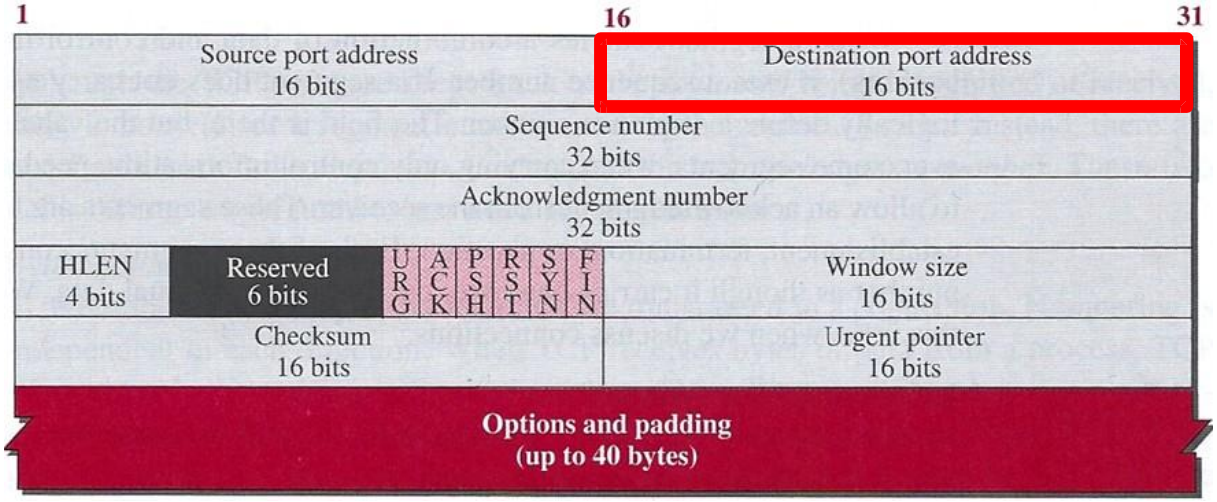


Source (sender) port address: This 16-bit field specifies the port number of the application program on the host that sent the segment.

This field has the same function as the UDP header's Source port address.

Transport Layer

Transmission Control Protocol (TCP): Segment format

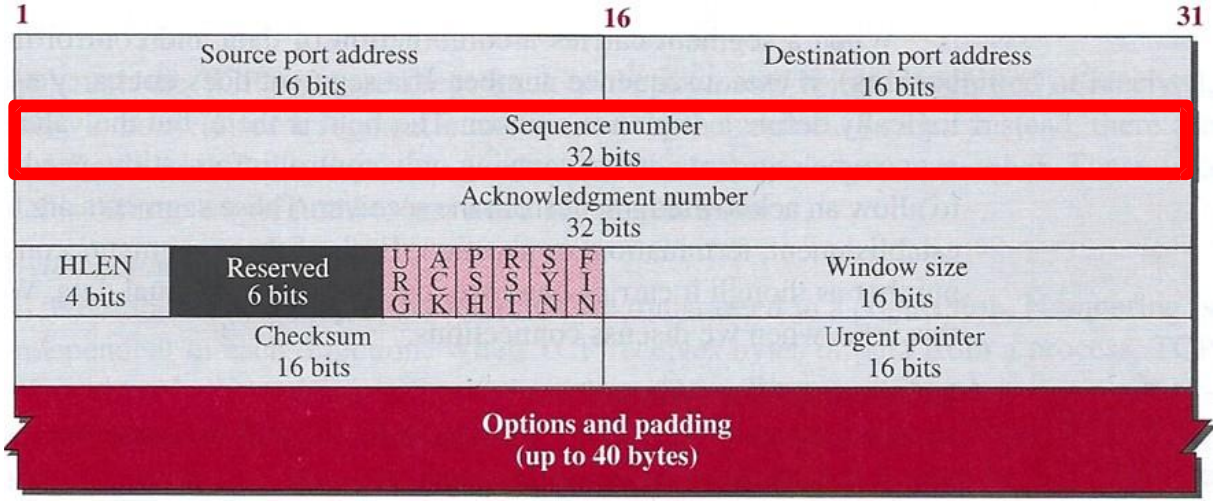


Destination (receiver) port address: This 16-bit field indicates the port number of the application program on the host that received the segment.

This field has the same function as the UDP header's Destination port address.

Transport Layer

Transmission Control Protocol (TCP): Segment format

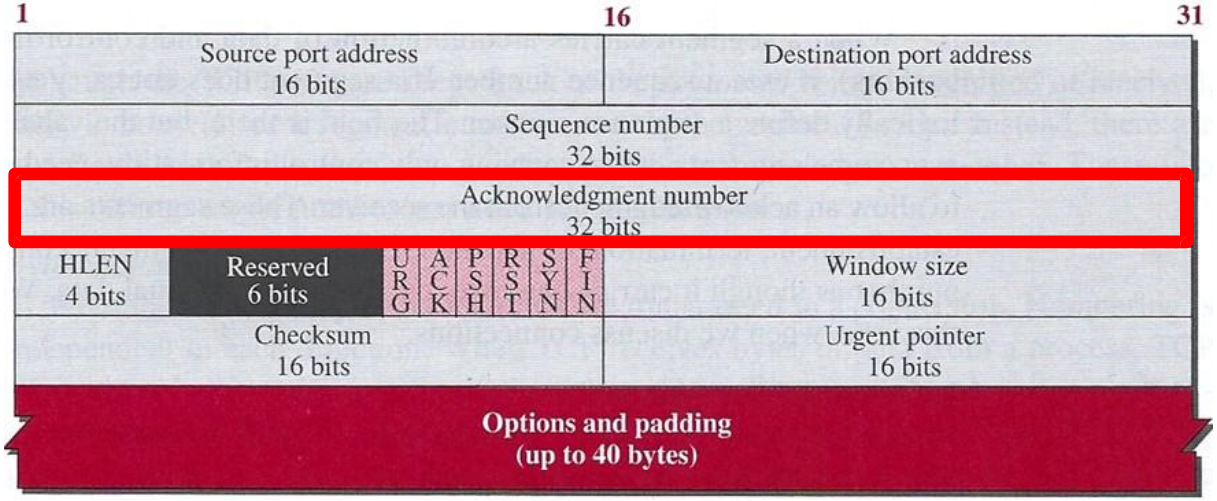


Sequence number: This 32-bit field indicates the number that was assigned the first data byte in the segment.

Remember that TCP numbers all the bytes, so the next segment has a sequence number, which is the ***current sequence number + number of bytes in the current segment***

Transport Layer

Transmission Control Protocol (TCP): Segment format

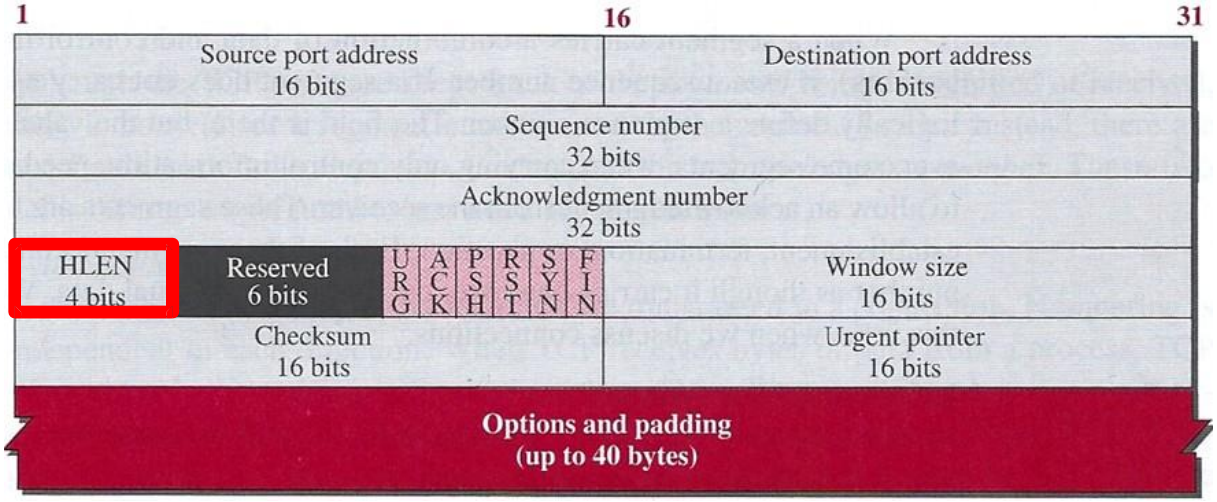


Acknowledgment number: This 32-bit field indicates the sequence number of the segment that the receiver expects to receive from the sender.

if the receiver itself has data to be sent back to the sender, then the acknowledgement can be sent back to the sender together with the data. This is called ***piggybacking***.

Transport Layer

Transmission Control Protocol (TCP): Segment format



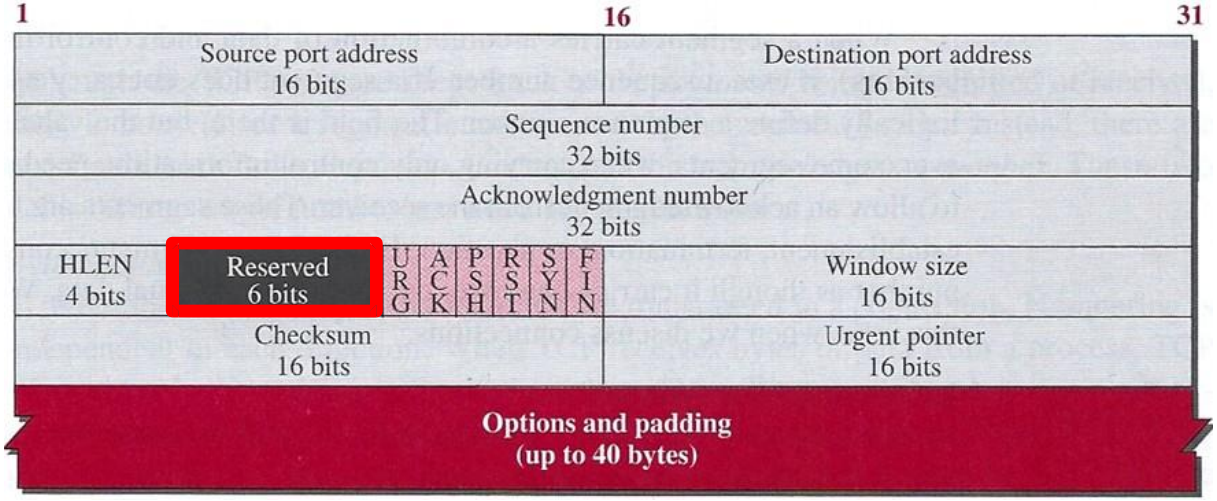
Header length (HLEN): This 4-bit field indicates the header size in terms of the number of 4-byte words in the TCP header.

The header can be between *20* and *60* byte.
Therefore, the value in this field is between ***5*** and ***15***.

(5 = 5 x 4 = 20 and 15 = 15 x 4 = 60)

Transport Layer

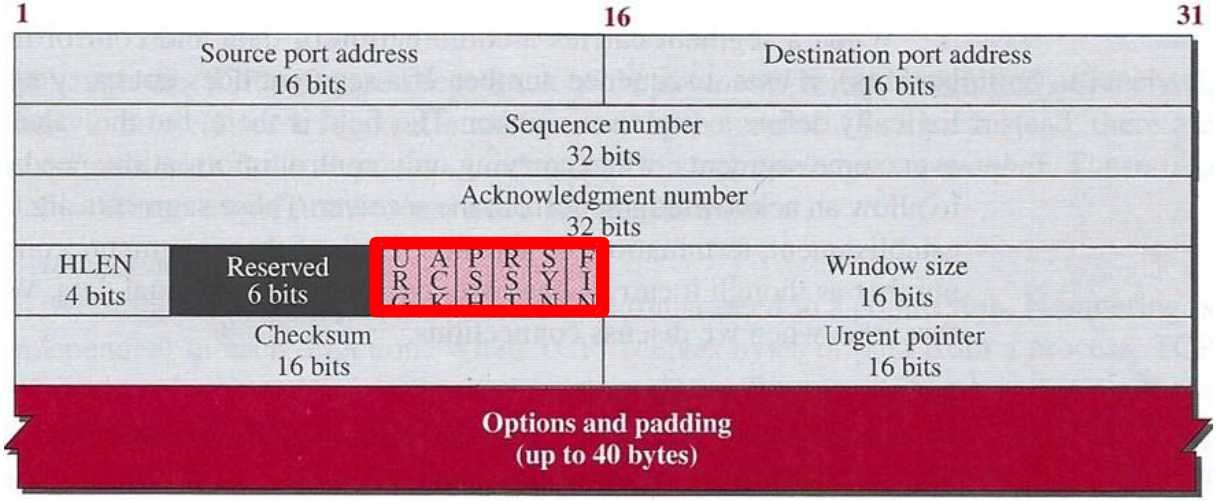
Transmission Control Protocol (TCP): Segment format



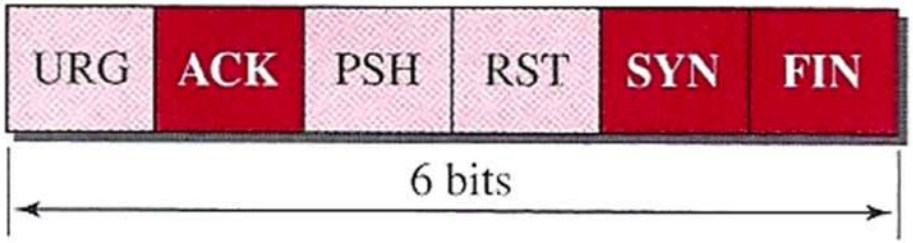
Reserved: This 6-bit field is reserved for future use.

Transport Layer

Transmission Control Protocol (TCP): Segment format



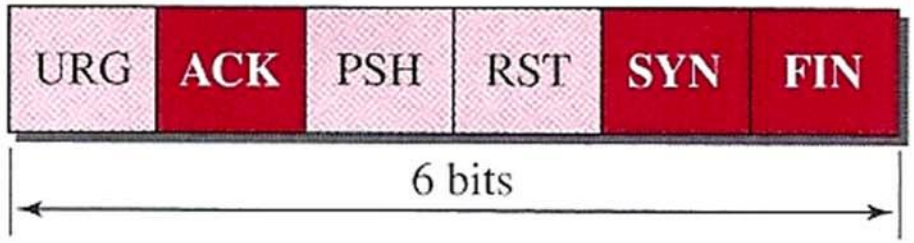
Control: This 6-bit field indicates 6 different flags (one or more can be activated at a time)



URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH : Request for push
RST : Reset the connection
SYN: Synchronize sequence numbers
FIN : Terminate the connection

Transport Layer

Transmission Control Protocol (TCP): Segment format



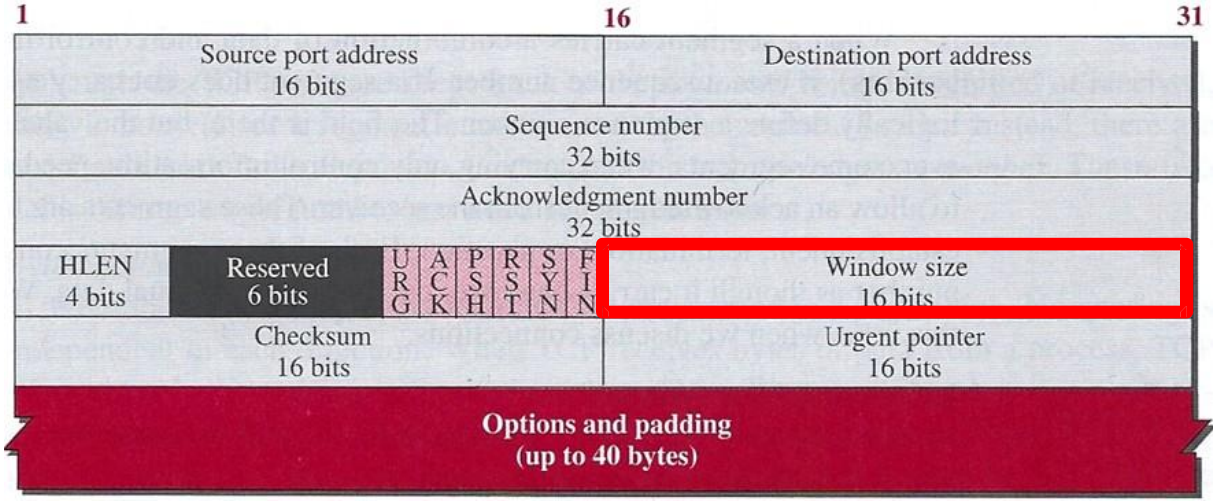
URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH : Request for push
RST : Reset the connection
SYN: Synchronize sequence numbers
FIN : Terminate the connection

These flags enable flow control, connection establishment and termination, connection abortion and the mode of data transfer in TCP.

