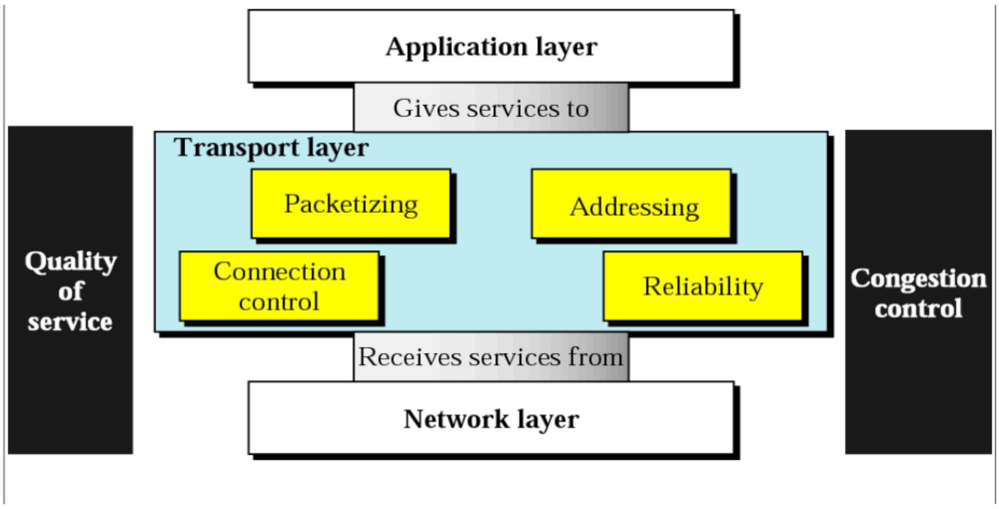
lecture 10 Transport Layer

Services provides by transport layer



Transport vs Data Link

- function is similar
- transport layer manages traffic **across an internetwork (end to end delivery)**

Process to process delivery

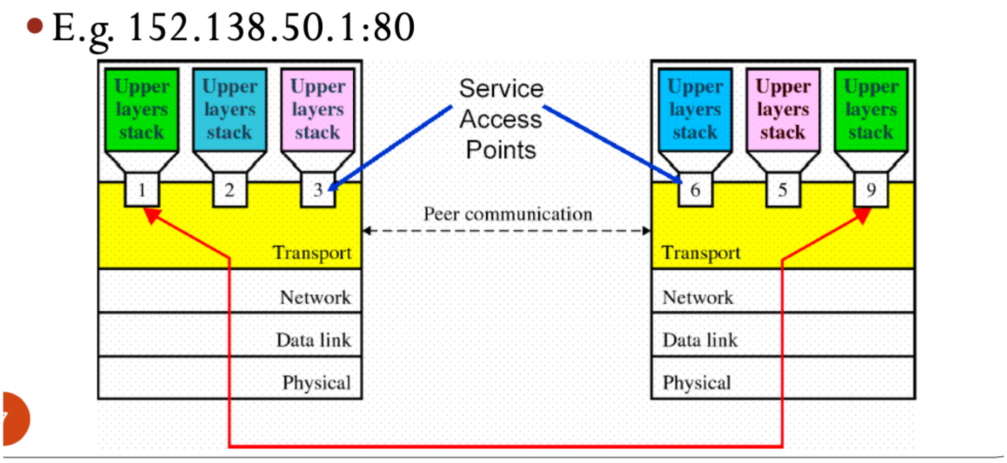
- provide logical communication between application processes
- run in *end system*
  - message to *segments*, pass to the network layer
  - reassembles segments into messages, passes to application layer
- more than one process running in the same host
  - *multiplexing / Demultiplexing*

Service Access Points

- allows *multi-tasking*
- delivery message between *applications running on the two machines*
- through **service access points**
- To identify the end applications individually, assign a **service-point address**, or **port number** to each application

