第20章 运输层协议

南京大学计算机系 黄皓教授

2007年11月16日 星期五

2007年11月23日 星期二

Chapter 20. Transport Protocols

(1) Transport Mechanisms

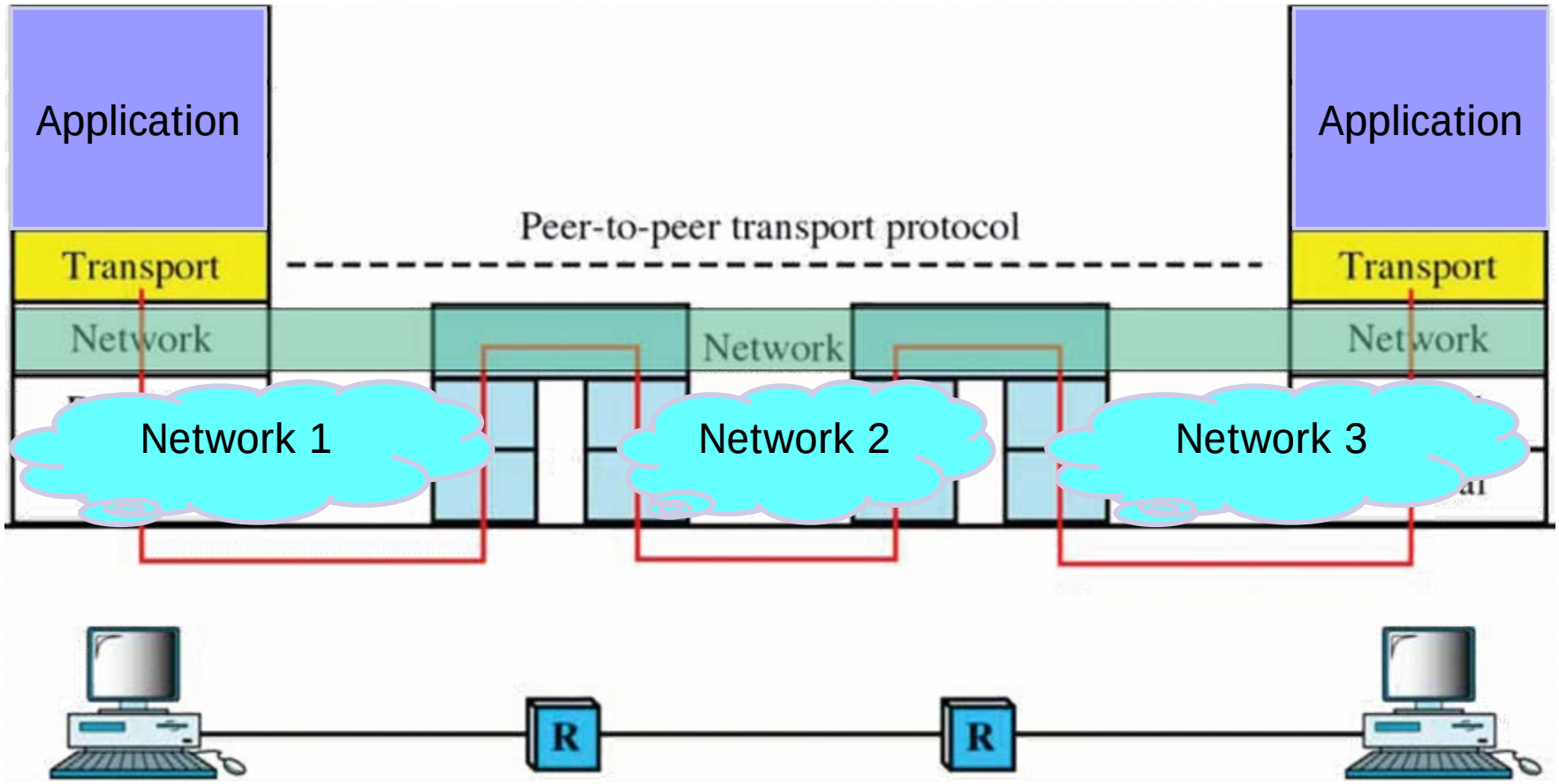
(2) TCP – Transmission Control Protocol

(3) TCP Congestion Control

(4) UDP – User Datagram Protocol

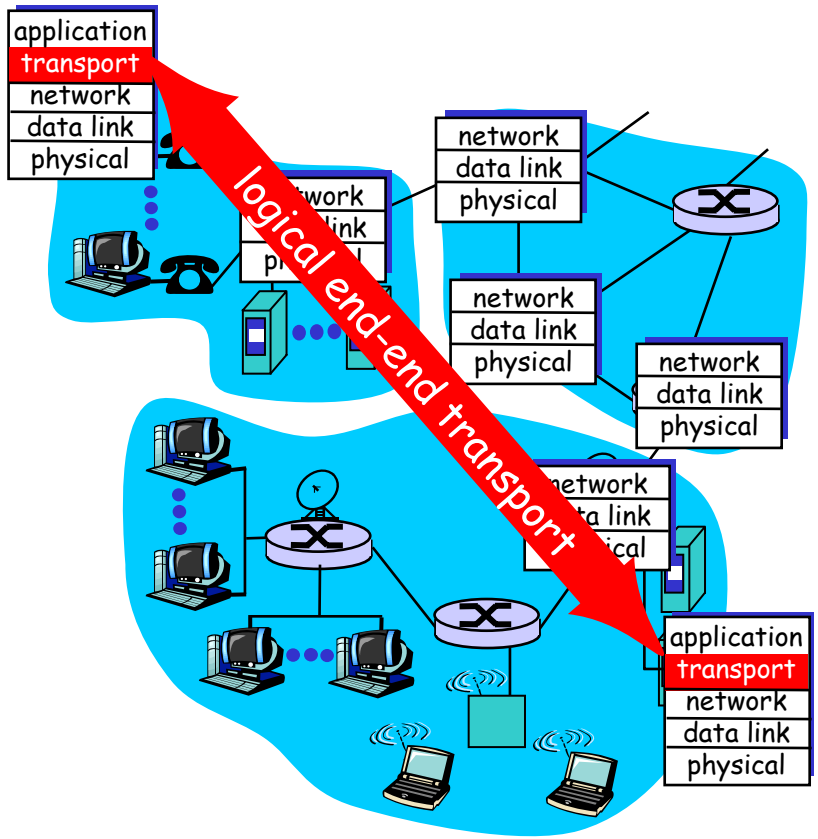
1. Transport Mechanisms

The Transport Layer



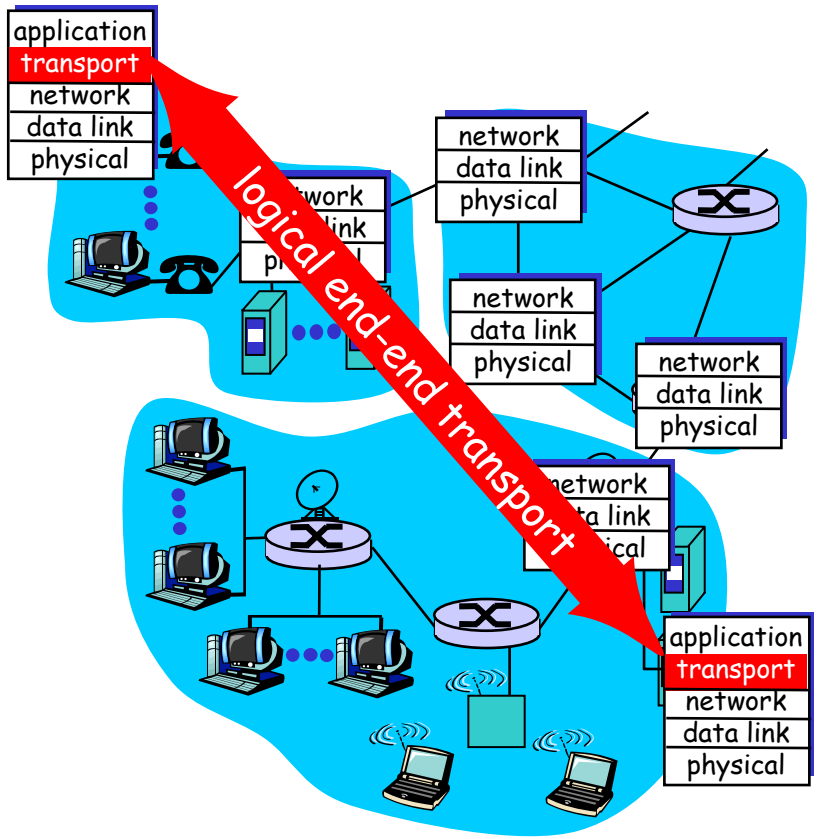
Transport Services

- Provide **logical communication** between app processes running on different hosts
- Transport protocols run in end systems
  - **Send side**: breaks app messages into **segments**, passes to network layer
  - **Receive side**: reassembles segments into messages, passes to app layer
- More than one transport protocol available to apps
  - Internet: TCP and UDP



Internet Transport-Layer Protocols

- **Reliable, in-order delivery** (TCP)
  - Connection oriented
  - Congestion control
  - Flow control
- **Unreliable, unordered delivery** (UDP)
  - No-frills extension of "best-effort" IP
- Services **not available** yet
  - Delay guarantees
  - Bandwidth guarantees



Connection Oriented Mechanisms

- **Logical connection**
  - Establishment
  - Maintenance
  - Termination
- **Reliable** communication

Reliable Sequencing Network Service

- Assume arbitrary length message delivered **in sequence**
- Assume virtually **100% reliable delivery** by network service
- Examples
  - Reliable packet switched network using **X.25**
  - Frame relay using **LAPF control** protocol
  - IEEE 802.3 using **connection oriented LLC** service