Flag	Description
URG	The value of the Urgent Pointer field is valid
ACK	The value of the acknowledgment field is valid
PSH	Push data
RST	Reset the connection
SYN	Synchronize sequence numbers in the connection phase
FIN	Disconnect

Transport Layer

Transmission Control Protocol (TCP): Segment format

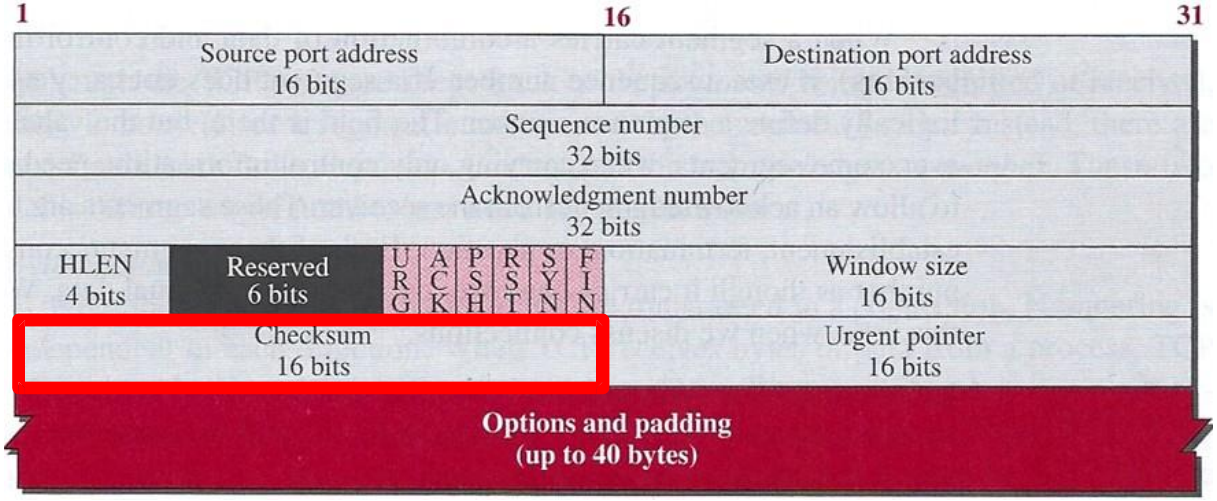


Window size: This 16-bit field indicates the window size of the sending TCP in bytes.

The value in the field is often referred to as the ”receiving window” (**RWND**) and is determined by the receiver. The sender must obey the dictation of the receiver in this case.

Transport Layer

Transmission Control Protocol (TCP): Segment format



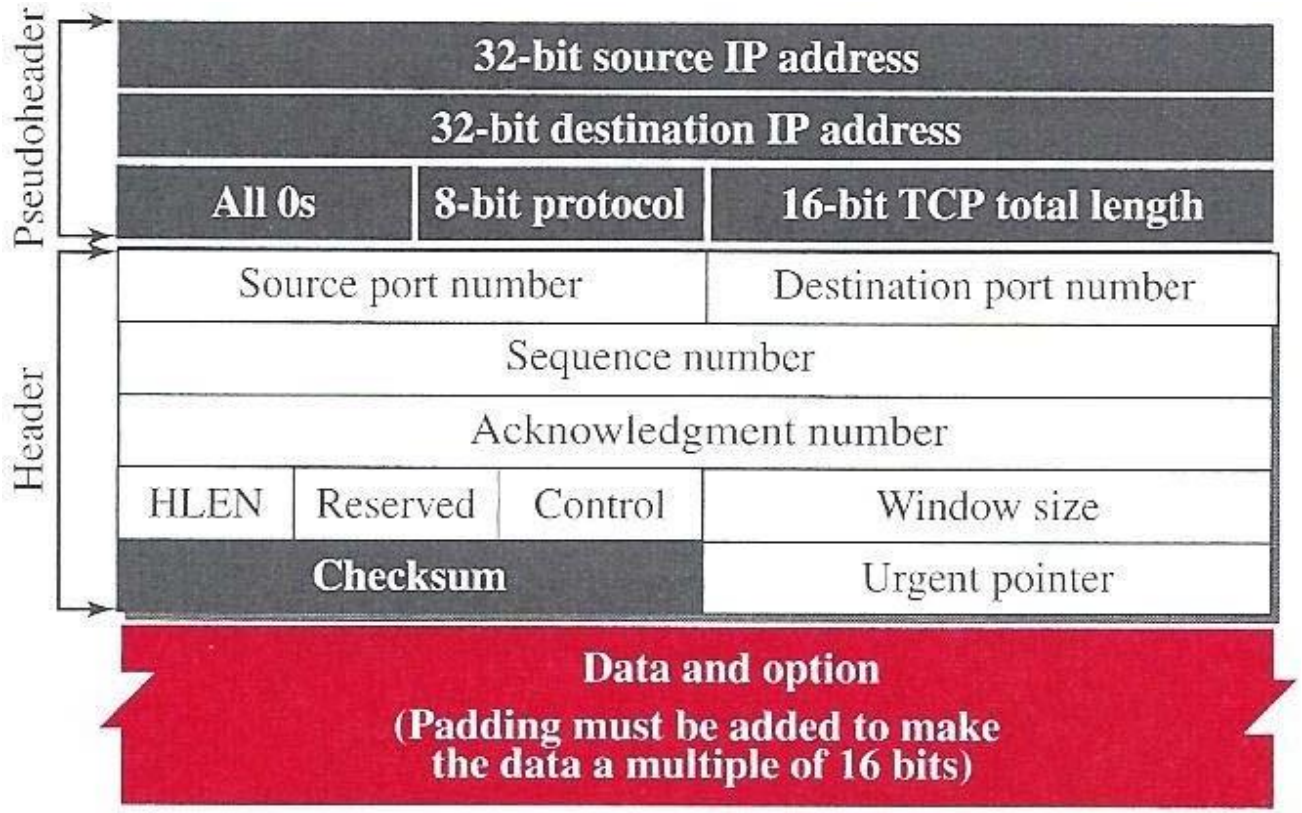
Checksum: This 16-bit field contains the checksum.
The calculation of this follows the same procedure as for UDP.

However, the checksum was optional to include in UDP datagrams, whereas it is mandatory to include it in TCP segments.

The pseudoheader serves the same purpose in this protocol as in UDP.
Note, however, that the protocol field has the value of ***6***.

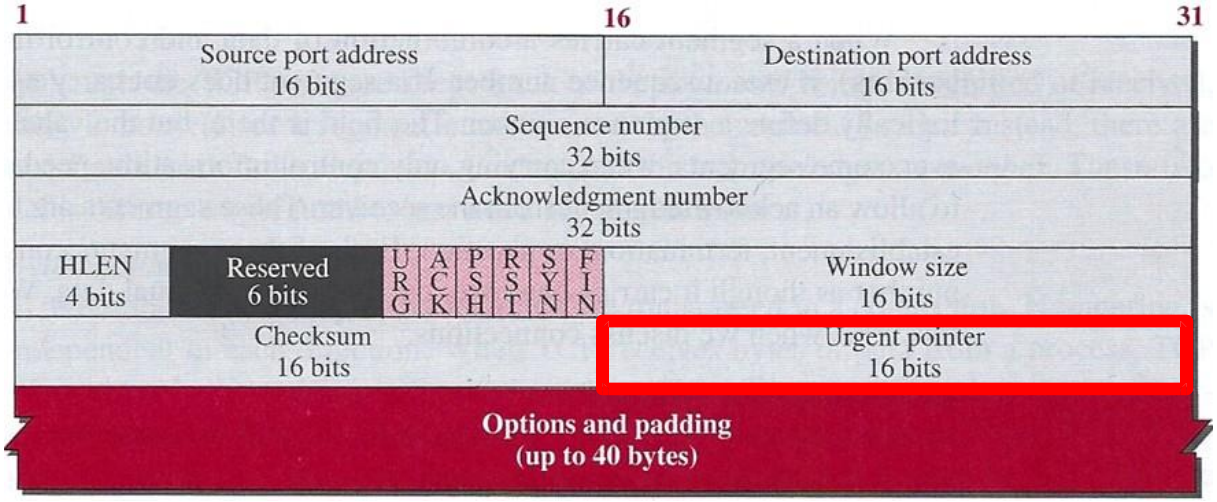
Transport Layer

Transmission Control Protocol (TCP): Segment format



Transport Layer

Transmission Control Protocol (TCP): Segment format

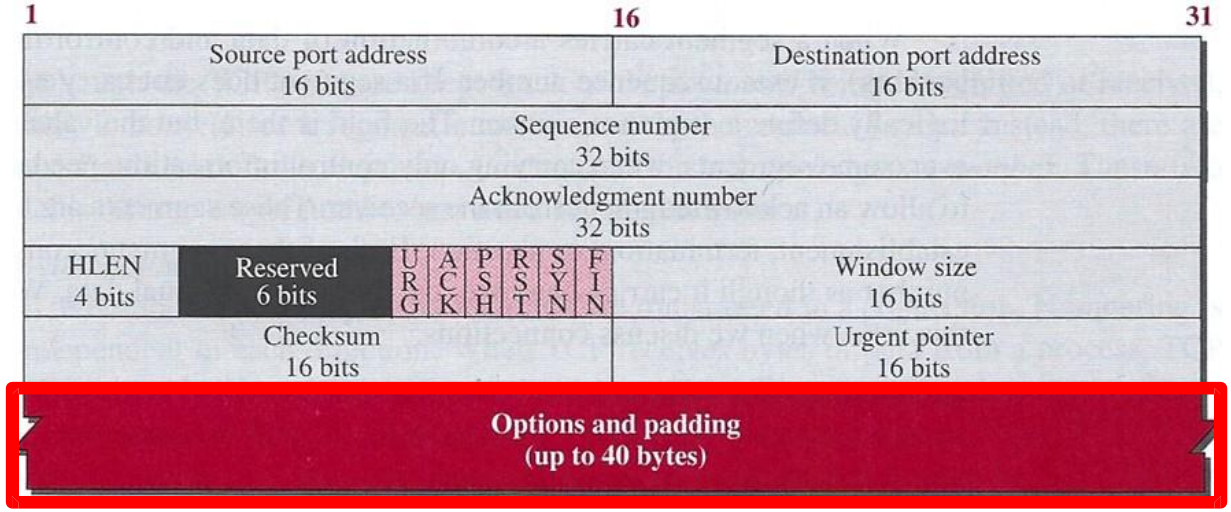


Urgent pointer: This 16-bit field, whose value is only valid when the URG flag is set, is used when segments contain data of an urgent nature!

The value in the field must be added to the sequence number to find the number of the last urgent byte in the data section of the sequence. **(we'll take a closer look at this later).**

Transport Layer

Transmission Control Protocol (TCP): Segment format



Options: There can be up to 40 bytes of optional information in a TCP header.

We will not go into that in detail here.

Encapsulation

- A TCP segment encapsulates the data received from the application layer.
- The TCP segment is encapsulated in an IP datagram at the network layer.
- The IP datagram is encapsulated in a frame at the data link layer.

TCP connection

Transport Layer

Transmission Control Protocol (TCP): TCP connection

TCP is connection-oriented, the connection is logical (not physical). TCP operates at a higher level. TCP uses the services of IP to deliver individual segments to the receiver, but it controls the connection itself.

IP thus has no knowledge of this connection; it simply provides service to TCP which manages the connection. For instance, If a segment is lost or corrupted, it is retransmitted. Unlike TCP, IP is unaware of this retransmission. If a segment arrives out of order, TCP holds it until the missing segments arrive; IP is unaware of this reordering.

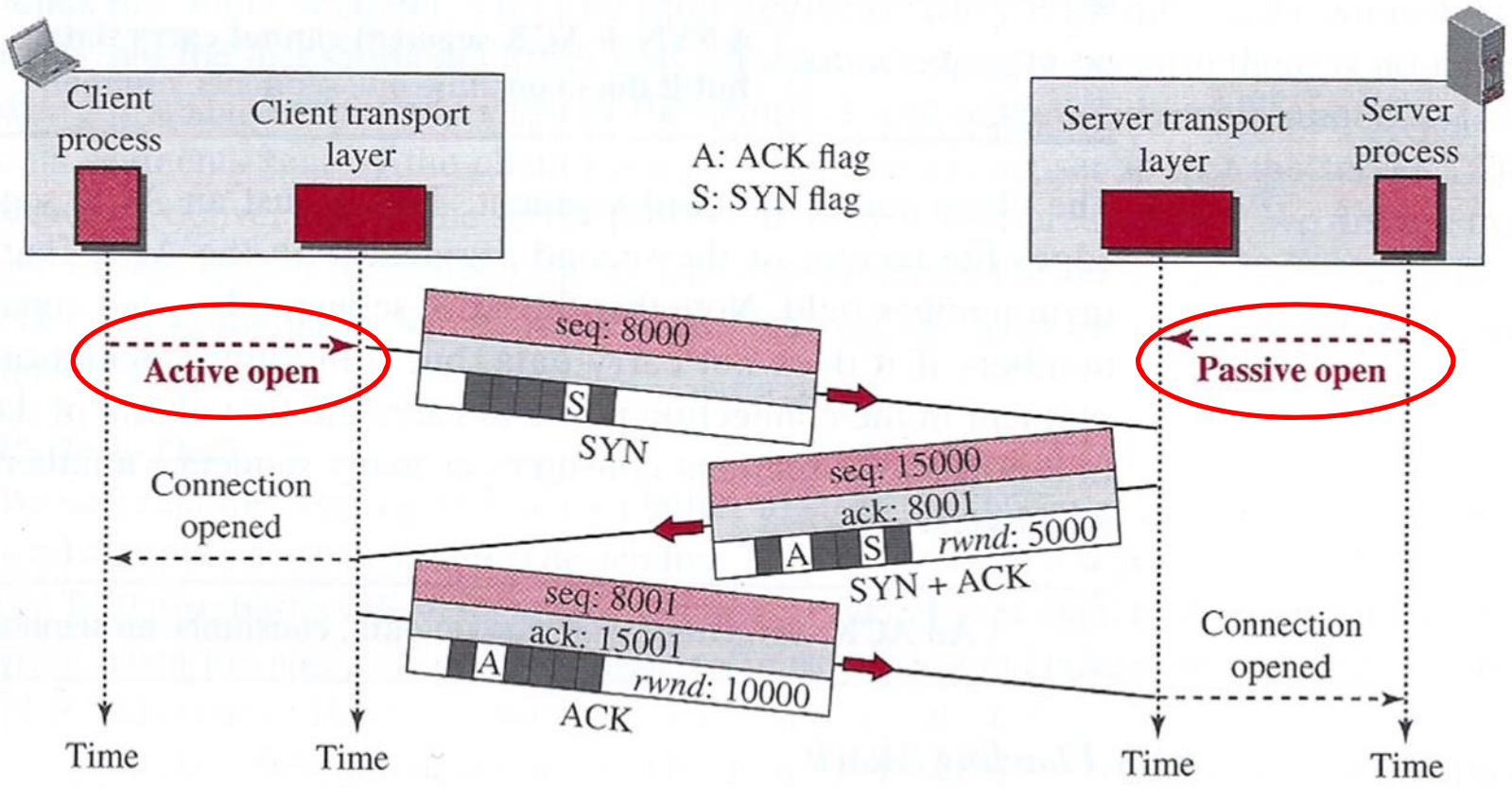
In TCP, connection-oriented transmission requires three phases:

1. **connection establishment**
2. **data transfer**
3. **and connection termination**