Port Number

- *16 bits* integer
- **socket address = IP address + port number**



- E.g. 152.138.50.1:80

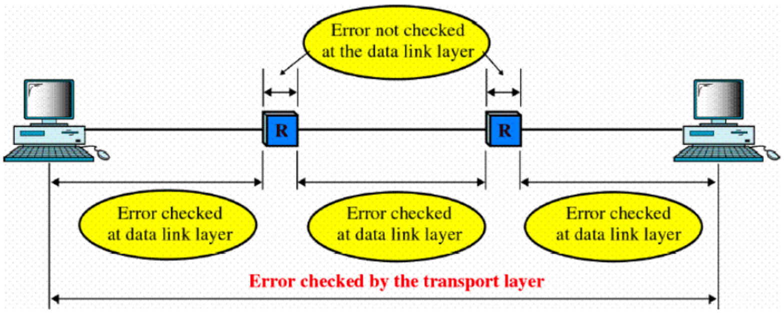
- well-known ports: assigned and controlled by IANS
- Registered ported: prevent duplication
- Dynamic ports: *used by any process*

Connection Control at Transport Layer

- connectionless or connection-oriented
- **connectionless**
  - no connection establishment or connection release
  - segment arrive may *out of sequence*
  - *no acknowledgement*
- **Connection-oriented**
  - Establish connection before data transfer
  - Release at the end
  - arrive in order
- Connection - oriented delivery incurs *more overhead*

Error Control at Transport Layer

- data link layer - CRC
- transport layer - **checksum**
- data link layer is reliable, why need this?
  - *IP is best-effort delivery*
  - errors at the IP layer


Error not checked at the data link layer
R
R
Error checked at data link layer
Error checked at data link layer
Error checked at data link layer
Error checked by the transport layer

Flow Control at Transport Layer

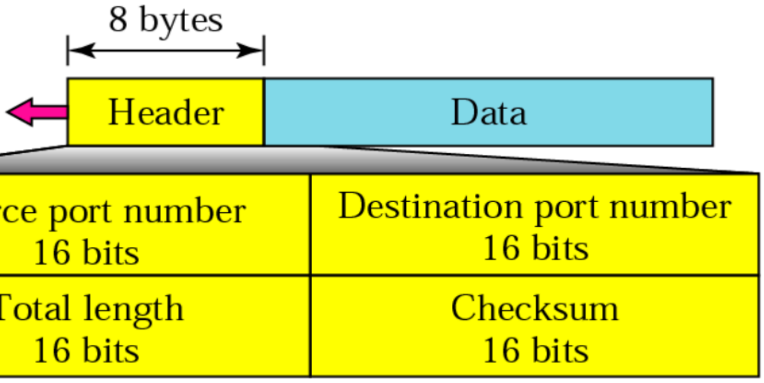
- **sliding window mechanism**
- window size may be variable
  - advertised by the *receiver in the ack*
  - according to the available buffer size

Internet Transport-Layer Protocols

- Transmission Control Protocol (TCP)
  - Connection-oriented
  - Flow and error Control
  - Congestion Control
- User Datagram Protocol (UDP)
  - Connectless
  - Unreliable

User Datagram Protocol (UDP)

- UDP **Header Fields**
  - Source and destination *port address*
  - Length: total length of *entire segment* (in *byte*)
  - checksum: error detection (head plus data) (optional)



- Data sequence *not guaranteed*
- Reception *not guaranteed*
- Connectionless
- Data can be sent to **multiple destinations** and received from **multiple** destinations
- No flow control
- Route updating protocols
- *Real time* data

Transmission Control Protocol (TCP)

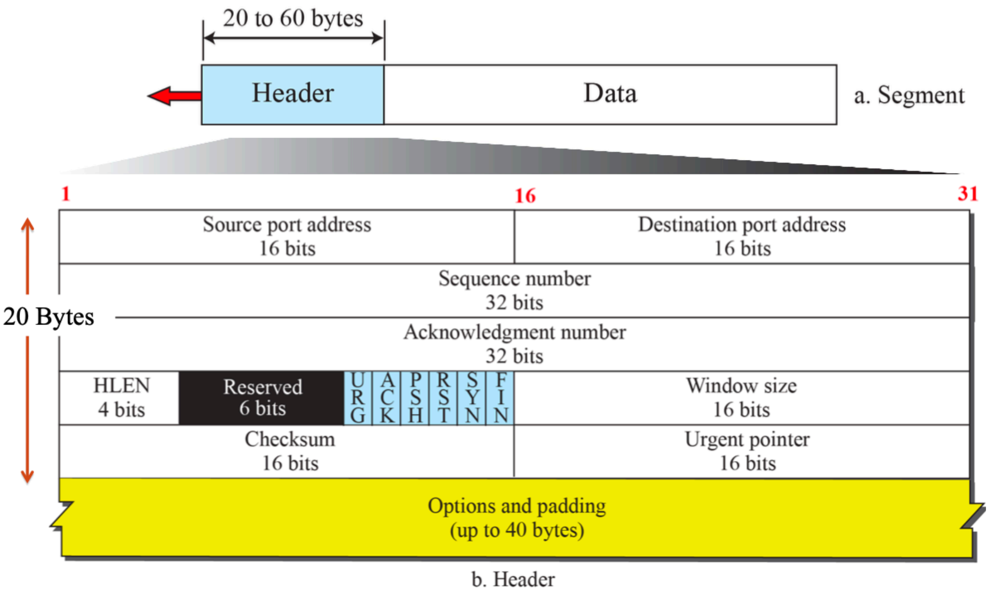
- reliable but complex
- **connection-oriented**
- **stream-oriented**: data as a *stream of byte*
- begin: alerting the receiver
- end with explicit connection termination