1.1 Transport Protocol built on Reliable Networks

- Issues

  (1) Addressing

  (2) Multiplexing

  (3) Flow Control

  (4) Connection establishment and termination

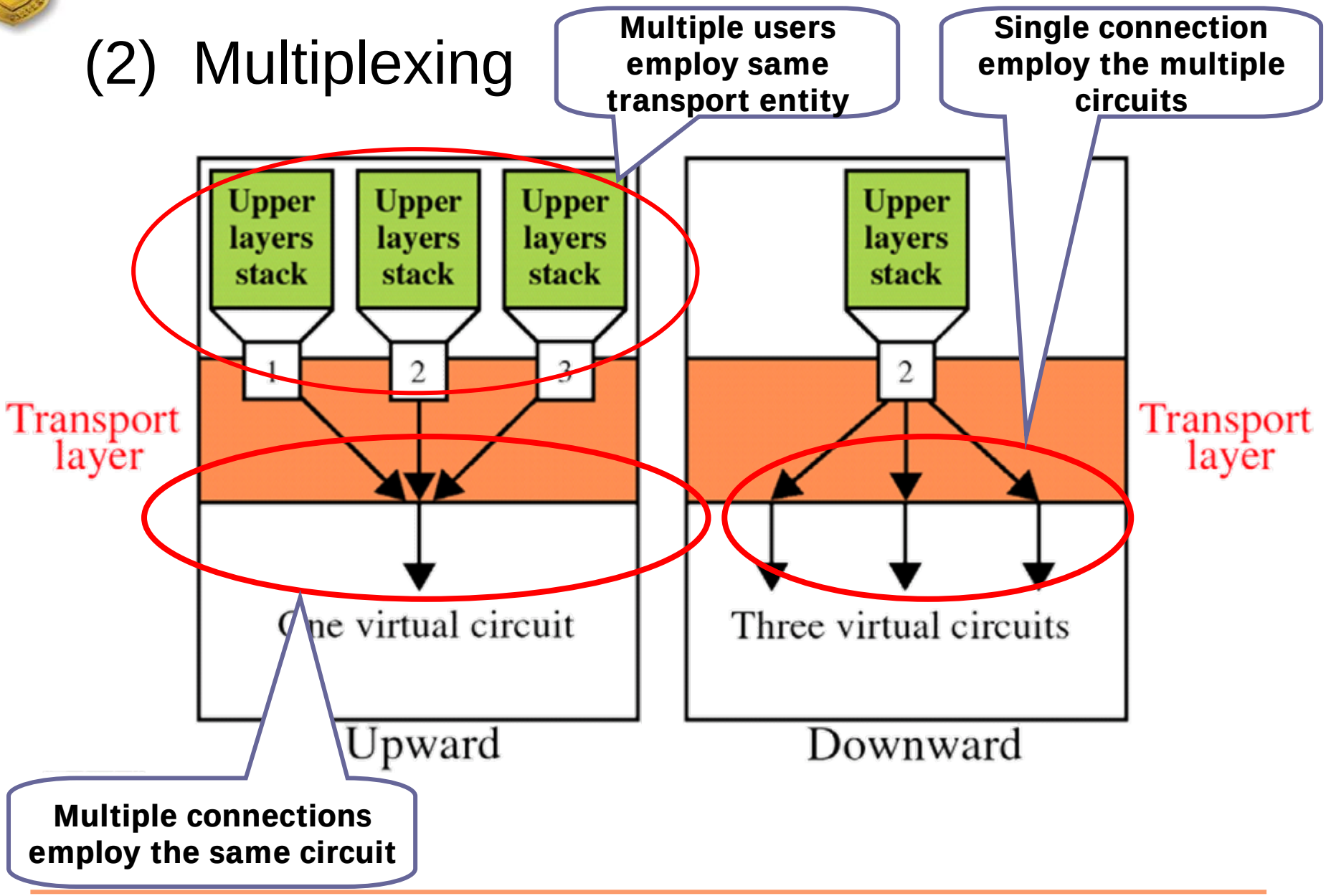
（1） Addressing

- **Application processes** on end systems
  - **Process identification**
    - SAP on Transport entity, represents a particular transport service (TS) user
    - e.g. port number on TCP
  - **Transport entity identification**
    - Generally only one per host
    - If more than one, then usually one of each type
    - e.g. specify transport protocol (TCP, UDP)
  - **Host address**
    - The attached end system
    - In an internet, a global internet address
  - Network number
- Internet TCP/UDP addressing
  - <HostIP, Port>, called a **socket**

Finding Addresses

- How TS user get to know each other
  - **Target host**
  - Target port: use conventions
- 4 methods can be used
  - Know address ahead of time
    - e.g. collection of network device stats
  - **Well known addresses**
  - **Name server**, directory service
  - Sending process request to known address
    - Create a TS user on target host

(2) Multiplexing



Multiplexing / Demultiplexing

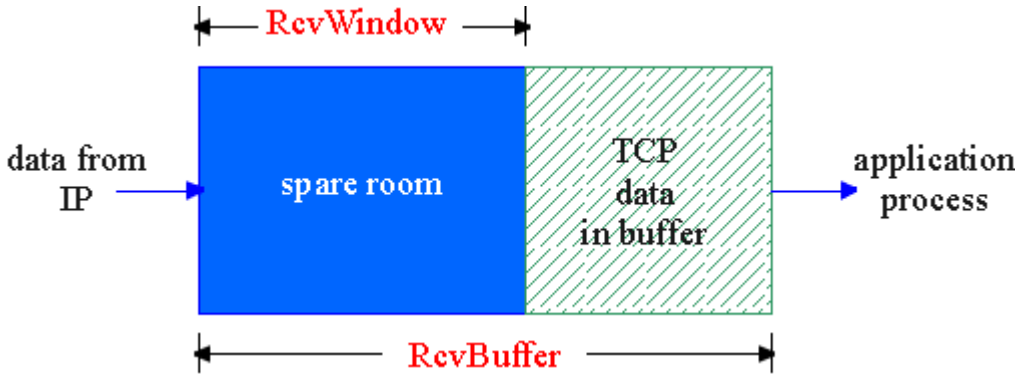
- **Multiplexing at send host**
  - Gathering data from multiple ports, enveloping data with specific header
  - Encapsulates with src and dest IP addresses
  - Sends datagrams through src–dest IP link
- **Demultiplexing at receive host**
  - Each datagram routed by src and dest IP addresses
  - Each datagram carries 1 transport-layer segment
  - Each segment has src and dest port number for application process
- Host uses src and dest **sockets pair** to direct segment from / to appropriate processes