Transport Layer

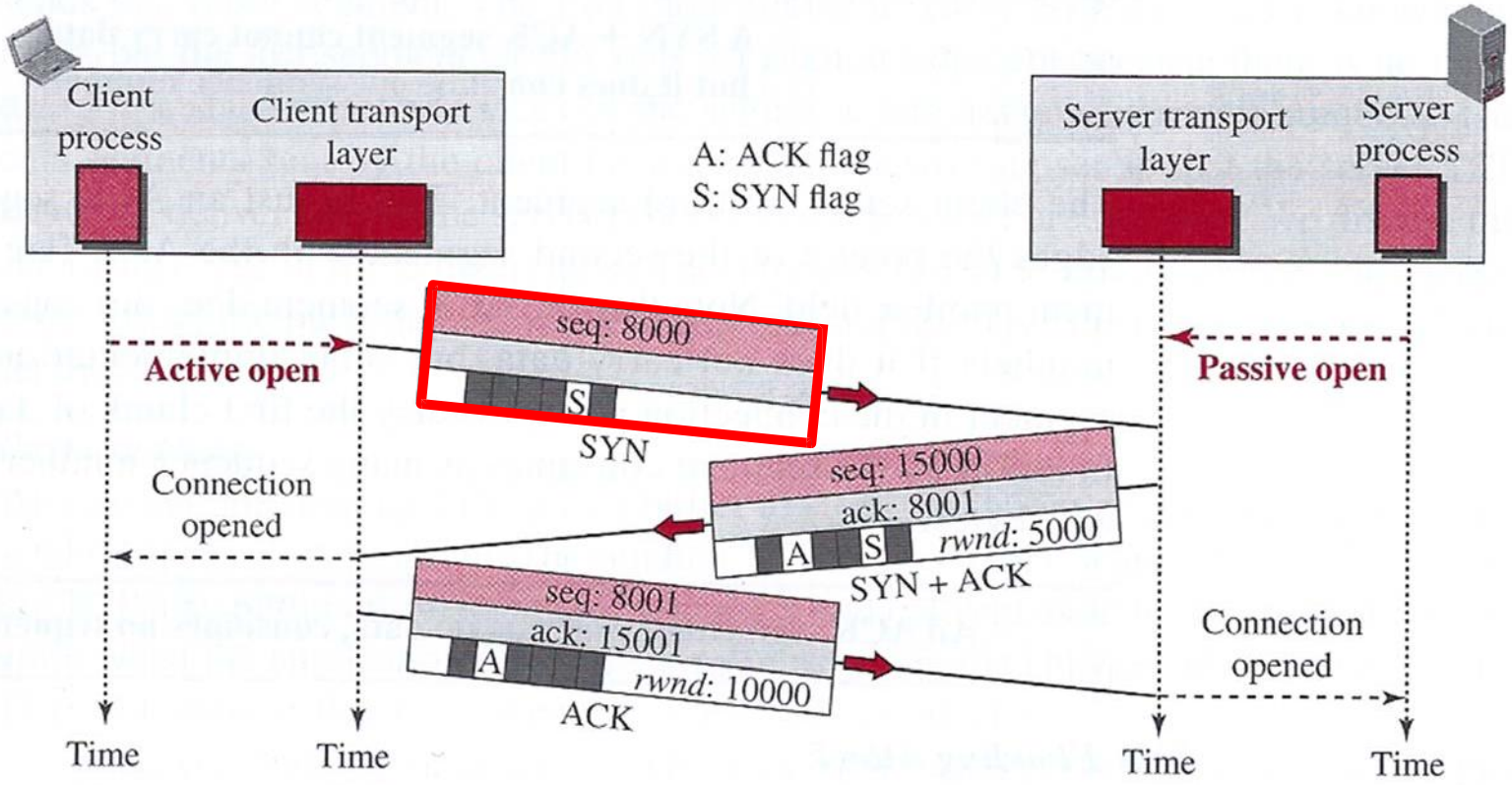
Transmission Control Protocol (TCP): 3-way handshaking

The server application notifies its TCP that it is ready to receive a connection (from any host in the world).
This is called ***passive open***.
The client process sends a request to connect to the server.
This is called ***active open***.



Transport Layer

Transmission Control Protocol (TCP): 3-way handshaking

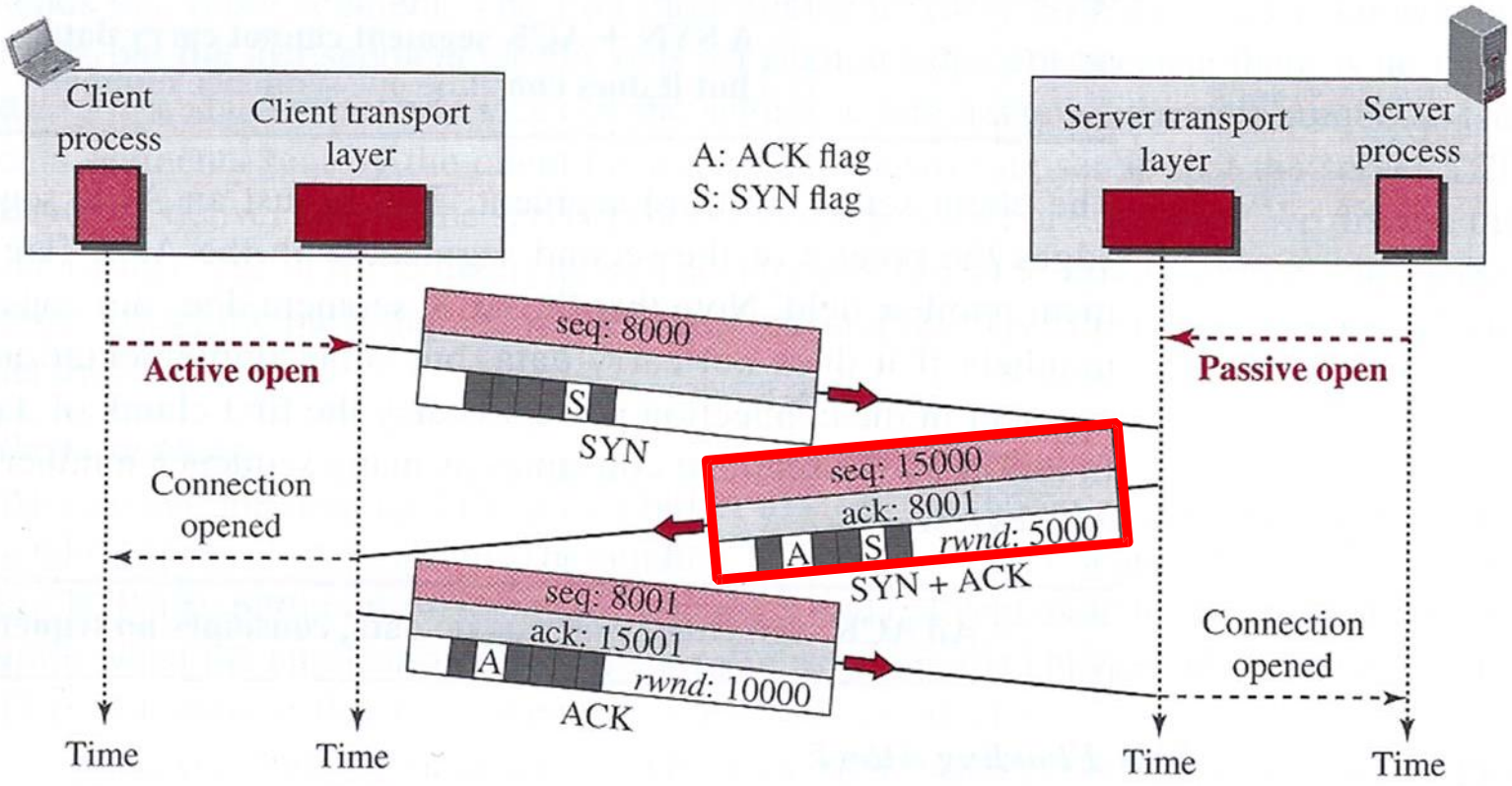


The client sends the first segment, a so-called ***SYN segment***, where only the SYN flag is set. This is done to give the client a starting point for the following sequence numbers.

Note that the SYN segment is a control segment and carries no data. However, it consumes one sequence number because it needs to be acknowledged. We can say that the SYN segment carries one imaginary byte.

Transport Layer

Transmission Control Protocol (TCP): 3-way handshaking

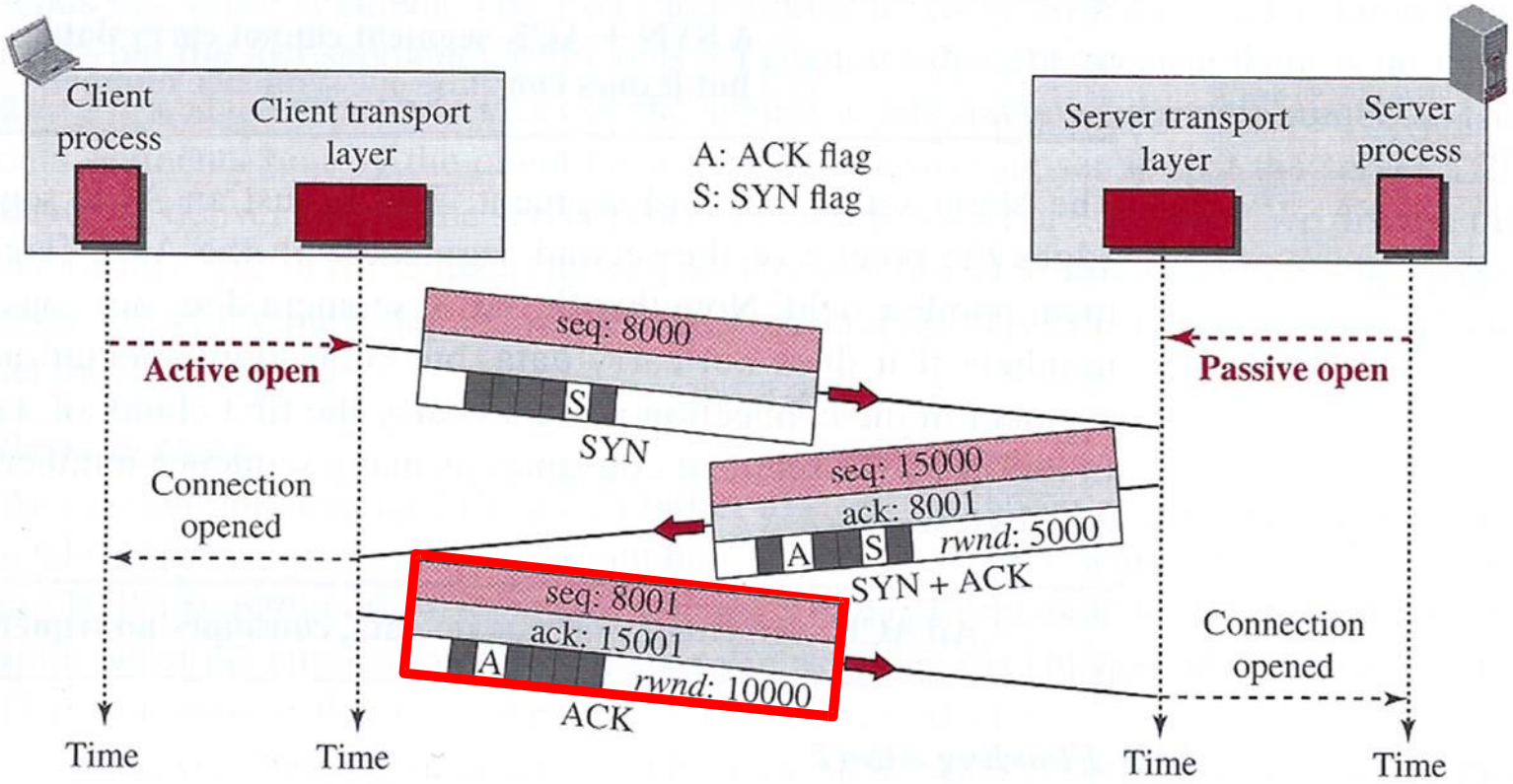


The server sends, a so-called ***SYN + ACK segment***, where the SYN and ACK flags are set. SYN is set to give the server a starting point for the following sequence numbers. ACK is a receipt for the previously received segment.

The receive window size at the server side is defined as 5000.

Transport Layer

Transmission Control Protocol (TCP): 3-way handshaking



The client sends, an ***ACK segment***, where the ACK flag is set.
ACK is a receipt for the previously received segment.

Note that the ACK segment does not contain data, therefore consumes no sequence number.

The receive window size at the client side is defined as 10000.

Transport Layer

Transmission Control Protocol (TCP): SYN flooding attack

The procedure used to establish connection in TCP is vulnerable to a serious security issue called **SYN flooding attack**.

This happens when one or more malicious attackers send a large number of ***SYN segments*** to a server pretending that each of them is coming from a different client by faking the source IP addresses in the datagrams.

The server assumes that the clients are performing the ***active open***, and therefore allocates resources (such as buffers, etc.) to them.

The server sends ***SYN + ACK segments*** to the fake IP addresses, these segments are of course lost.

But the server rapidly runs out of its resources and is likely to crash! The attack is also called ***denial-of-service attack***.

Transport Layer

Transmission Control Protocol (TCP): SYN flooding attack

Strategies exist to prevent such attacks:

- There may be a limit to how many SYN requests you will accept within a certain period of time.
- One can filter away datagrams that come from unwanted IP addresses.
- One can postpone the resource allocation until the server can verify that the connection request is coming from a valid IP address.
  This can be done with so-called ***cookies***.
  SCTP uses this method.

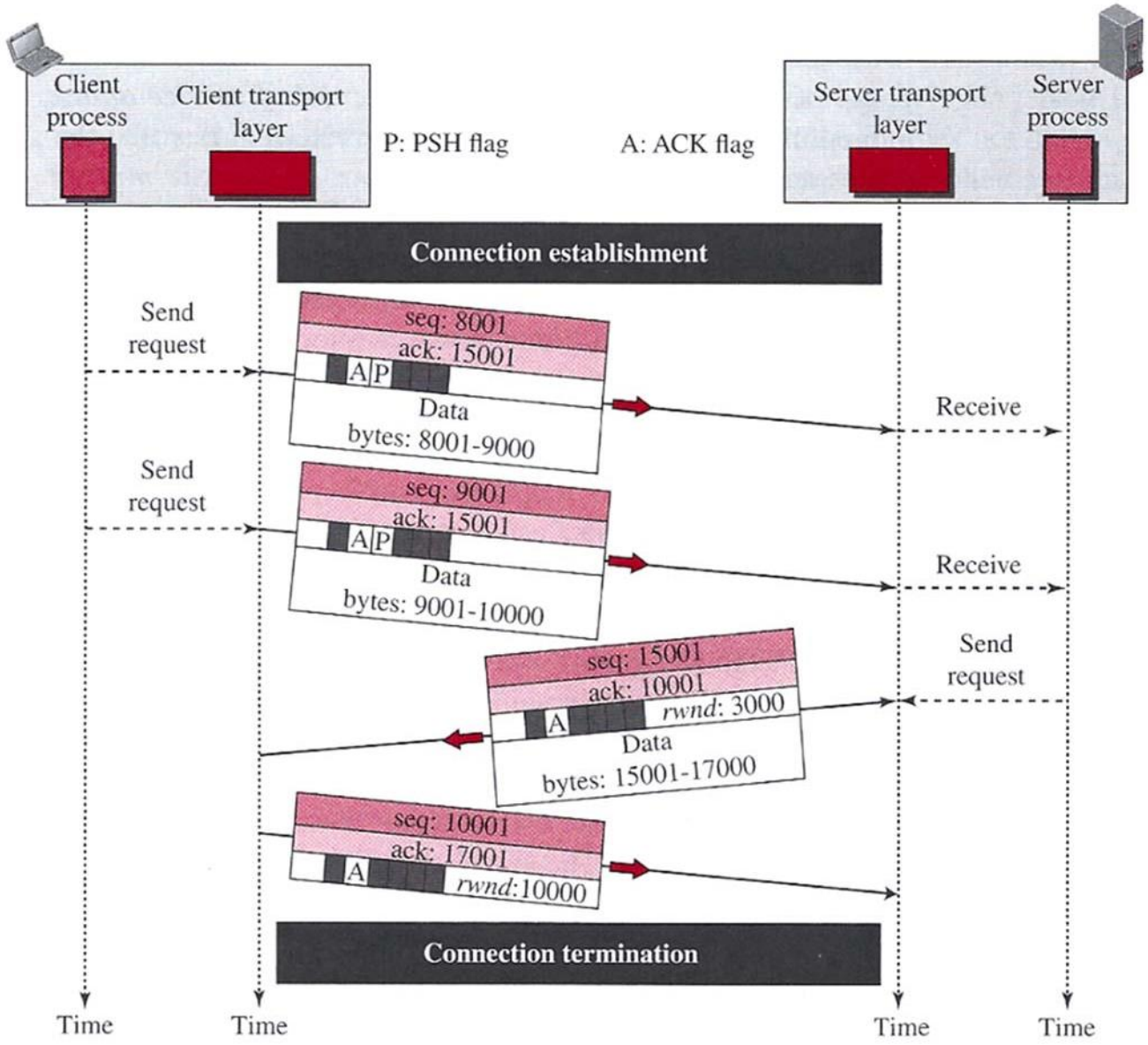
Transport Layer

Transmission Control Protocol (TCP): Data transfer

The client sends 2000 bytes of data in two segments

The server sends 2000 in one segment

The client set the PSH (Push) flag, indicating that data should be delivered to the server process as soon as TCP on the server side has received it.



Transport Layer

Transmission Control Protocol (TCP): Data transfer

Pushing Data

If a sender wants to get the data sent off quickly, then it does not wait for a window of agreed size to be completely filled with data.

- Data is sent immediately in segments!
- And the PSH flag is set.

This indicates to the receiver that this data should be processed and delivered to the application layer as soon as it arrives. So, you do not have to wait for more data (which could fill a receiving window).

NOTE

Although the PSH flag can be enabled, most TCP implementations ignore such requests nowadays.