Basic Operation of TCP

- divides *long stream of data* into small data units called **segment**
- segments are carried across networks, encapsulated inside IP datagram
  - IP may divide a TCP segment into *multiple IP fragments*
- reorders the segments base on their **sequence numbers**
  - may arrive out of order and or with errors

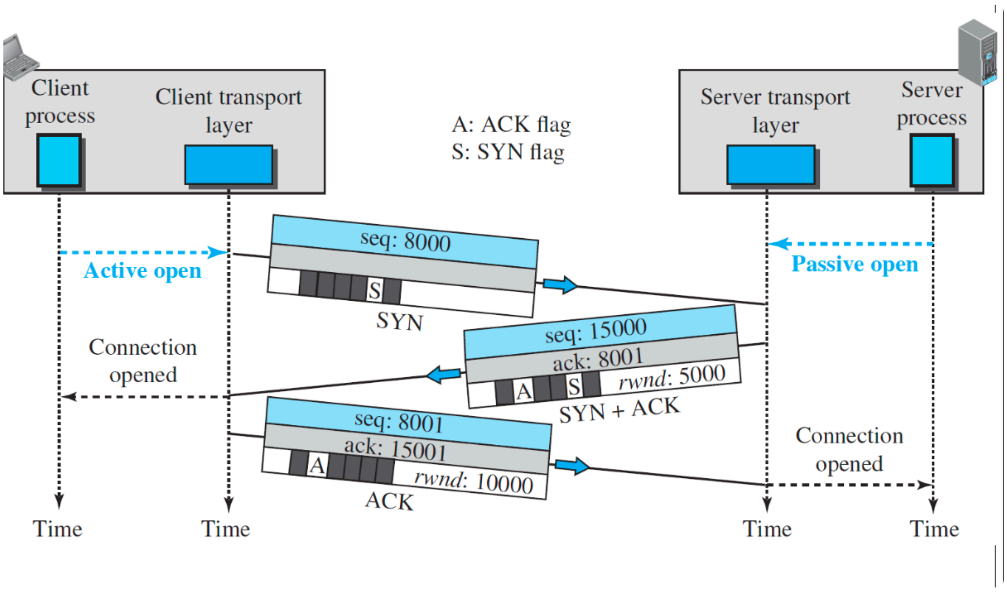
TCP Segment Structure

	Field	Size (bits)	Description
1	Source Port Address	16	Identifies the application program in the source computer
2	Destination Port Address	16	Identifies the application program in the destination computer (e.g., Telnet = 23)
3	Sequence Number	32	Specifies the byte number assigned to the first byte of data in the segment
4	Acknowledgment Number	32	Valid **only if the ACK bit is set** indicates *the next expected byte* sequence number
5	Header Length (HLEN)	4	Specifies the *length of the TCP header* in units of 4 bytes
6	Reserved	6	Reserved for future use (*currently all 0s*)
7	Control Field	6	Includes the following control flags: **URG**: this segment is urgent, and urgent pointer field is significant **ACK**: when set, ack number is valid **PSH** : push, read *immediately* **RST**: reset **SYN**: *connection establishment* **FIN** *finish, termination*
8	Window Size	16	Defines the *size of the sliding window* (maximum value: 2^{16} - 1)
9	Checksum	16	Used for error detection; includes a pseudo-header containing IP information
10	Urgent Pointer	16	Valid **if URG is set;** defines the end of urgent data and the start of normal data
11	Options	Variable up to 40	Additional optional fields; commonly used options: Maximum Segment Size, Window Scale Factor, Timestamp
12	Maximum Segment Size	Variable	Specifies the *maximum segment size* during initial connection request
13	Window Scale Factor	Variable	Used during connection setup to scale the window size (default unit: bytes) a *scale factor* to the window field in unit of 2^F bytes, where max F=14
14	Timestamp	Variable	Used to calculate *round-trip* delay by requesting the receiver to return a time value