(3) Flow Control

- Basic concept
  - Sender won't overflow receiver's buffer by transmitting **too much too fast**
  - The receiving TS user can not keep up
    - Transport buffer may overflow
  - The receiving transport entity can not keep up
    - Network buffer may overflow
- **Speed-matching** service
  - Matching the send rate to the receiving app's drain rate

(3) Flow Control — Receive Buffer

- The receive side of TS connection has a receive buffer



- App process may be slow at reading from buffer

(3) Flow Control

① **Do nothing**

- Segments that overflow are discarded
- Sending transport entity will fail to get *ACK* and retransmit

Problem:

- Further adding to incoming data
- Reliable network now **becoming unreliable**

(3) Flow Control

② Refuse further segments

- Aggregate flow of multiplexed connections are **controlled in a whole**

Problem:

- Clumsy, add much communication of flow control info
- Other connections are sacrificed due to a greedy app user

(3) Flow Control

③ Fixed sliding window

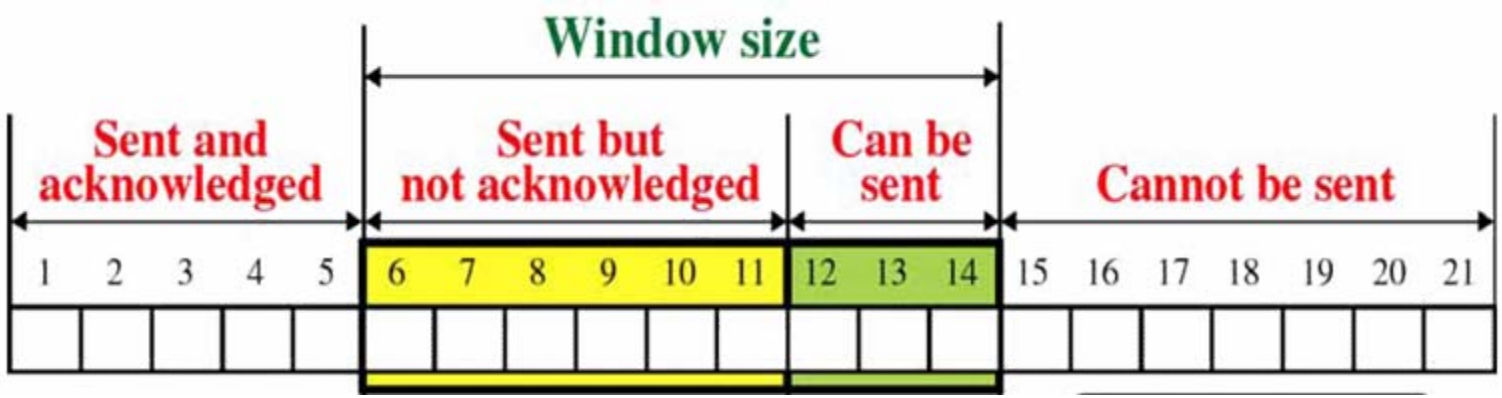
- Works well on reliable direct links

Problem on non-reliable network:

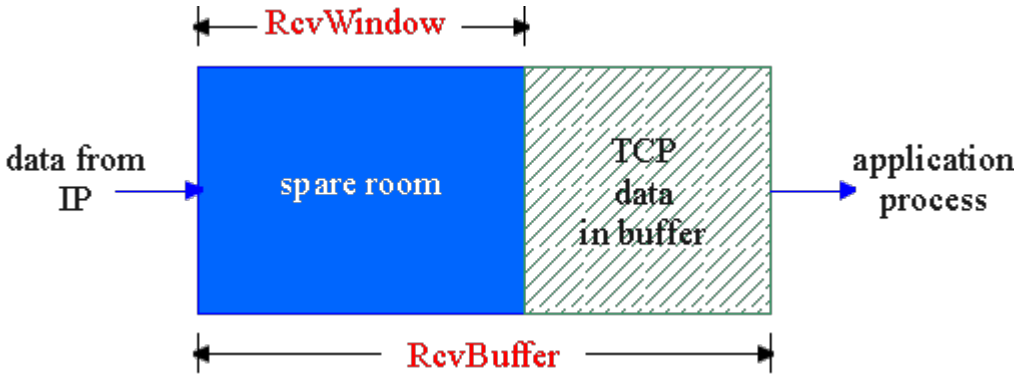
- Failure to receive *ACK* is taken as flow control indication
- **Can not distinguish between lost segment and flow control**
- Not flexible for congestion control mandated in networks

(3) Flow Control

④ TCP uses credit scheme



(3) Flow Control — Credit Scheme (1)