Transport Layer

Transmission Control Protocol (TCP): Data transfer

Urgent Data

TCP is a steam-oriented protocol. That is, data is presented as a data stream to the applications.

Each byte in the data stream has its own number.

But in special cases, applications need to send urgent bytes. some bytes that need to be treated in a special way by the application at the other end. The solution is to send a segment with the URG bit set.

- TCP on the sender side creates a segment where the block of urgent data is inserted first in the data section.
- TCP sets the URG flag
- and inserts the offset, in the urgent pointer field, which indicates the last urgent byte in the data section.

Transport Layer

Transmission Control Protocol (TCP): Connection termination

Either of the two parties involved in the data exchange (client or server) can close a connection.

It is usually the client which takes this initiative.

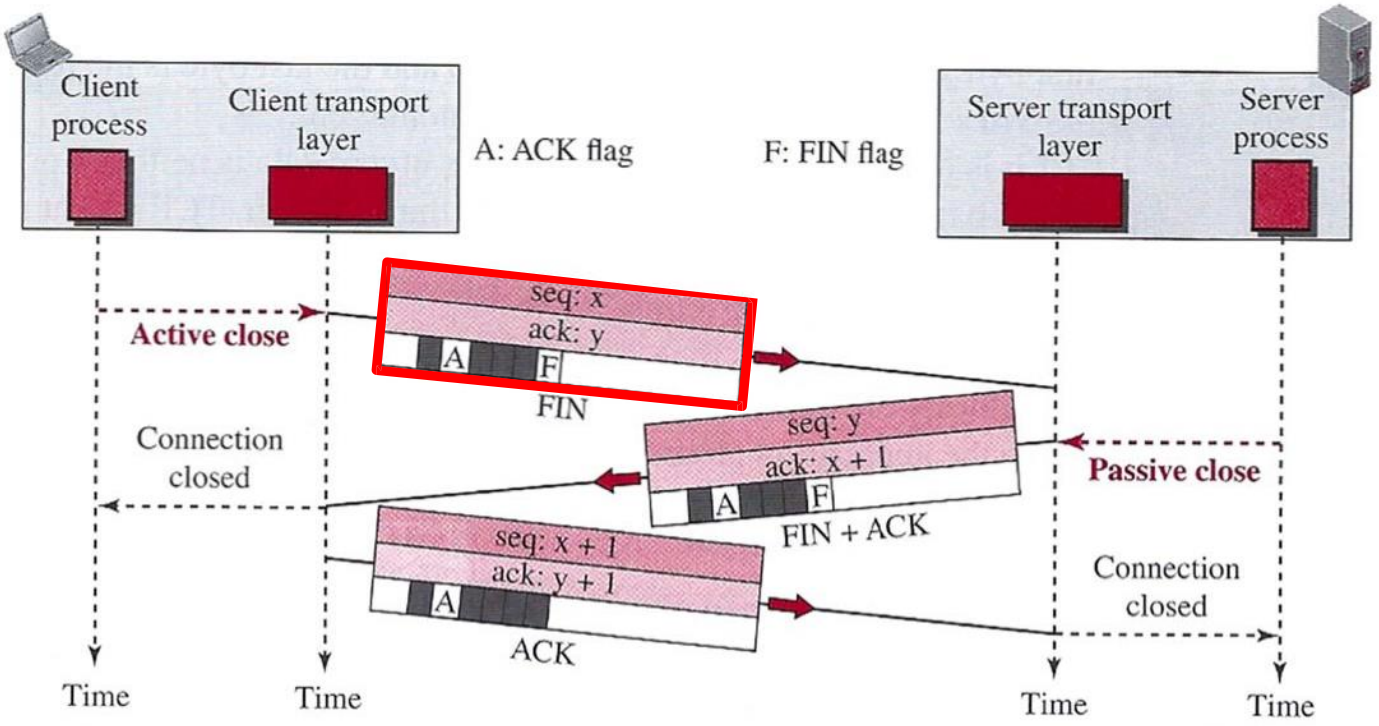
Most implementations today allow two procedures for connection termination:

- 3-way handshaking
- 4-way handshaking with half-close option.

Transport Layer

Transmission Control Protocol (TCP): Connection termination

3-way handshaking



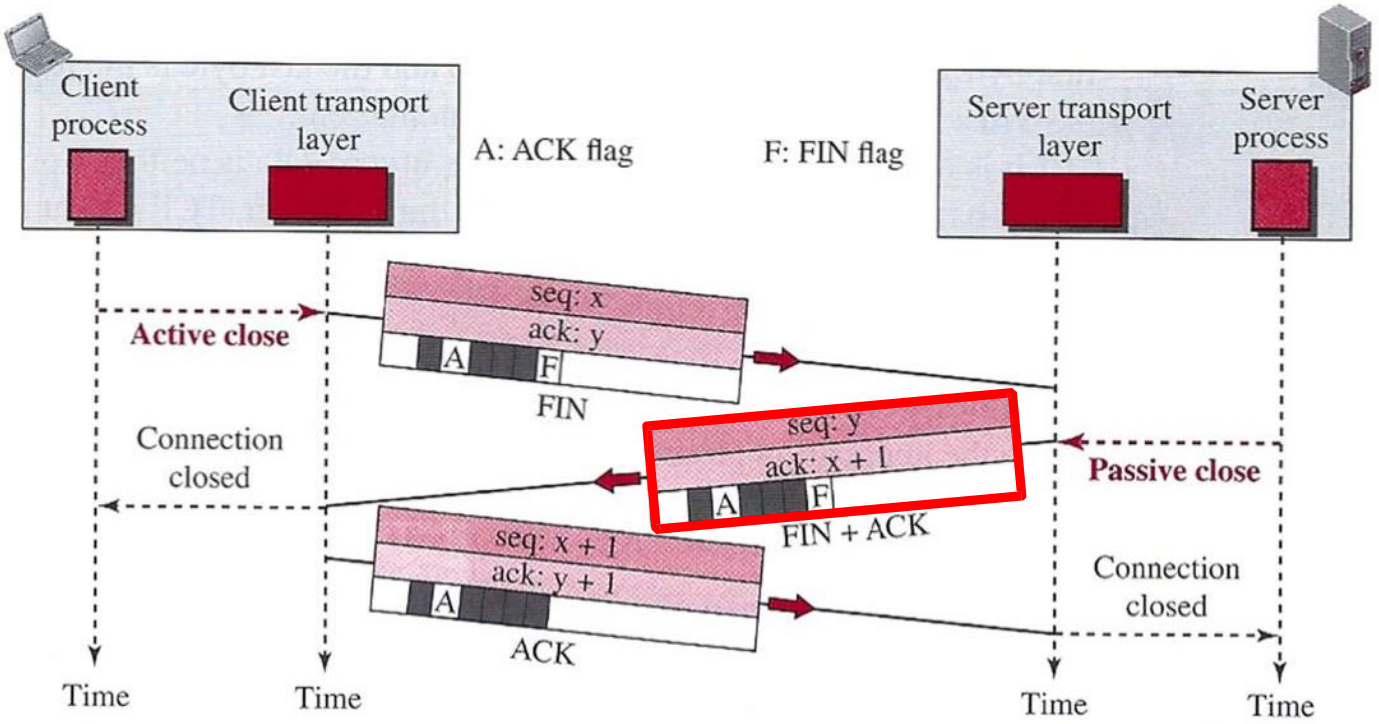
The client sends the first segment, so-called ***FIN segment***, where the FIN flag is set. Note that the FIN segment can include the last chunk of data sent by the client or it can also just be a control segment as shown in the figure.

The segment consumes a sequence number, as the segment must be acknowledged.

Transport Layer

Transmission Control Protocol (TCP): Connection termination

3-way handshaking



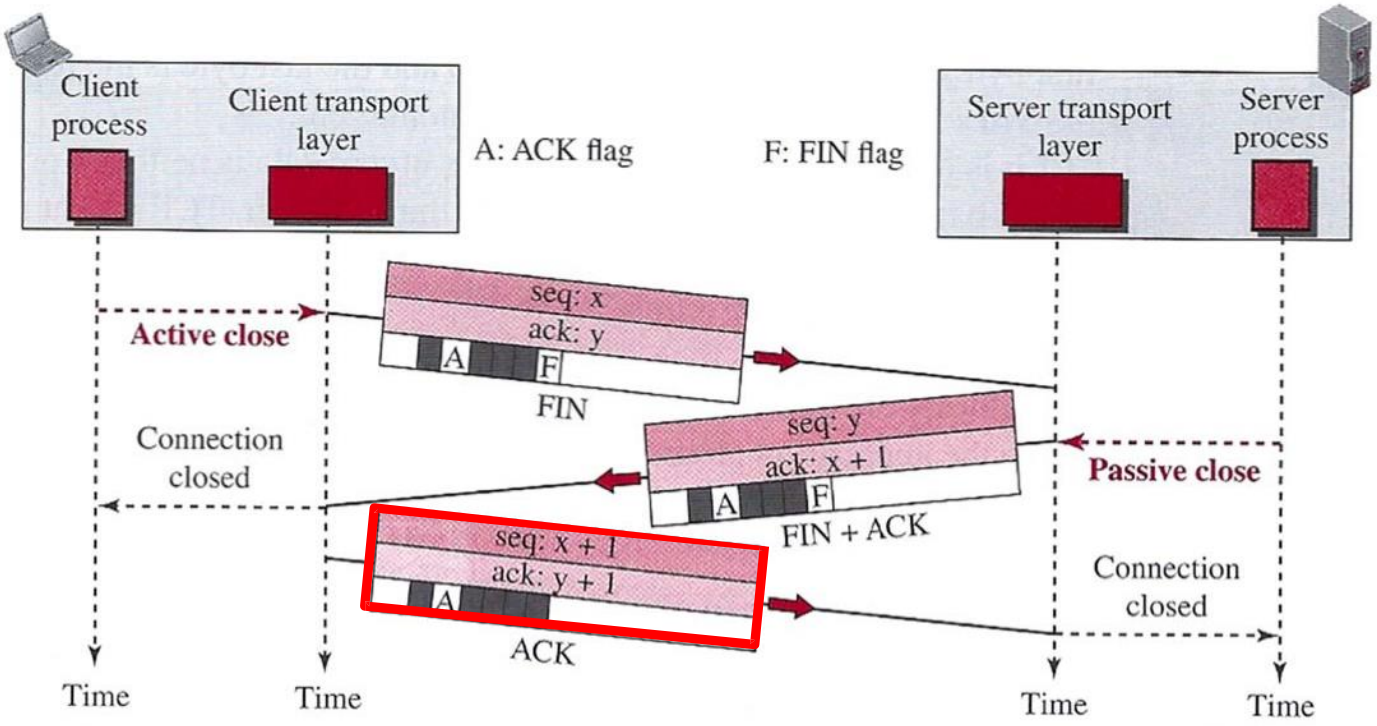
The server informs its associated process of the situation and sends a ***FIN + ACK segment*** where the FIN and ACK flags are set. Also note here that this segment can also contain the last chunk of data from the server, or it can also just be a control segment as shown in the figure.

The segment takes a sequence number, as the segment must be acknowledged.

Transport Layer

Transmission Control Protocol (TCP): Connection termination

3-way handshaking



The client sends the last segment, an ***ACK segment*** where the ACK flag is set.

This segment is an acknowledgment of the receipt of the segment from the server.

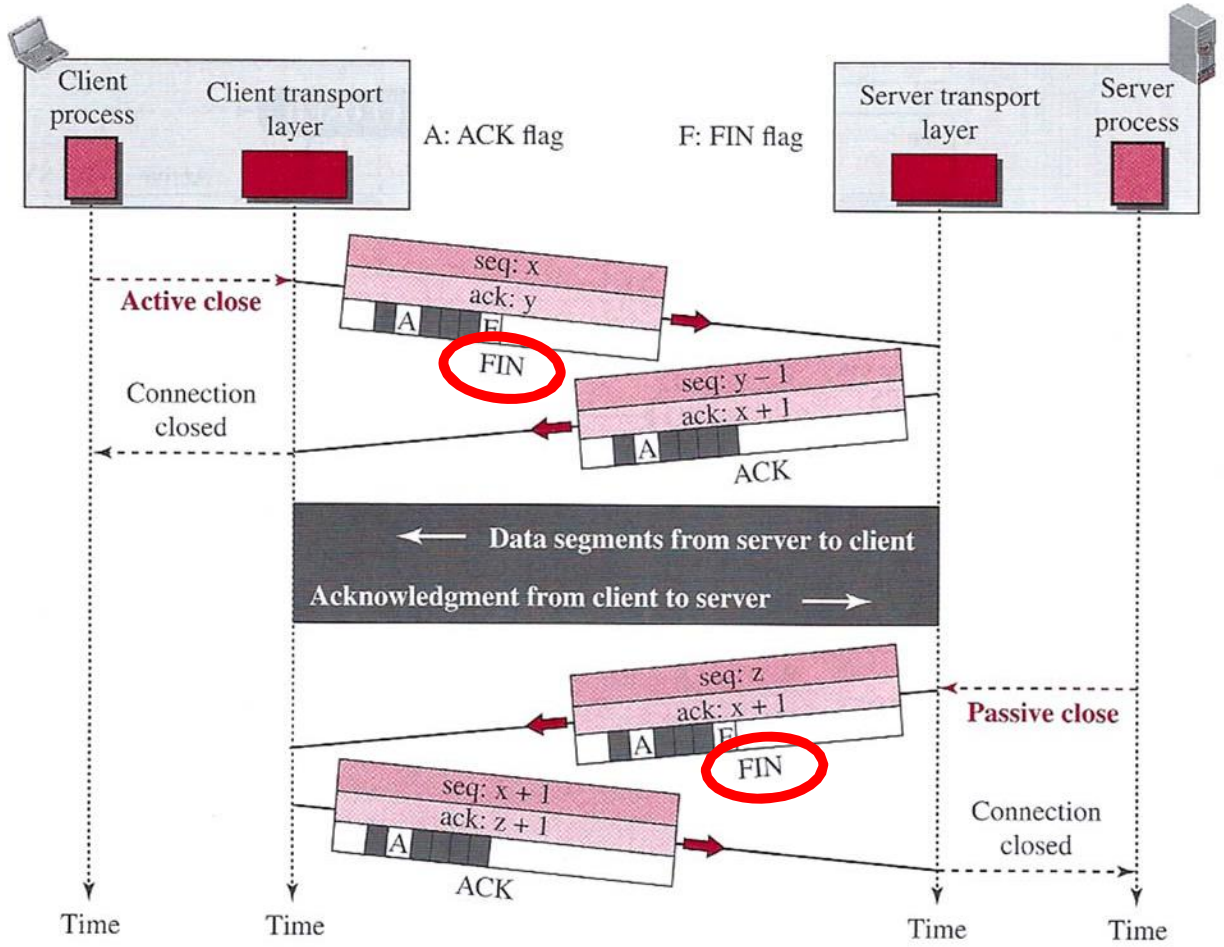
Note that the segment cannot carry data and therefore does not consume a sequence number.