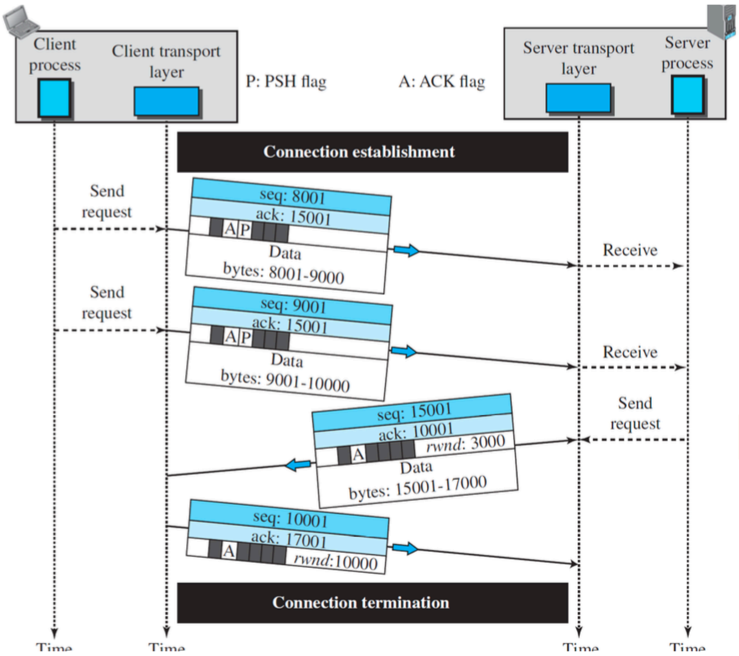
Connection Establishment

- **full-duplex mode**
- **Three way handshaking**
- *Server* program tells its tcp that it is ready to accept a connect: **passive open**
- *Client* program issues a request to its tcp for a **active open**
- SYN segment cannot carry data, but it *consumes* one sequence number
- SYN + ACK cannot carry data, but does *consumes* one sequence number
- ACK if carry no data, consumes *no sequence number*



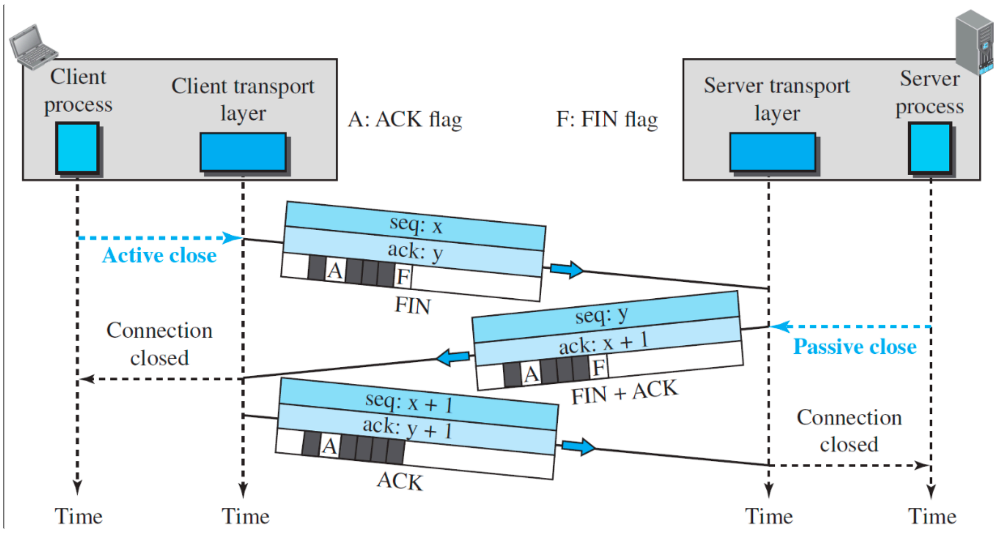
Data Transfer

- The Push bit (*PSH*): as soon as possible and not to wait for more data to come