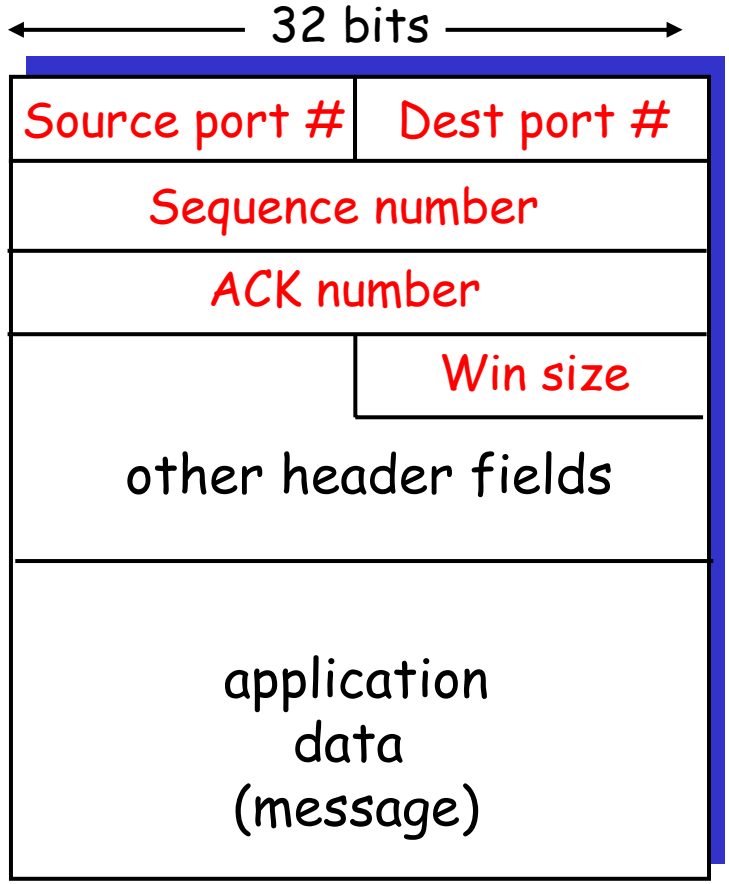
- **Spare room** in receive buffer

  RcvWindow = RcvBuffer -[LastByteRcvd - LastByteRead]

- Rcvr advertises spare room (credits) by including value of RcvWindow in segments
- Sender limits unACKed data to RcvWindow

(3) Flow Control — Credit Scheme (2)

- Greater control on reliable network
- More effective on unreliable network
- **Decouples flow control from *ACK***
  - May *ACK* without granting credit
- Each octet has a sequence number
- Each transport segment has *seq number*, *ack number* and *window size* in header