Transport Layer

Transmission Control Protocol (TCP): Connection termination

Half-close

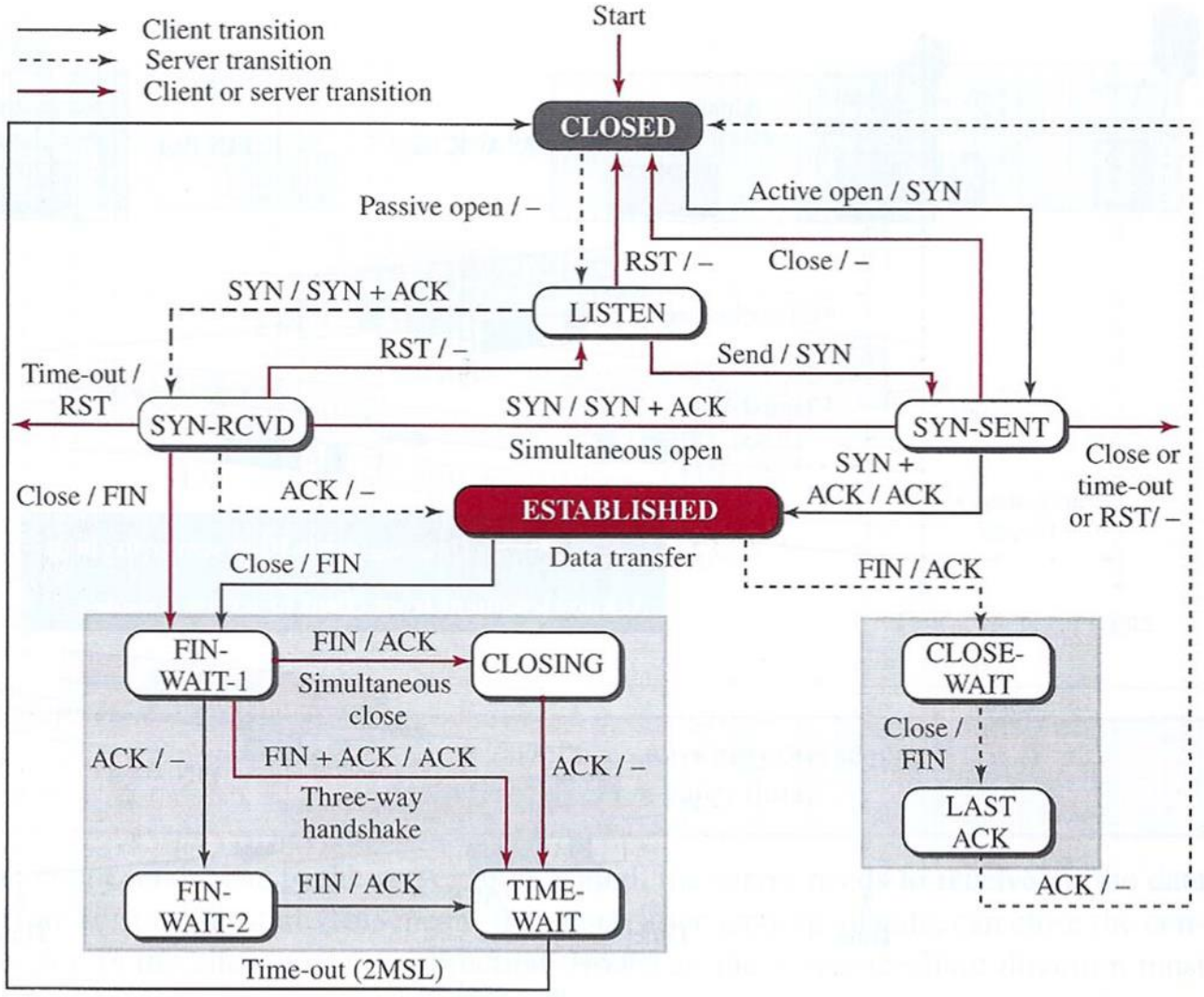
In TCP, one end can stop sending data while still receiving data. This is called a **half-close**. Either the server or the client can issue a half-close request. It can occur when the server needs all the data before processing can begin, e.g., sorting.



TCP state transition diagram

Transport Layer

Transmission Control Protocol (TCP): State Machine

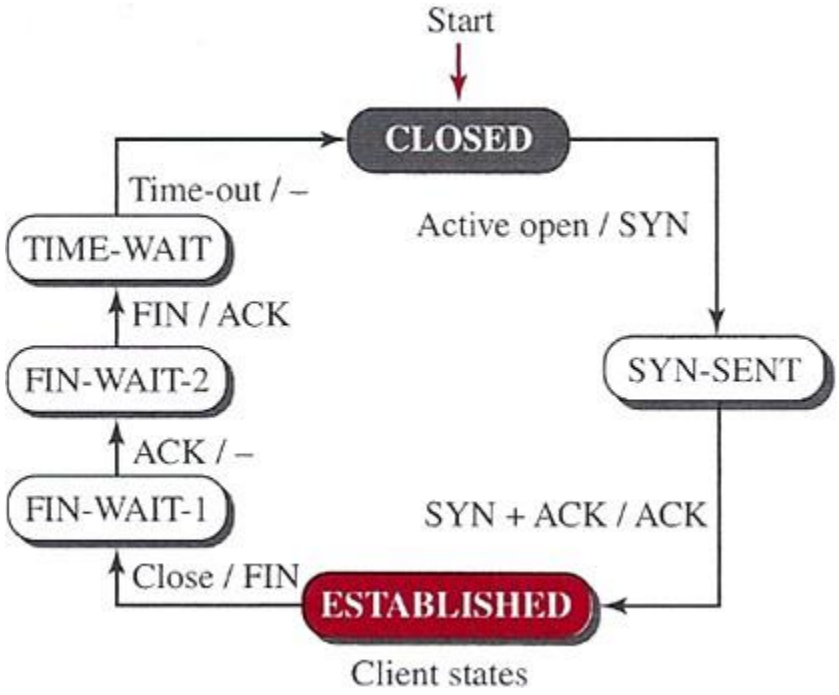


Here both sender and receiver are mixed together!

no comments

Transport Layer

Transmission Control Protocol (TCP): A Half-close scenario



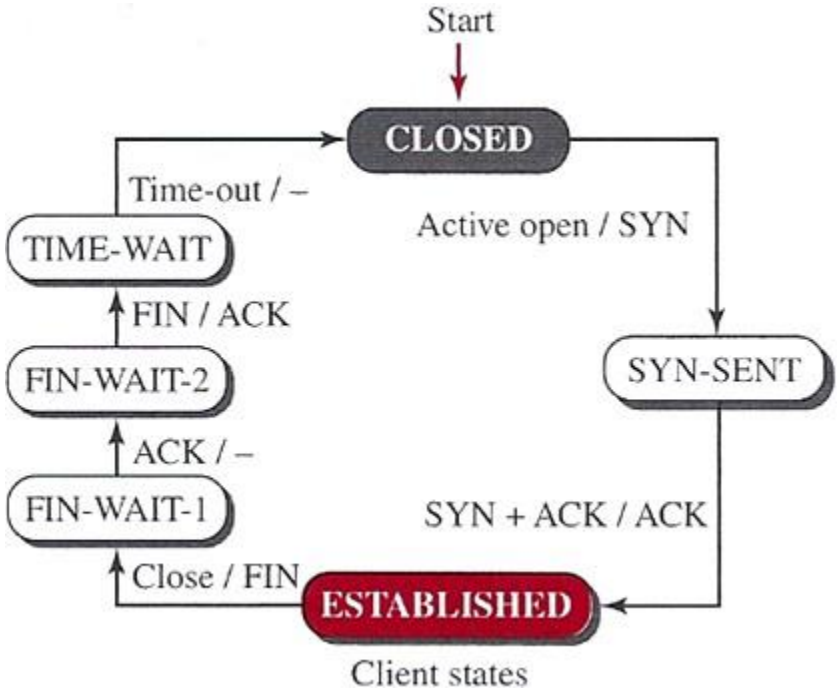
Client side: Establishment

- The client process issues an ***active open*** command to TCP to request a connection to a specific socket address.
- TCP sends a **SYN segment** and moves to SYN-SEND state.
- When a **SYN + ACK segment** is received, an **ACK segment** is sent and moved to the ESTABLISHED state.

Data are transferred, presumably in both directions, and acknowledged.

Transport Layer

Transmission Control Protocol (TCP): A Half-close scenario

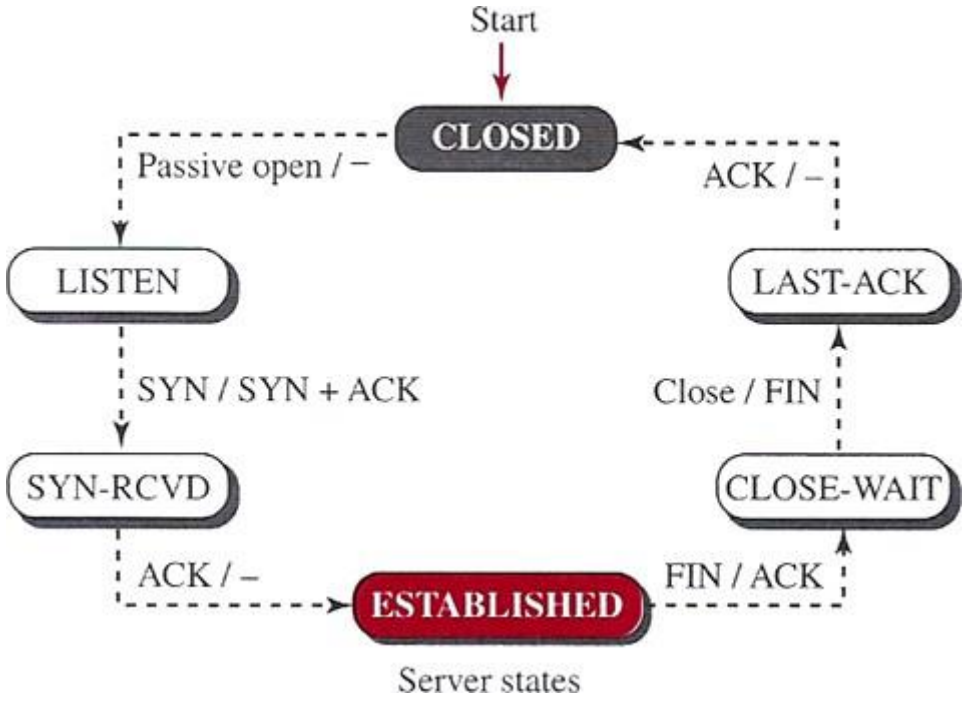


Client side: Closing

- When the client process has no more data to send, an ***active close*** command is issued.
- TCP sends a **FIN segment** and goes to **FIN-WAIT-1** state.
- When an **ACK segment** is received, it moves to **FIN-WAIT-2** state.
- When the client receives a **FIN segment** from the server, an **ACK segment** is sent and moved to **TIME-WAIT** state.
- This mode is held for 2 **MSL** seconds (**M**ax. **S**egment **L**ifetime), after which it is moved to **CLOSED** state

Transport Layer

Transmission Control Protocol (TCP): A Half-close scenario



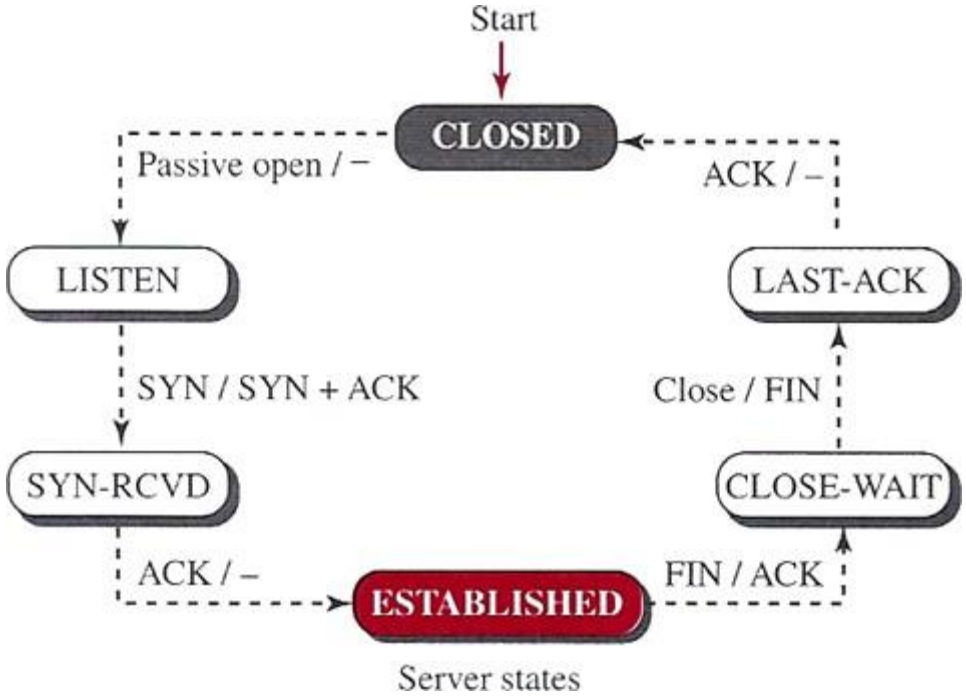
Server side: Establishment

- The server process issues a ***passive open*** command
- TCP moves to the **LISTEN** state until a **SYN segment** is received.
- TCP sends a **SYN + ACK segment**, and moves to **SYN-RCVD** state, here it waits for ACK.
- When an **ACK segment** is received, it is moved to the **ESTABLISHED** state.

Data are transferred, presumably in both directions, and acknowledged.

Transport Layer

Transmission Control Protocol (TCP): A Half-close scenario



Server side: Closing

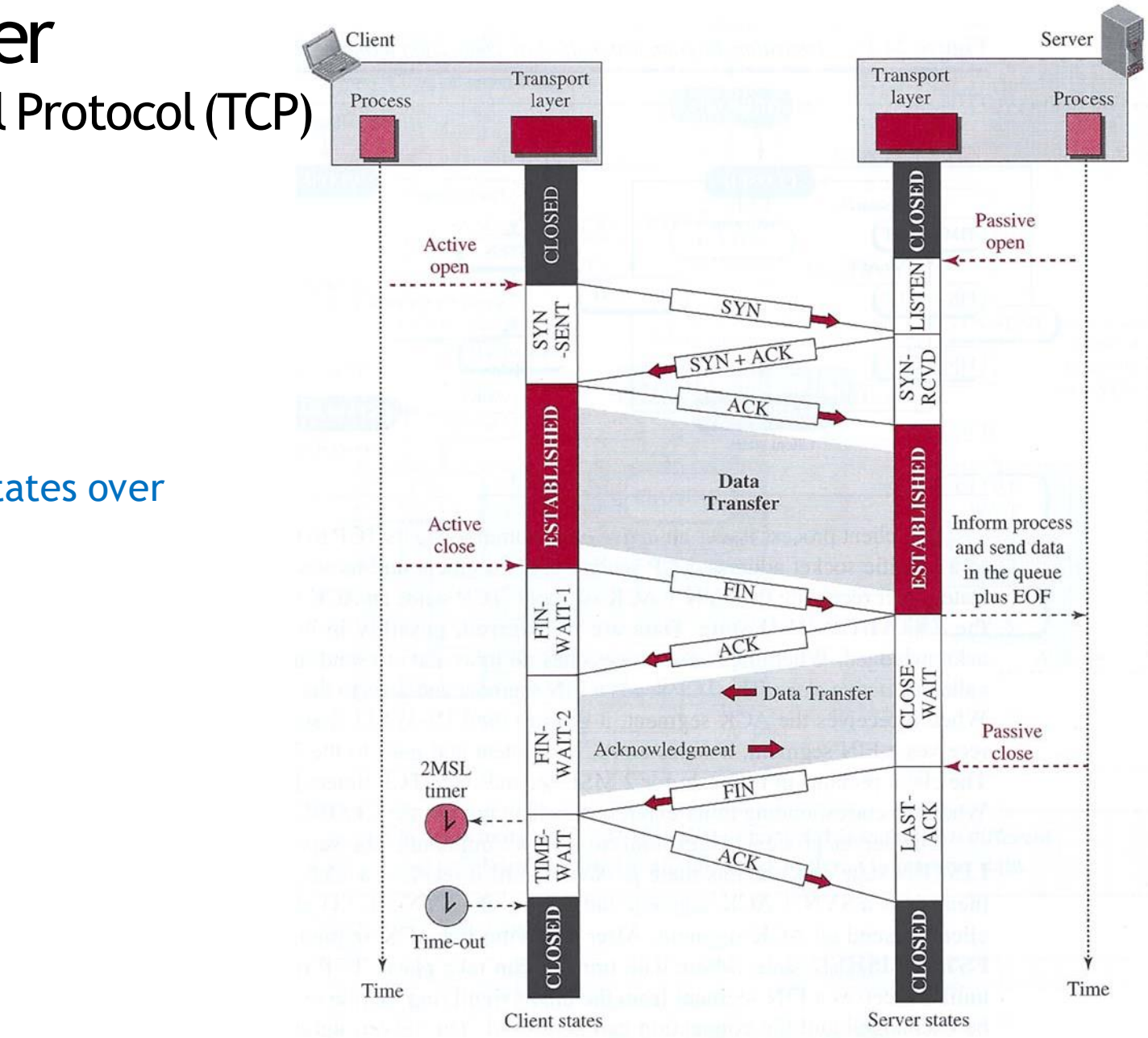
- When a **FIN segment** is received, it is the signal from the client that there is no more data to be sent.
  When the server receives this FIN segment, all queued data is transferred to the server process (at the application layer) together with a virtual EOF marker.
- An **ACK segment** is sent and moved to CLOCE-WAIT state.
- Only when a ***passive close*** command is received from the server process, then a **FIN segment** is sent and moved to LAST-ACK state.
- When the last **ACK segment** is received from the client, it is moved to CLOSED state.

Transport Layer

Transmission Control Protocol (TCP)

A Half-close scenario

the same scenario with states over the timeline.



Windows in TCP

Transport Layer

Transmission Control Protocol (TCP): Windows in TCP

Before we start looking at things like data transfer, flow-, error- and congestion-control, we just need to describe the windows that TCP uses.