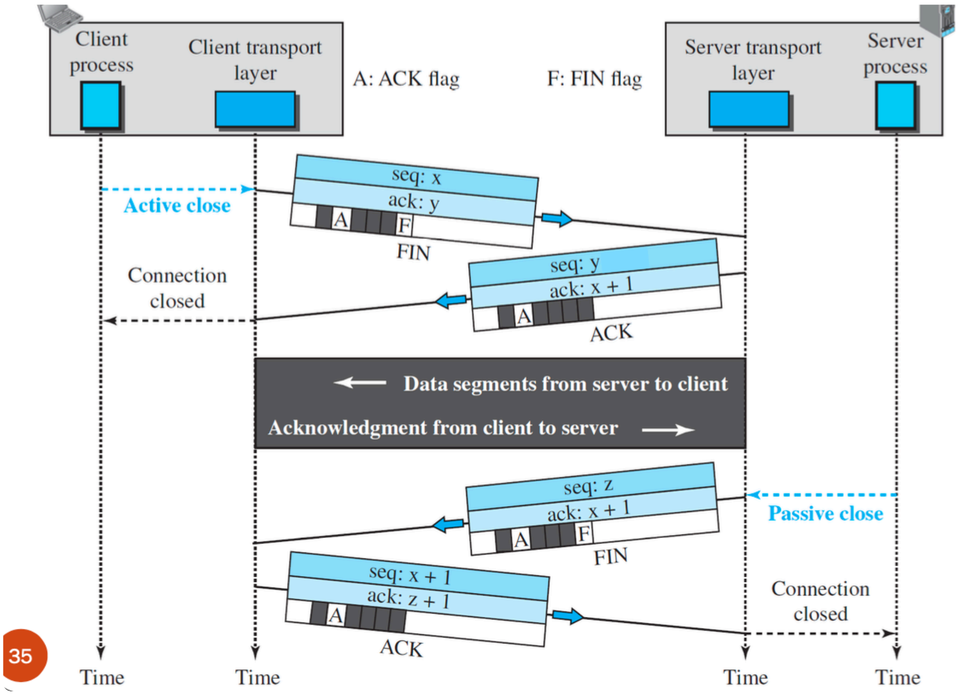


The push bit (PSH) is to let the receiving TCP know that the segment includes data that must be delivered to the receiving application program as soon as possible and not to wait for more data to come