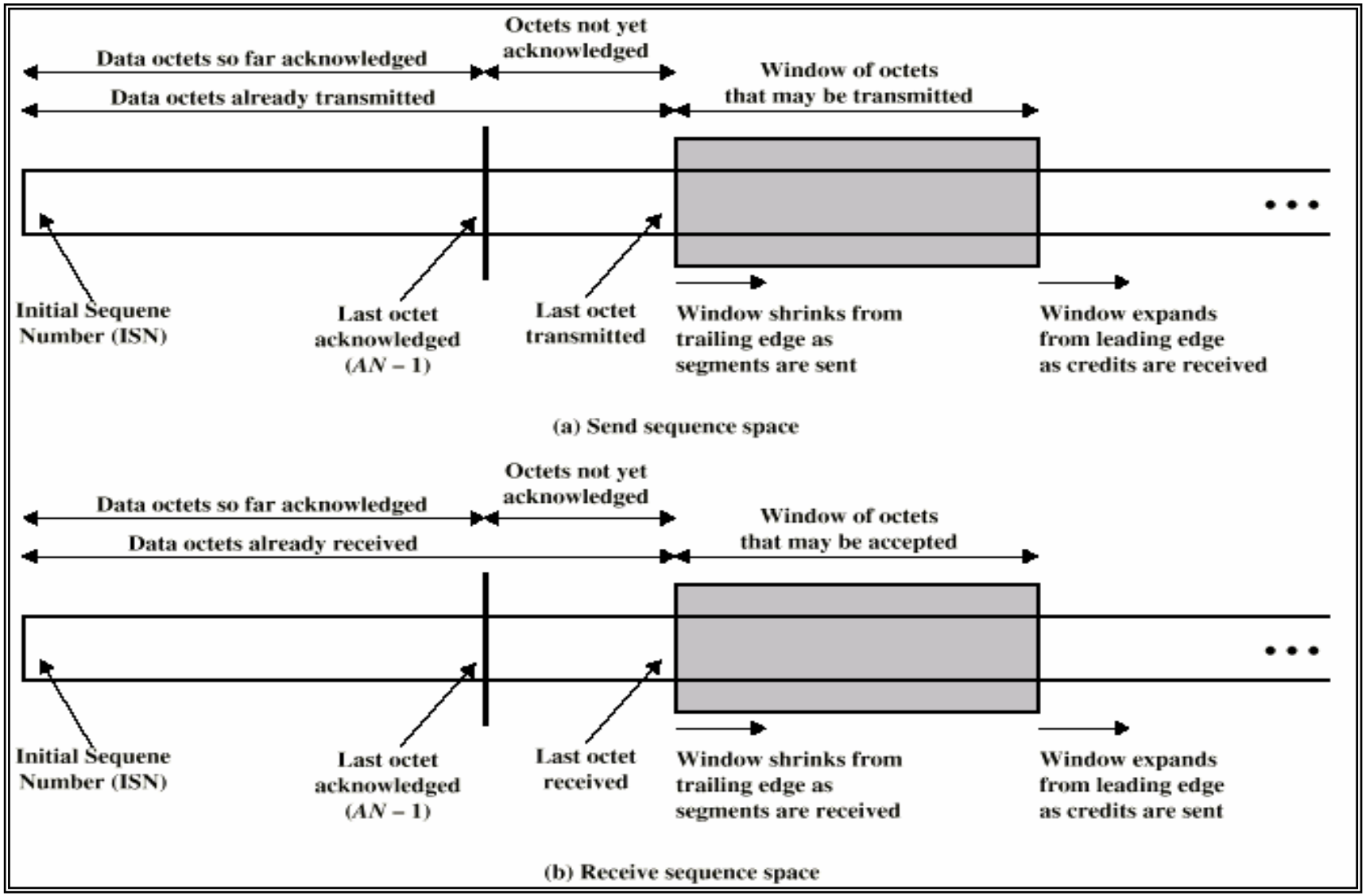


(3) Flow Control — Use of Header Fields

- When **sending a segment**
  - *seq number* (SN) is that of first octet in segment
  - *ACK* includes AN=i, W=j
    - All octets through SN=i-1 **acknowledged**
      - Next expected octet is i
    - Permission to send **additional window** of W=j octets
      - i.e. octets from i to i+j-1