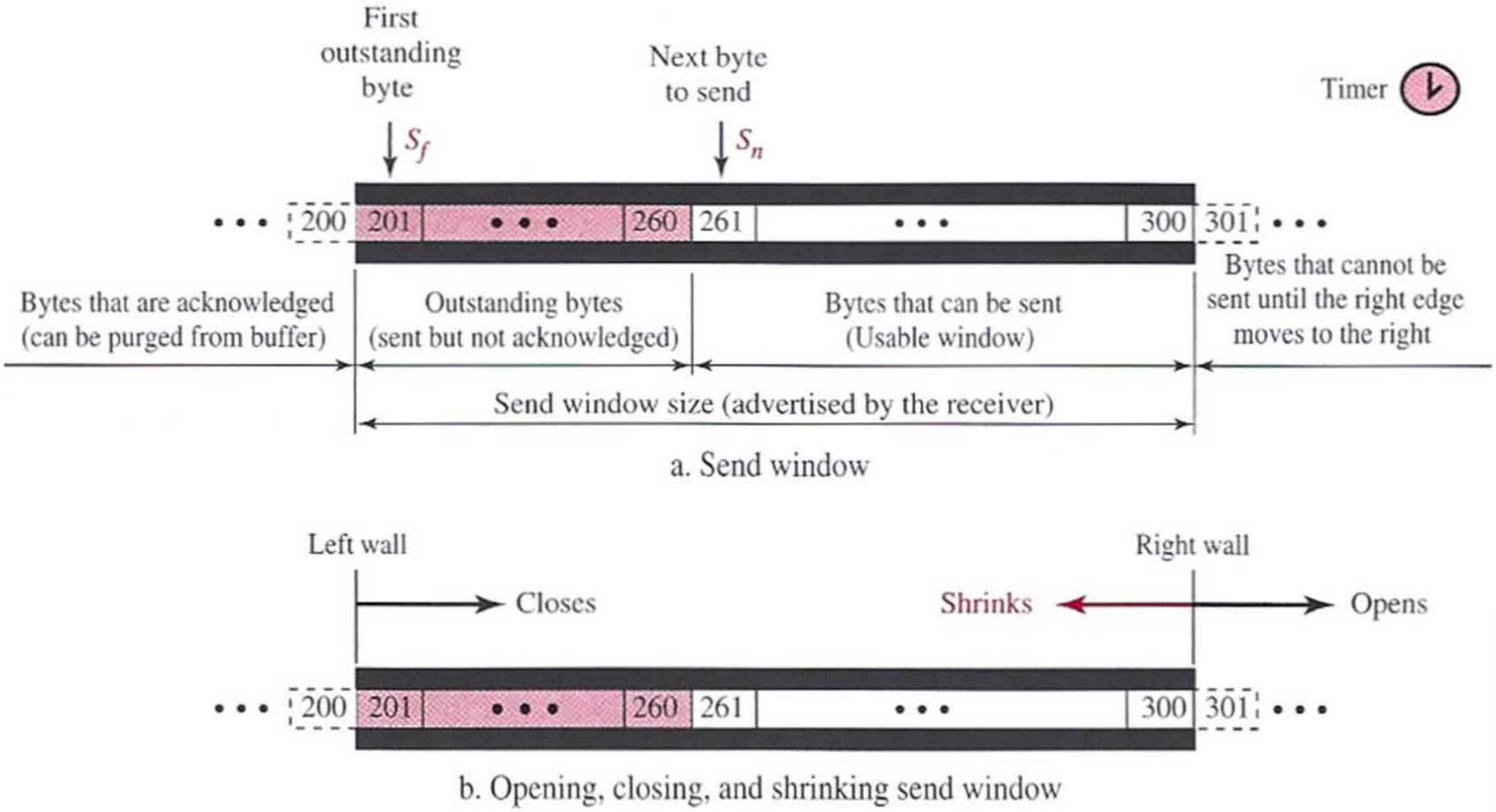
- A send window.
- A receive window.

Since data transfer is full duplex (both ways), there are of course both send and receive windows at client and server sides.

To make the discussion simple, we make an unrealistic assumption that communication is only unidirectional (say from client to server); the bidirectional communication can be inferred using two unidirectional communications with piggybacking.

Transport Layer

Transmission Control Protocol (TCP): Send window



Note: the window size here is ***100 bytes***.

Later we will see how the size of the window is determined by the receiver (in connection with flow control and congestion control)

The arrows in the figure show how the size of the window can be controlled

Transport Layer

Transmission Control Protocol (TCP): Send window

Send window in TCP is similar to what we saw with the Selective-Repeat protocol, but there are some differences:

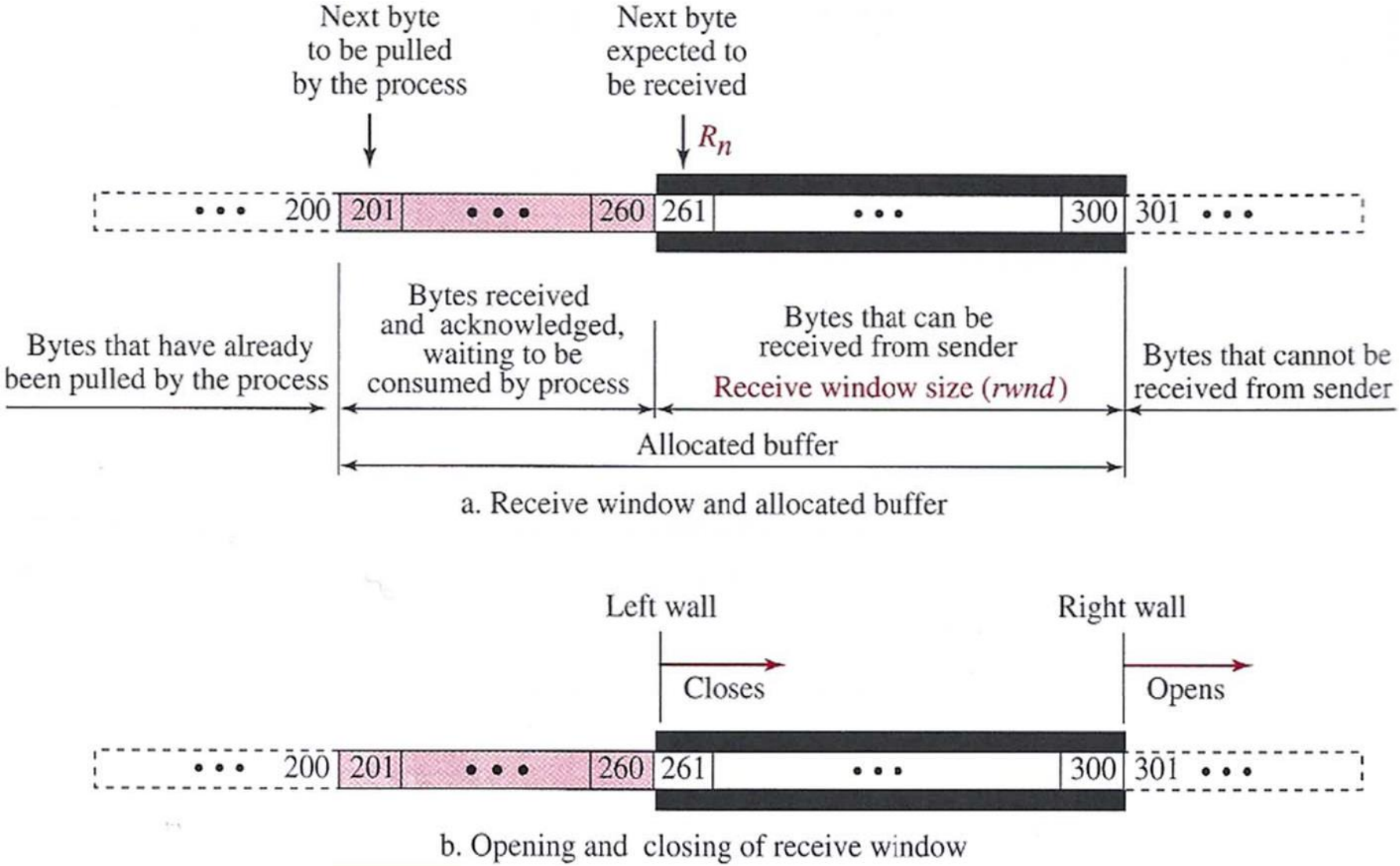
1. The Selective-Repeat window is specified in the number of packets.
   In TCP, the window is specified in number of bytes even if the transmission takes place in segments (the size is always specified in bytes)

2. 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. The theoretical Selective-Repeat protocol may use one timer for each sent packet, but the TCP protocol uses only one timer.

Transport Layer

Transmission Control Protocol (TCP): Receive window



Note: the window size here is ***100 bytes***.

Transport Layer

Transmission Control Protocol (TCP): Receive window

There are also a few differences in TCP compared to the Selective-Repeat protocol:

1. The first thing we see is that TCP allows the receiving process (on the application layer) to pull data up at its own pace.
   This means that parts of the allocated buffer may contain data that has been received and acknowledged but has not yet been retrieved by the process.
   The size of the receiver window (**rwnd**) is therefore always smaller (or equal to) the size of the buffer.
   The size of the receiver window is an expression of how many bytes can be received before data loss occurs.

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

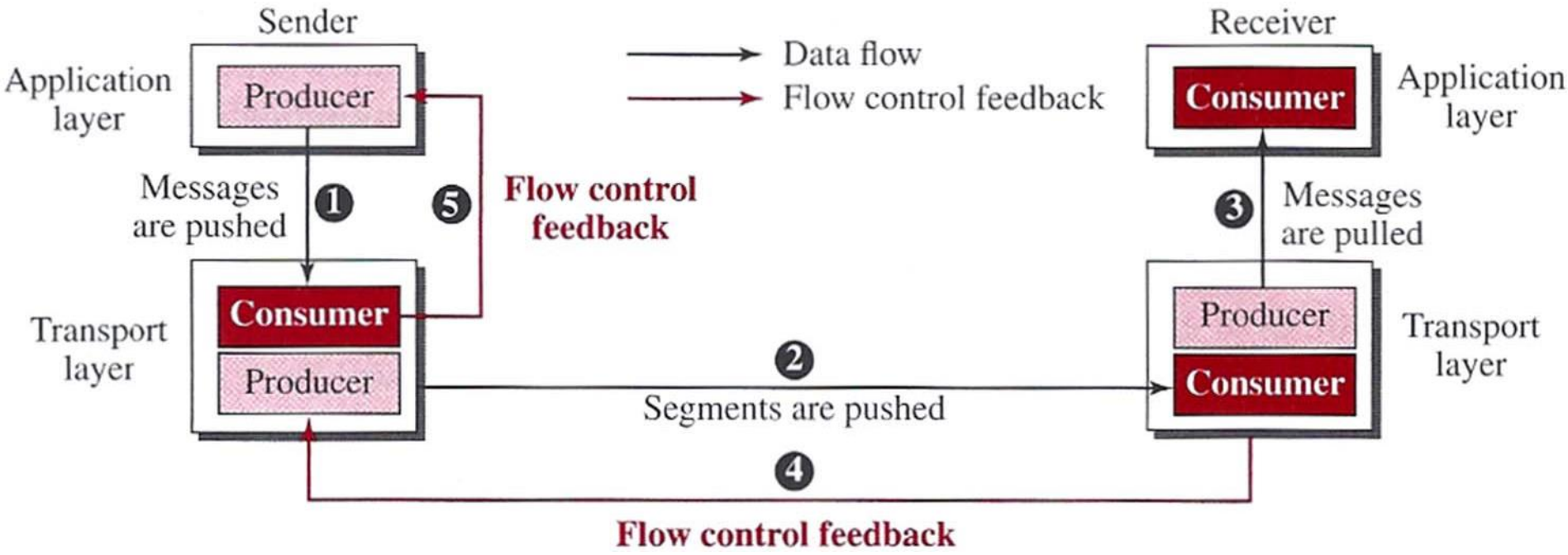
2. We recall that an acknowledge in the Selective-Repeat protocol was selective and valid for a particular package that was received flawlessly.
   In TCP, the acknowledgment mechanism is ***cumulative***, announcing the next byte that is expected to be received (similar to Go-Back-N). Newer TCP version use both cumulative and selective acknowledgments.

Transport Layer

Transmission Control Protocol (TCP): Flow control

Flow control ensures that the sender's production of data balances with the receiver's consumption of data.

TCP separates flow control from error control. Here we only look at the flow control



Flow control in TCP

Transport Layer

Transmission Control Protocol (TCP): Flow control

Opening and closing the window of the receive window

In order to achieve flow control, TCP forces the sender and receiver to adjust their windows in a coordinated way.

- The receive window closes (moves its left wall to the right) when more bytes arrive from the sender.
- The receiver window opens (moves its right wall to the right) when more bytes are pulled by the process.
- Note here that it is not allowed to shrink (move the right wall to the left) the receive window.

Opening, closing and shrinking of the send window is controlled by the receiver.

- The send window closes (moves its left wall to the right) when a new acknowledgments (**ACK**) allow it to do so.
- The send window opens (moves its right wall to the right) when the receive window size (**rwnd**) advertised by the receiver allows it to do so (new ackNo + new rwnd > last ackNo + last rwnd).
- The send window shrinks (moves its right wall to the left) in the event this situation does not occur.

Transport Layer

Transmission Control Protocol (TCP): Flow control - An example

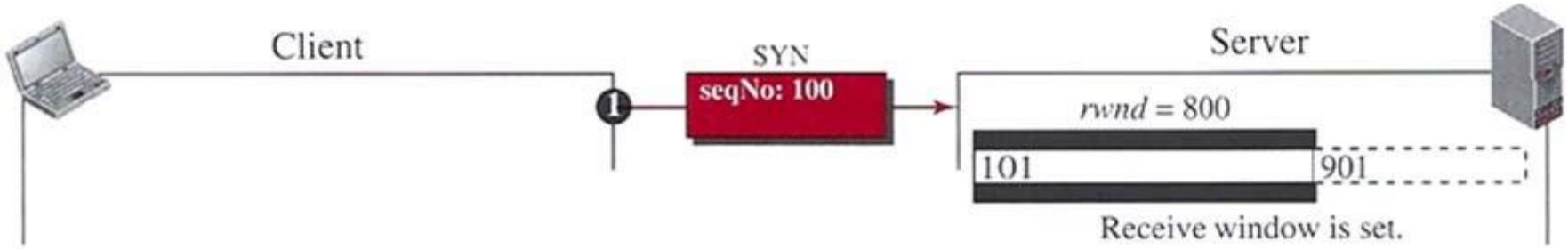
We look at how the send and receive window are set during the establishment of a connection.

Note that the example only shows one-way data transfer. We do not look at error control here either.

Note also: even if the client defines the server's send window to be 2000 bytes, it is not shown, as we only look at one-way communication here.

Transport Layer

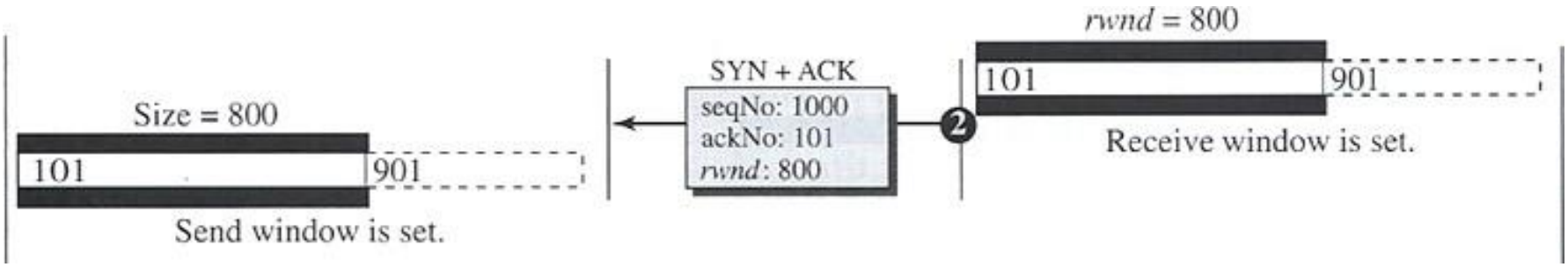
Transmission Control Protocol (TCP): Flow control - An example



The client sends a **SYN segment** to establish a connection.
The client announces its initial **seqNo = 100**.
When this segment arrives at the server, a buffer of 800 bytes (an estimate) is allocated.
The server sets its window to cover the entire buffer (***rwnd=800***).
Note that the next expected byte is ***101***.