Connection Termination Using Three-Way Handshake



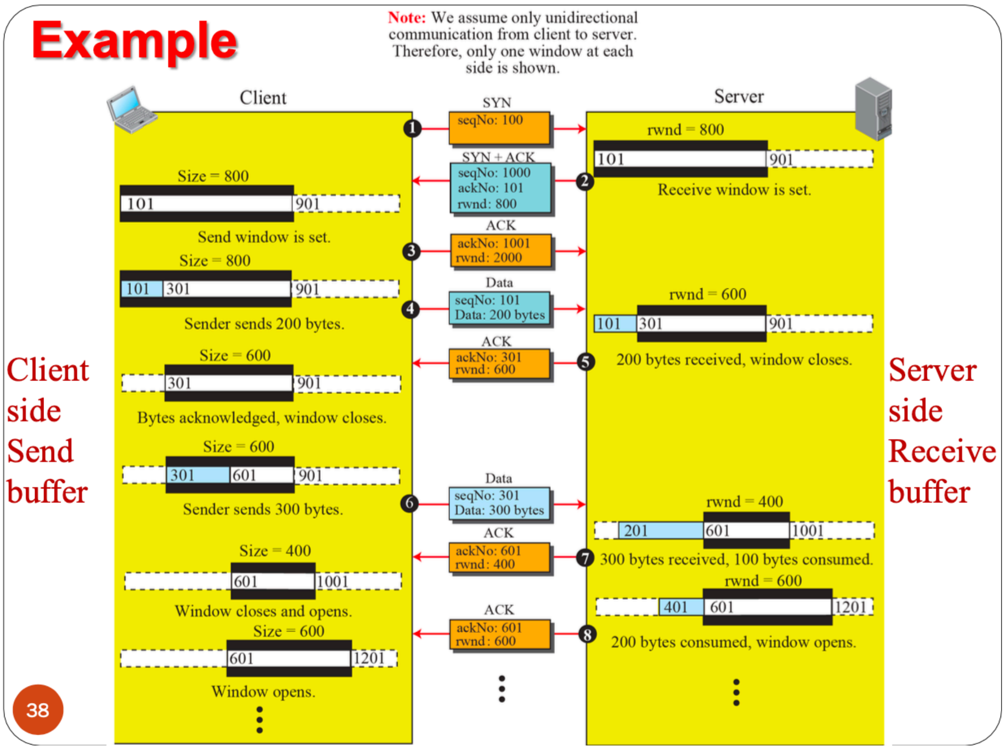
Half close



- FIN and FIN + ACK consume sequence number
- ACK if no data, do not consume

TCP Flow Control

- The window size **rwnd** may be variable
  - advertised by the **receiver** in the ack message, according the **available buffer size**
  - `rwnd = RcvBuffer – Buffered Data`


Example

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

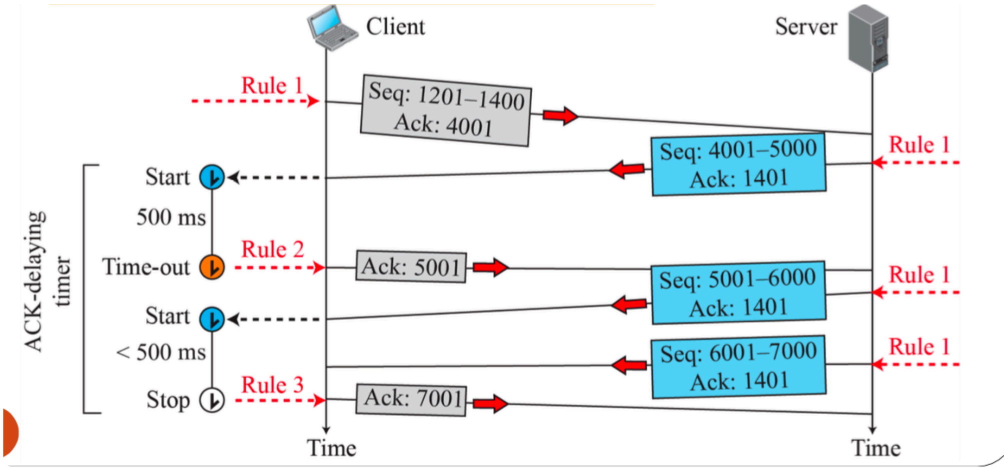
Error Control

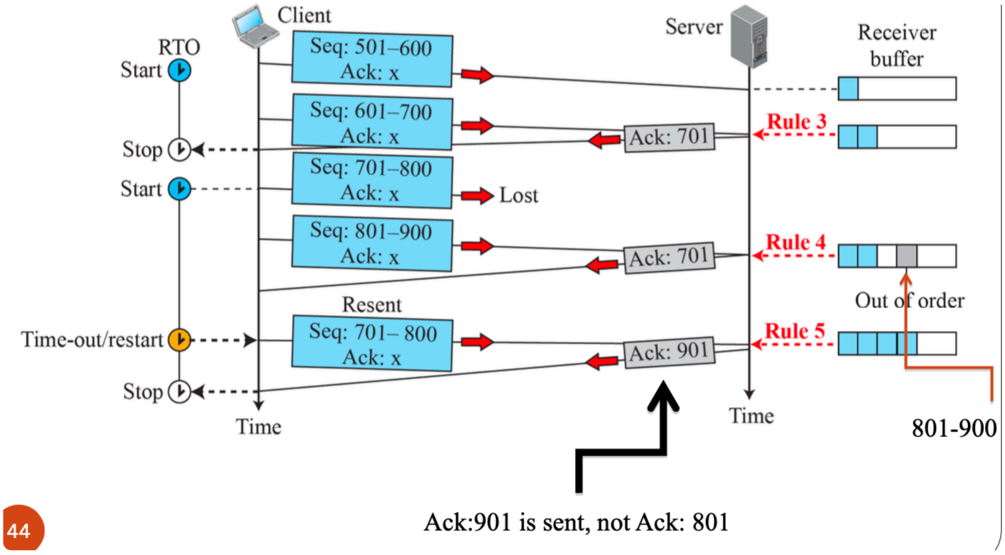
- *Detect and resent* corrupt segments
- *resend* lost segments
- *storing* out of order segment
- *detect and discard* duplicated segments