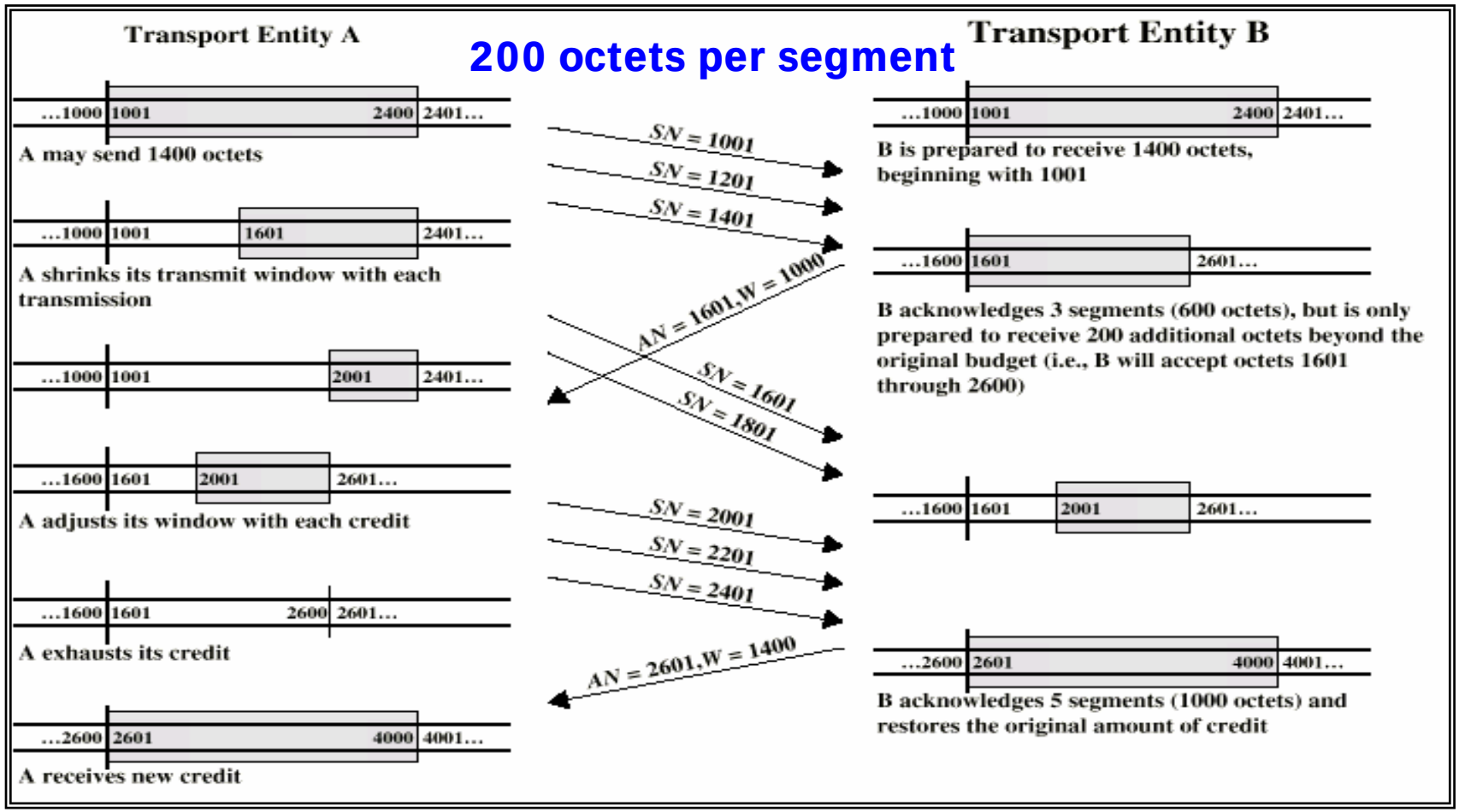
(3) Flow Control

Sending and Receiving Windows



(3) Flow Control

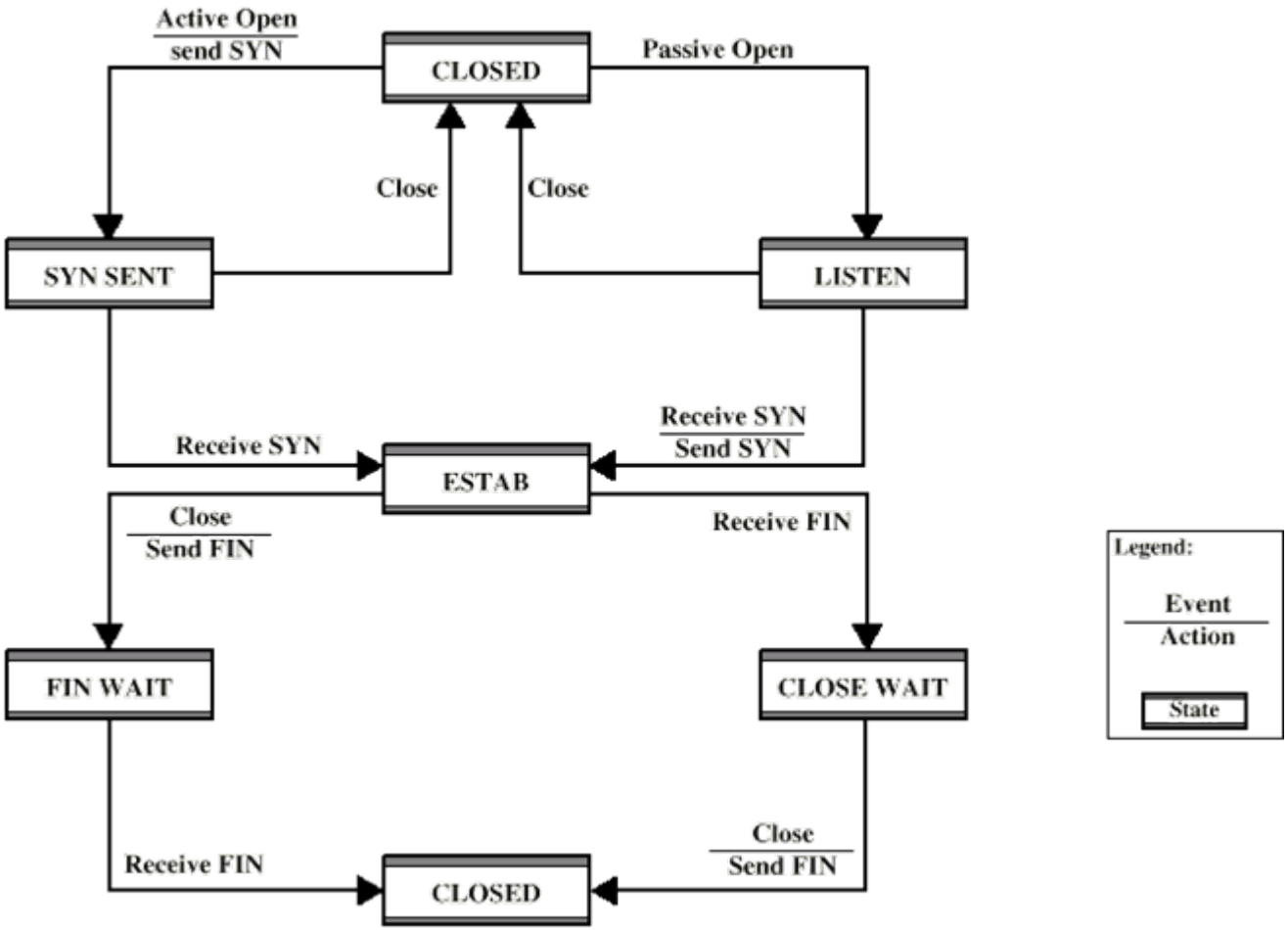
- Credit Allocation Procedure



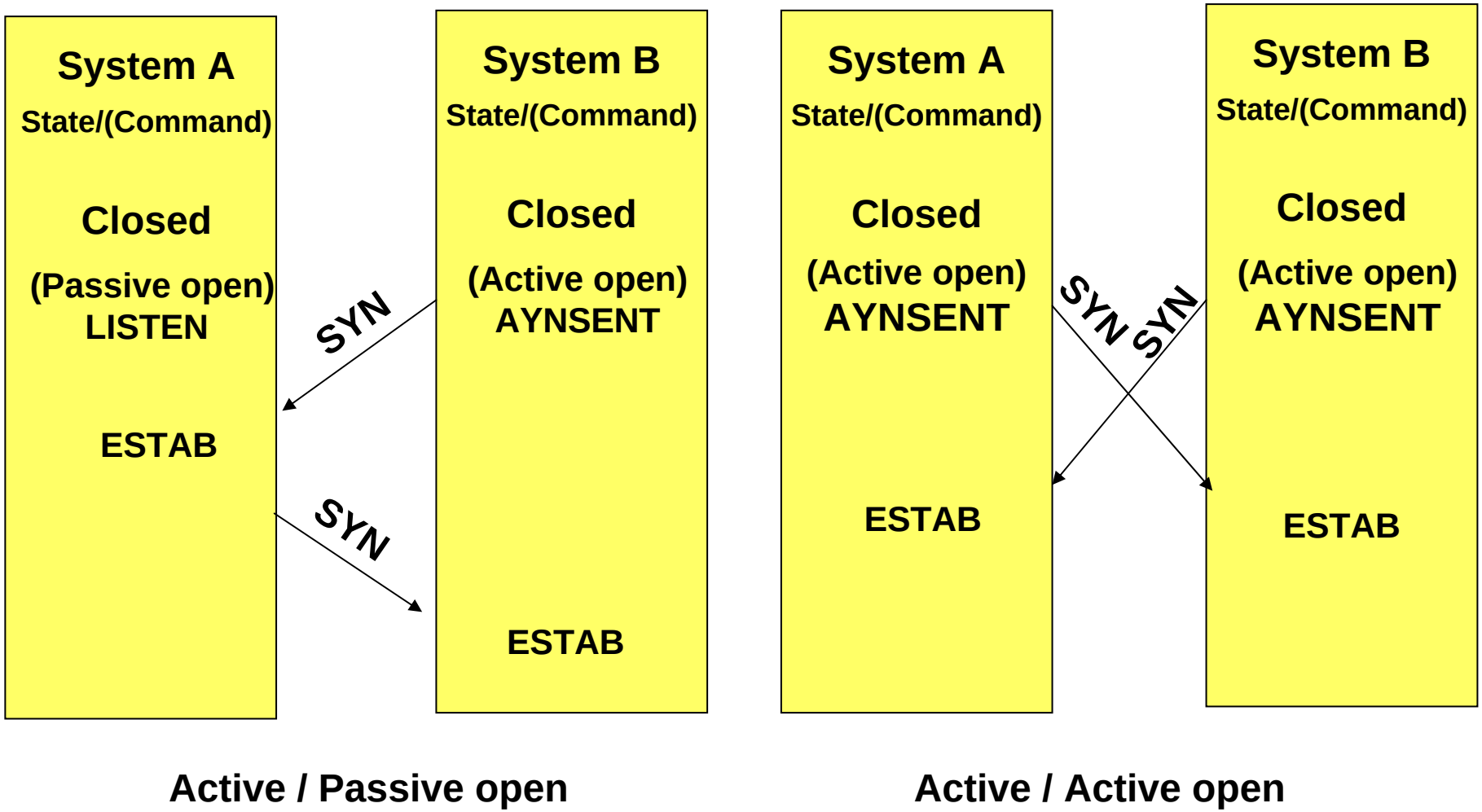
(4) Connection Establishment and Termination

- 2 ends establish **connection** before exchanging data segments
  - Allow each end to know the other exists
  - Negotiation of **optional parameters**
    - e.g. initial *seq numbers*, max segment size, max window size, *QOS*
  - Allocation of transport entity resources
    - e.g. buffers
- Gets mutual agreement

Simple Connection State Diagram(1116)



Connection Establishment



Handle Pending Request

- SYN comes while requested TS user is not listening
  - Reject with RST (Reset)
  - Queue request until matching open issued
  - Signal TS user to notify of pending request
    - Just accept without passive open

Termination

- Either or both sides issue **terminate**
- Reach **mutual agreement**
- **Abrupt termination**
  - Pending segments from other end may lost
- **Graceful termination**
  - All outstanding data is transmitted from both sides
  - Both sides agree to terminate

Side Initiating Termination

- TS user issues *Close request*
- Transport entity sends FIN, **requesting termination**
- Connection placed in FIN Wait state
  - Continues to accept data and deliver data to user