Transport Layer

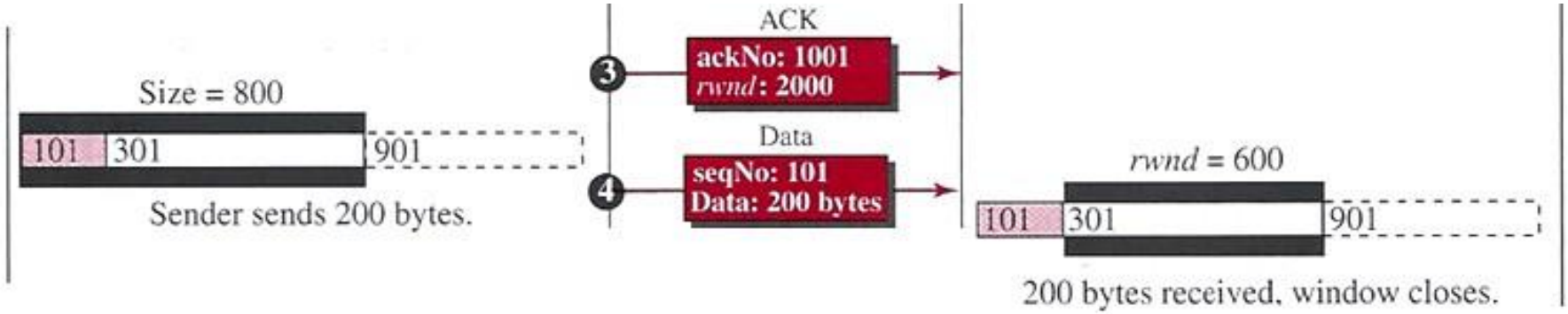
Transmission Control Protocol (TCP): Flow control - An example



The server sends a **SYN + ACK segment.**
The segment uses ***ackNo = 101*** to show that the server expects bytes starting from ***101***.
The server also announces that the client can set its window to 800 bytes. (rwnd: 800)

Transport Layer

Transmission Control Protocol (TCP): Flow control - An example

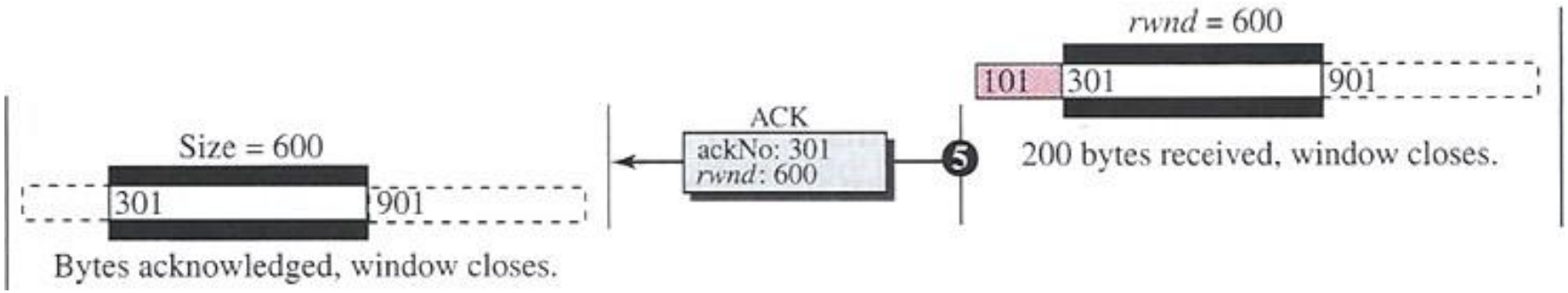


(3) The client sends an **ACK segment** (rwnd = 2000 is not shown in the figure).

(4) After the client has set its window to ***800 bytes*** (determined by the server), it sends ***200 byte*** (from ***101*** to ***300***). The client creates a segment and sends it. The segment shows ***seqNo=101*** and ***data:200 bytes***.
The client window is adjusted: ***200 bytes*** sent, but not acknowledged yet.
When the segment is received by the server, the ***200 bytes*** are saved, and the window is adjusted (now ***600*** left)

Transport Layer

Transmission Control Protocol (TCP): Flow control - An example



The server acknowledges bytes up to and including ***300*** (next expected byte ***301***).

At the same time, 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. The client's send window size is decreased to ***600*** bytes (rwnd = ***600***)

Note: Although the allocated buffer can store 800 bytes, the window cannot open (moving its right wall to the right) because the receiver does not let it.

Transport Layer

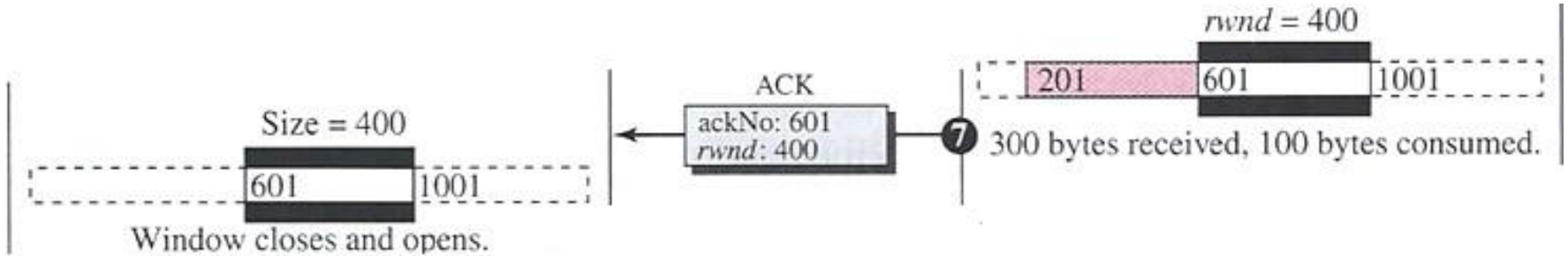
Transmission Control Protocol (TCP): Flow control - An example

The client sends ***300*** more bytes (seqNo = ***301*** data: ***300***).
The server stores the ***300 bytes***
At the same time, ***100 bytes*** are pulled by the server's process (on the application layer). 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 by 200 bytes. The receiver window size is now 400 bytes.

Transport Layer

Transmission Control Protocol (TCP): Flow control - An example

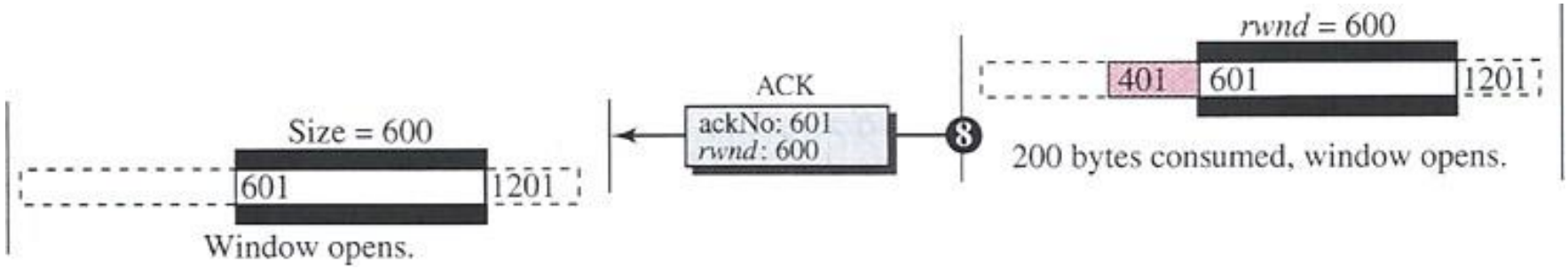


Server sends acknowledgement for ***300*** bytes (next expected byte is ***seqNo = 601***). The window size is set to ***400*** as calculated before.

When this segment arrives at the client, the client has no choice but to reduce its window again and set the window size to the value of rwnd = 400 advertised by the server. The send window closes from the left by 300 bytes and opens from the right by 100 bytes.

Transport Layer

Transmission Control Protocol (TCP): Flow control - An example



The server process pull additional ***200*** bytes and therefore adjusts the window size to ***rwnd = 600***.
The client is informed of this change by an acknowledgement segment.
Note that the next expected byte is still ***601***, as the server has not received anything.
The client adjusts its window as dictated ***rwnd = 600*** through opening its window by 200 bytes.

Transport Layer

Transmission Control Protocol (TCP): Shrinking of windows

As we have mentioned before, the receive window cannot shrink.

However, send window can shrink if the receiver defines a value for ***rwnd*** that results in shrinking the window.

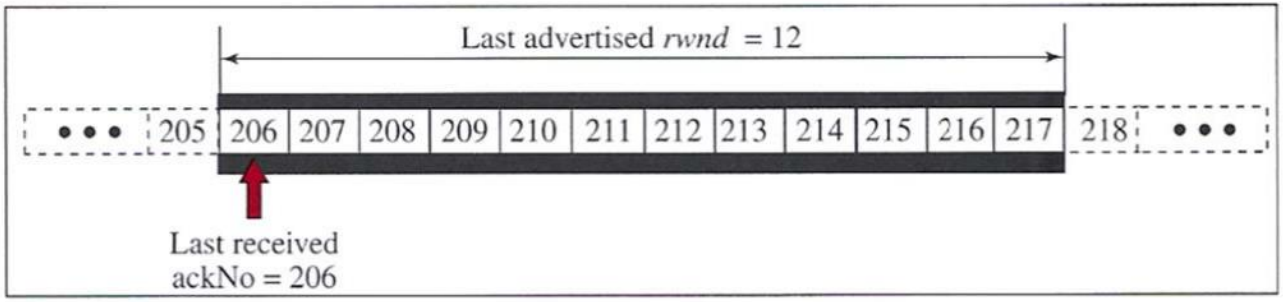
Some versions of TCP do not allow the send window to shrink. In other words, the relationship below needs to be kept:

new ackNo + new *rwnd* $\geq$ last ackNo + last *rwnd*

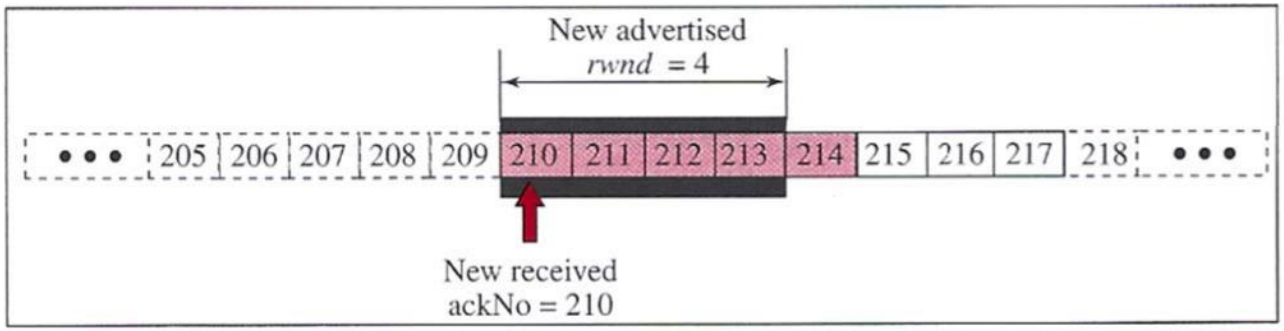
The inequality is a mandate for the receiver to check its advertisement.

Transport Layer

Transmission Control Protocol (TCP): Example



a. The window after the last advertisement



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

Here's an example showing what can go wrong. This happens between a and b.

- The sender has sent bytes **206 - 214**, from which only bytes **206 - 209** have been acknowledged.
- New **rwnd** is announced to **4**. (210+4 < 206+12)

Problem: byte 214 is outside the window even though it is sent but not acknowledged.

Since the receiver cannot know if all bytes up to 217 are being sent, the right-hand wall of the window must not be moved to a value smaller than 217.

Transport Layer

Transmission Control Protocol (TCP): Window Shutdown

There is a special case where the receiver can set **rwnd=0**.
This happens if the receiver does not want to receive data from the sender for a while (received data may need to be processed).

The sender does not set its window to 0, but simply stops sending until a new ***rwnd*** announcement arrives.

The sender can send a 1-byte segment to "test the connection" this is called: ***probing***.

It can give a lot of overhead though. Remember that 1 byte sent via TCP/IP provides: 20-byte TCP header + 20-byte IP header + 1 byte

Error control in TCP

Transport Layer

Transmission Control Protocol (TCP): Error Control

Error control consists of the following mechanisms:

- Detecting and resending corrupted segments.
- Resending lost segments.
- Storing out-of-order segments until missing segments arrive.
- Detecting and discarding duplicated segments.

Error detection and correction in TCP is achieved as follows:

- Checksum.
- Acknowledgment
- Time-out and Retransmission.

Transport Layer

Transmission Control Protocol (TCP): Checksum

Each segment contains a checksum field, which is used to identify corrupted segments.

If a segment is corrupted, then it is discarded and considered lost.
TCP uses a 16-bit checksum.

The 16-bit checksum is however considered insufficient today for new transport protocols, e.g., SCTP.

You cannot change it for TCP due to the compatibility.

Transport Layer

Transmission Control Protocol (TCP): Acknowledgment

TCP uses acknowledgment to acknowledge received data segments. Control segments that do not carry data are also acknowledged.
Note: ACK segments are not acknowledged.

Cumulative Acknowledgment (ACK): Originally, TCP was designed for cumulative acknowledgment. The receiver advertises the next byte it expects to receive, ignoring all segments received and stored out of order.
The ACK field in the TCP header indicates the number of bytes expected.
The number is only valid if the ACK flag is set to 1.
Discarded, lost or duplicate segments are not acknowledged.

Selective Acknowledgment (SACK): More and more versions of TCP are adding another type of acknowledgment called Selective Acknowledgment.
SACK does not replace ACK but provides additional information to the sender.
A SACK reports a block of bytes that are out of order or duplicate.
Since there is no place in the TCP header where such information can be inserted, SACK is implemented ***as an option in the end of the header***.

Transport Layer

Transmission Control Protocol (TCP): Retransmission

The heart of the error control mechanism is the retransmission of segments.
When a segment is **corrupted**, **lost** or **delayed**, it is retransmitted.

Two methods used today:
Retransmissions Time Out (RTO): Here TCP manages one timer for each connection. If this timer expires, the segment will be retransmitted (the segment in the front of the queue).
The timer restarts.
The value of the timer is based on the Round-Trip Time (RTT) and is adjusted dynamically.

Retransmission after three identical ACK segments: This rule applies if a segment is lost at the receiver.
If the sender continues to send segments, then there will be more out-of-order segments (since one is missing).
The receiver's buffer will become full!
The solution here is to retransmit the lost segment after receiving three identical ACK segments! The sender can see that segments from the receiver contain the same ACK value (namely on the lost one).
This feature is called **fast retransmission**.

Transport Layer

Transmission Control Protocol (TCP): Out-of-order segments

TCP was originally designed to discard all out-of-order segments..
Equivalent to Go-Back-N

This resulted in the missing and all subsequent segments to be retransmitted.

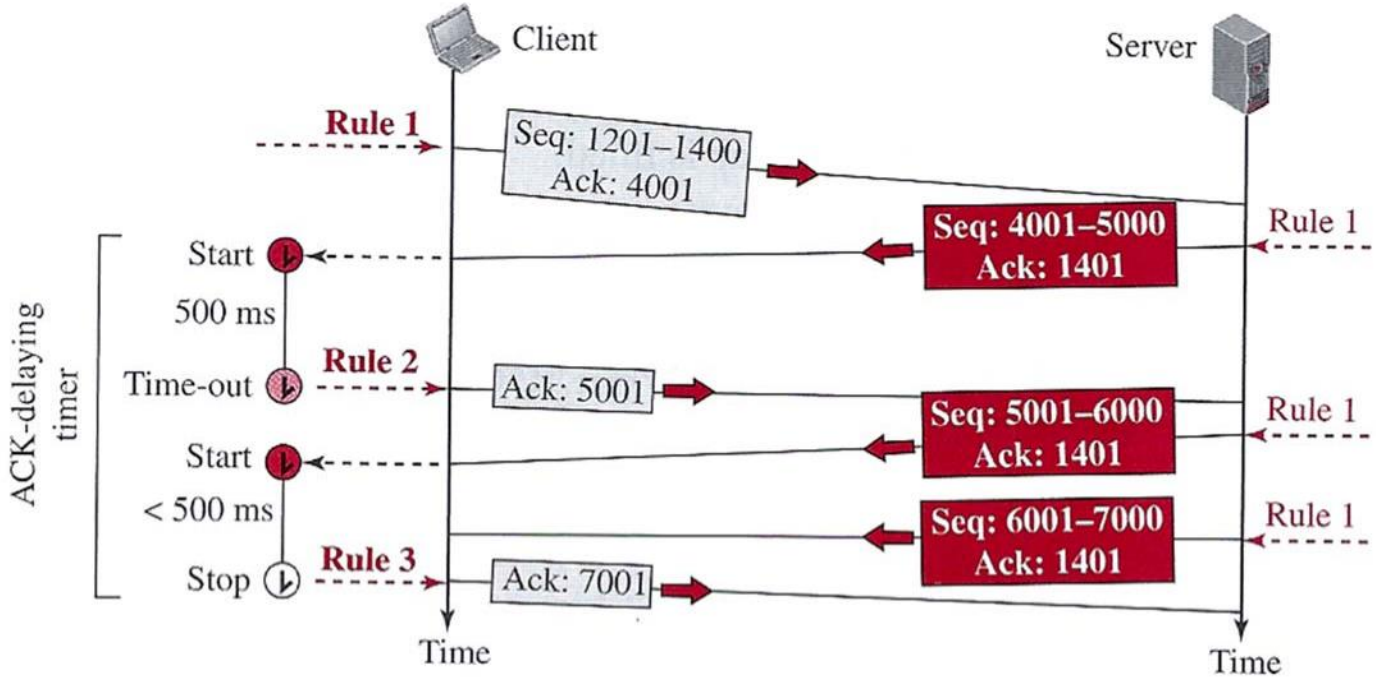
Most implementations today do not discard these out-of-order segments but store them temporarily until the missing segment arrives.