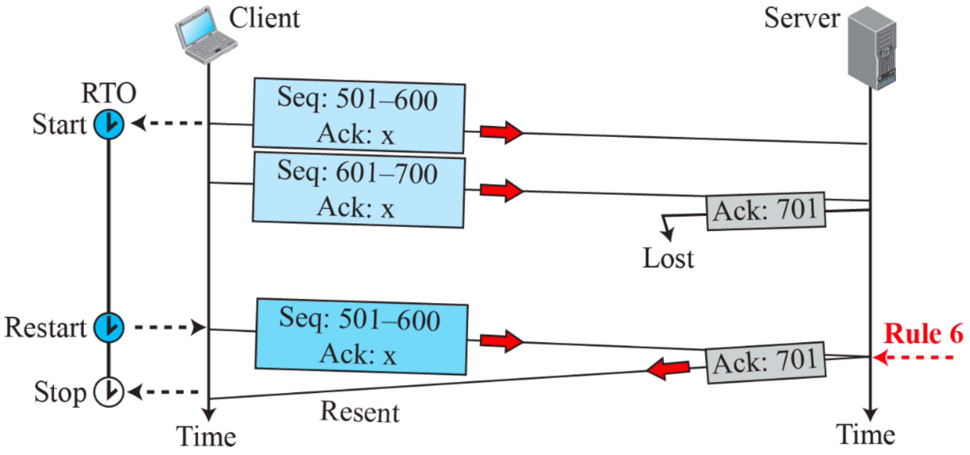
Acknowledgments

Use ack to confirm the receipt to data segments
RULES: to **generate ack**:

1. ack include next sequence number it expects in data segment
2. if **receiver** has no data to send, **delays** sending a ACK until another segment arrives or until a period of time
3. segment arrives right and pre not been ack, *immediately* sends an ACK
4. when segment *out of order*, number is higher; the receiver *Immediately* sends ACK (next expected segment)
   - *sender* retransmits the segments **in front of queue** after *RTO*
   - tcp today store out-of-order segments until the missing arrives
   - **Retransmission time out** (RTO): **Sending TCP** maintains for **each connection**
5. when missing arrives, receiver send ACK the next sequence number expected
6. if *duplicate segment*, receiver discards the segment, *immediately* send expected



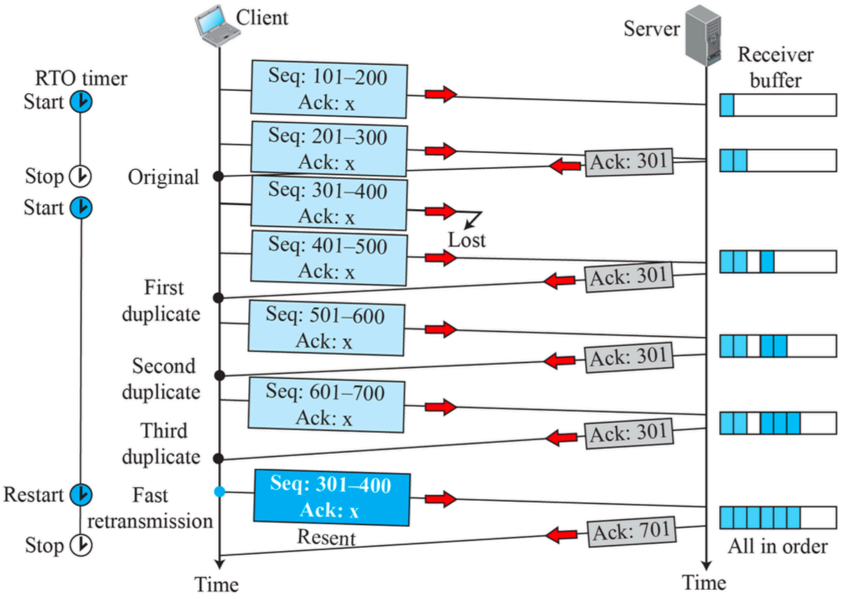




Retransmission

- In addition to *RTO*, today retransmits the *missing segment* **immediately when three duplicate ACK arrived**
- avoids long delay of RTO

◆Three Ack:301 are received before RTO



Client
Server
Receiver buffer
RTO timer
Start
Stop
Original
Start
Seq: 101–200 Ack: x
Seq: 201–300 Ack: x
Ack: 301
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
Ack: 301
Third duplicate
Restart
Fast retransmission
Seq: 301–400 Ack: x
Resent
Ack: 701
Stop
All in order
Time
Time