(outstanding segment = sent but not acknowledged)

Note that out-of-order segments are never delivered to the process. TCP guarantees that data are delivered to the process in order.

Transport Layer

TCP: ACK Rules - ACK Delayed



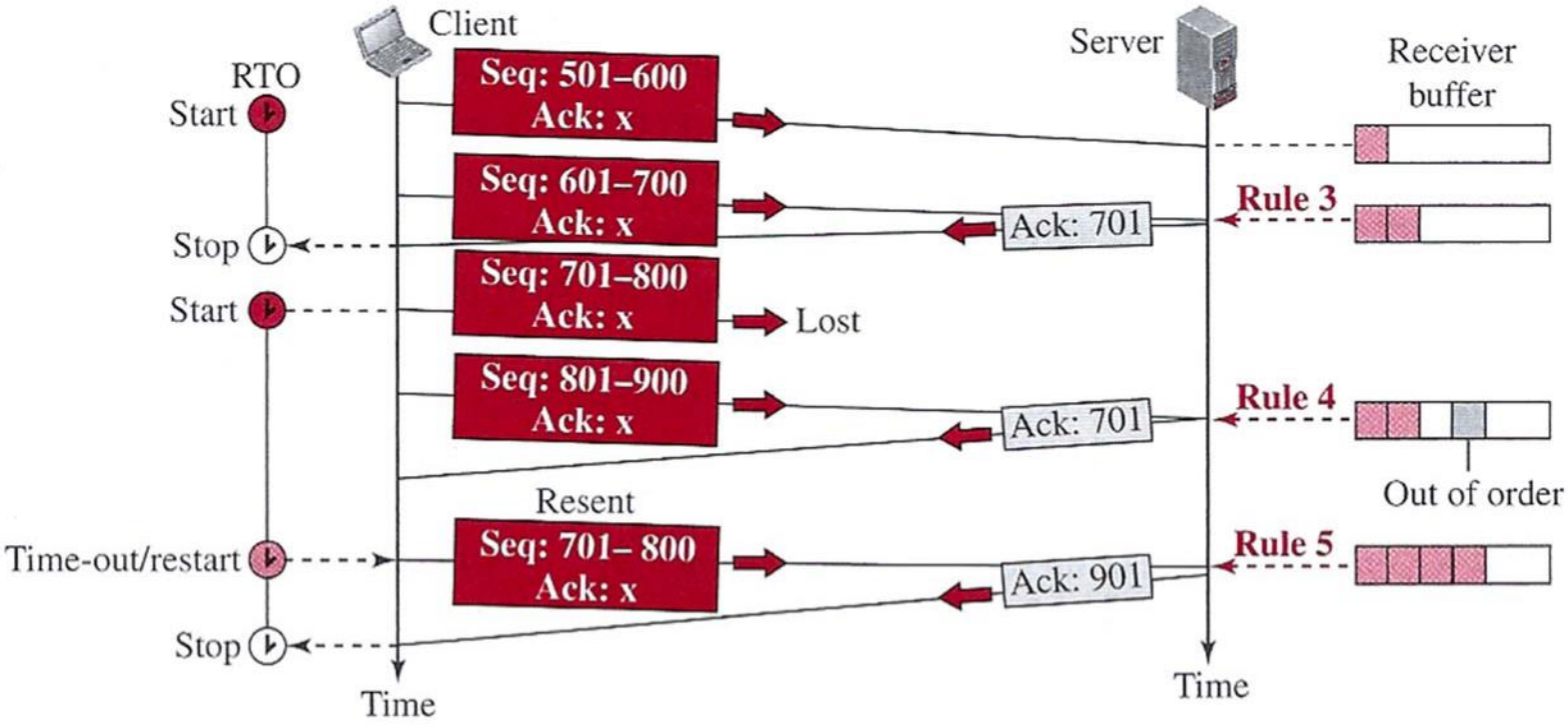
When the client receives the first segment from the server, it has nothing to send back, so it sends only one **ACK segment.**

But the receipt is only sent after **500** ms to see if more segments should come.

The next two segments are sent within **500** ms, therefore only be acknowledged after the last segment has arrived. Note cumulative ACK

Transport Layer

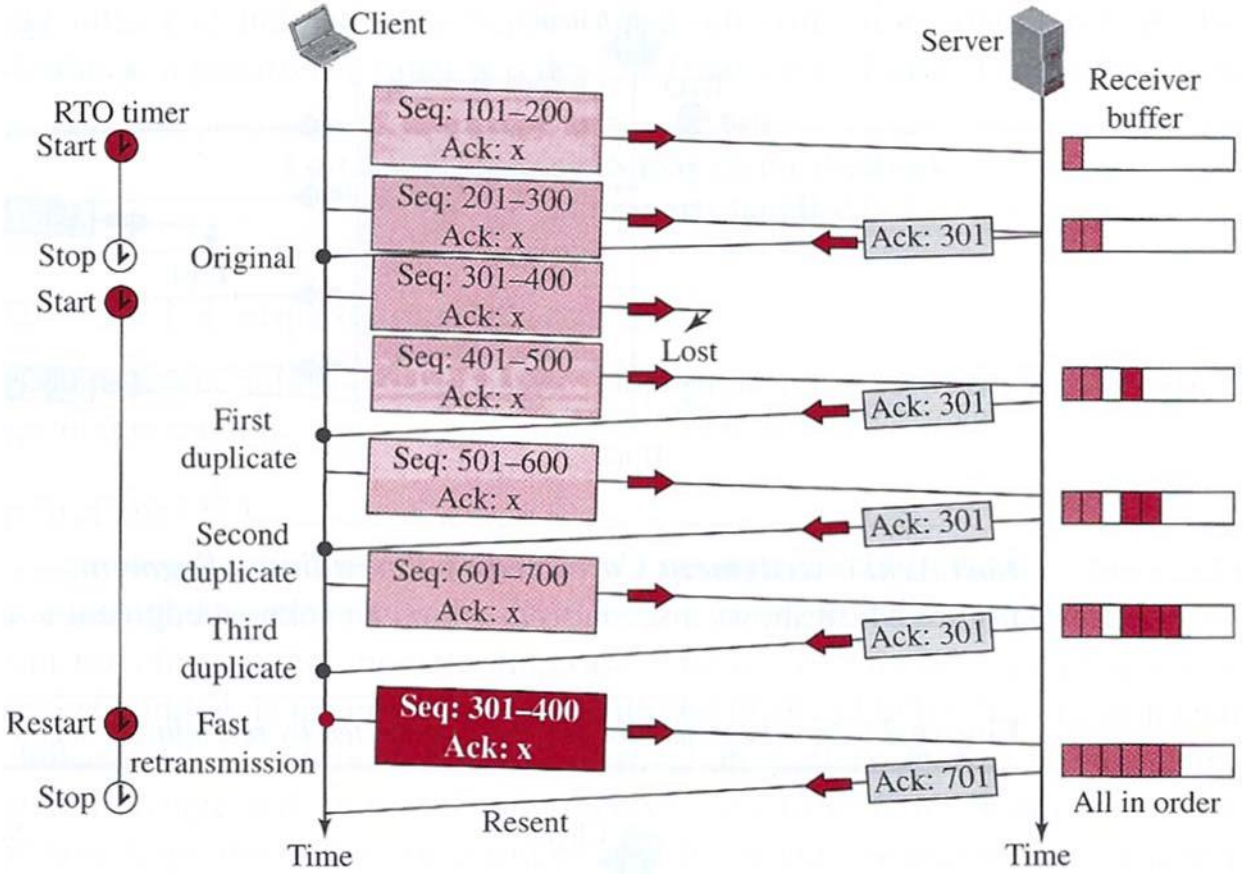
TCP: ACK Rules - Lost Segments



- The sender sends segments 1 and 2, which are acknowledged by the receiver.
- The sender sends segment 3, but it is lost! One timer has started (RTO = Resend Time Out)
- The sender sends segment 4, the receiver stores the segment with a gap and acknowledges it (still with 701)
- The timer (RTO) expires and segment 3 is resent, the receiver acknowledges (with 901 coming after segment 4 cumulative ACK)

Transport Layer

TCP: ACK Rules - Fast retransmission



Same as before, it is segment 3 that is lost.

Although the timer for segment 3 has not expired, the sender has received an ACK for segments 4, 5 and 6, all of which have the same AckNo. of 301.

Therefore, segment 3 is retransmitted immediately.

Transport Layer

TCP: ACK Rules - Delayed Segments

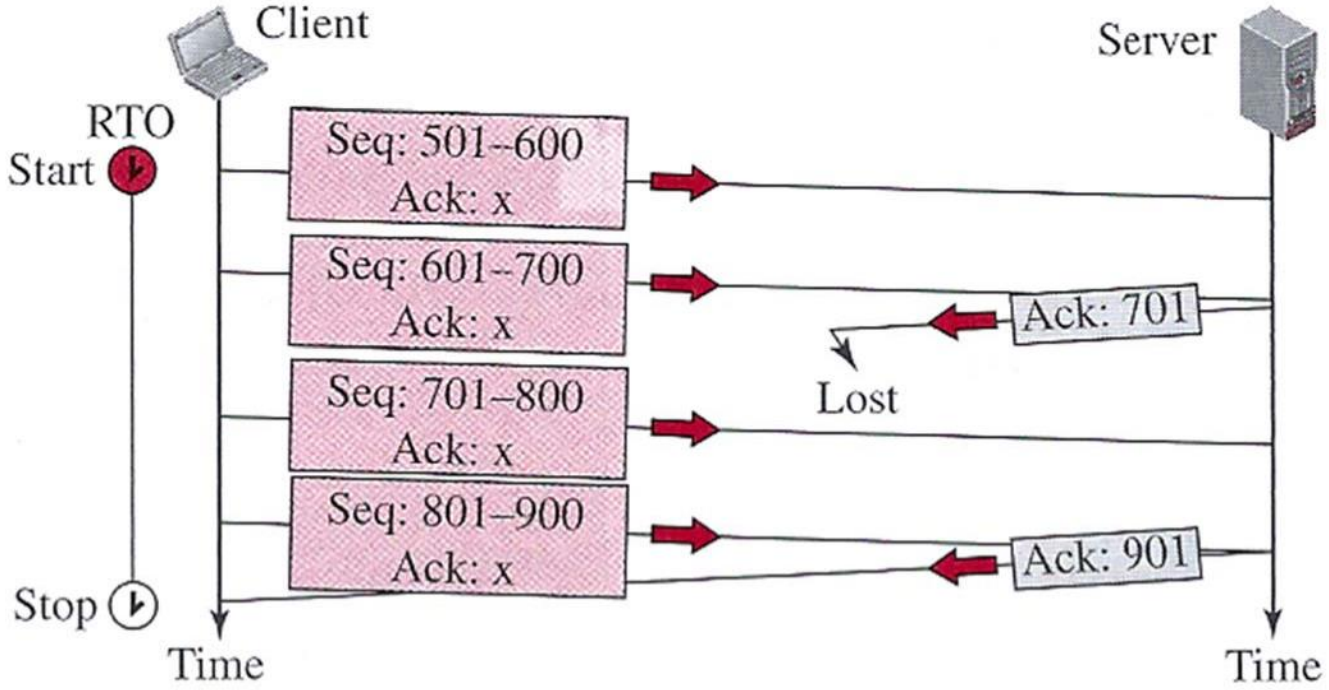
TCP segments are encapsulated in IP datagrams, can travel along different routes to their destination with different delays.

Hence TCP segments may be delayed. Delayed segments sometimes may time out (RTO timer) and be resent.

If the delayed segment arrives after it has been resent, it is considered a duplicate segment and discarded.

Transport Layer

TCP: ACK Rules - Automatic corrected lost ACK

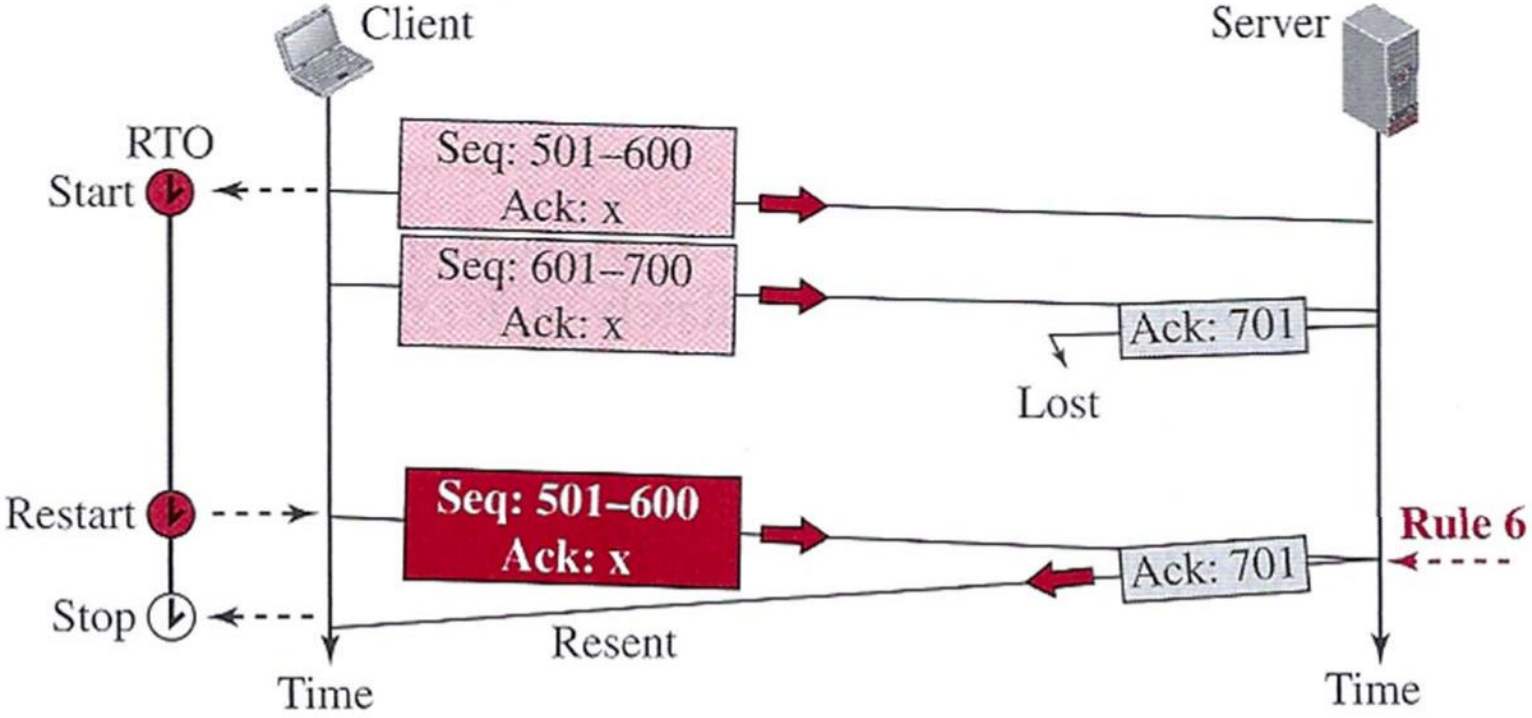


Here, the RTO timer fails to time out before ACK: 901 arrives.
This ACK (901) also acknowledges the lost one.
Since the ACK indicates the number 901, it means that everything up to 900 has been received correctly.

TCP uses cumulative acknowledgment. The next acknowledgment automatically corrects the loss of the previous acknowledgment.

Transport Layer

TCP: ACK rules - Lost ACK corrected by resending a segment



The next acknowledgment is delayed for a long time or is lost.

The correction is triggered when the RTO timer reaches that time, and the first segment is retransmitted, and will be discarded as a duplicate segment.

The receipt ACK: 701 tells the client that the second segment should not be resent. It has been received correctly.

Transport Layer

A brief comparison between the two transport protocols

UDP is a **message-oriented** protocol.

- A process (from the top layer) delivers a message to the UDP, which encapsulates and sends it over the network.
- UDP limits the size of the information, as each message is independent of each other.
- UDP can be used for IP telephony and real-time communication if stable data flow is important.
- UDP is unreliable, so a sender does not know the fate of sent messages. (lost, duplicated, out-of-order, corrupted, etc.)
- UDP lacks flow and congestion control.

TCP is a **byte-oriented** protocol.

- A process (from the upper layer) delivers data to TCP, which composes the messages into a stream of bytes sent as segments over the network.
- Since TCP provides a data stream, there is no limit to the size of the information.
- TCP is reliable. Duplicate segments are detected, the lost and corrupted segments are retransmitted, and all bytes are delivered to the receiver process in the order they were sent.
- TCP has flow and congestion control.