  - Not sends any more data
- When FIN received, inform user and close connection

Side Not Initiating Termination

- FIN received, Inform TS user
  - **No more data come from the other end**
- Place connection in *CLOSE Wait* state
  - Continue to accept data from TS user and transmit it
- TS user issues *CLOSE* primitive
  - Transport entity sends FIN
  - Connection closed

1.2 Transport Protocol on Unreliable Network

- Unreliable Network Service
  - Segments may **get lost**
  - Segments may arrive **out of order**
  - Examples
    - Internet using IP
    - Frame relay using LAPF core
    - IEEE 802.3 using unacknowledged connectionless LLC

Problems

1. Ordered Delivery
2. Retransmission strategy
3. Duplication detection
4. Flow control
5. Connection establishment
6. Connection termination
7. Crash recovery

(1) Ordered Delivery

- Problem
  - Segments may arrive out of order
- Handle
  - **Number segments sequentially**
  - TCP numbers each octet sequentially
  - Segments are numbered by the first octet number in the segment

(2) Retransmission Strategy

- Problem
  - Segment damaged in transit
  - Segment dropped due to buffer overflow at router
  - Sender does not know of failure
- Handle
  - Receiver: acknowledge successful receipt
  - Can use cumulative acknowledgement
  - Sender: waiting Timer for *ACK* timeout triggers re-transmission

Re-transmission Timer

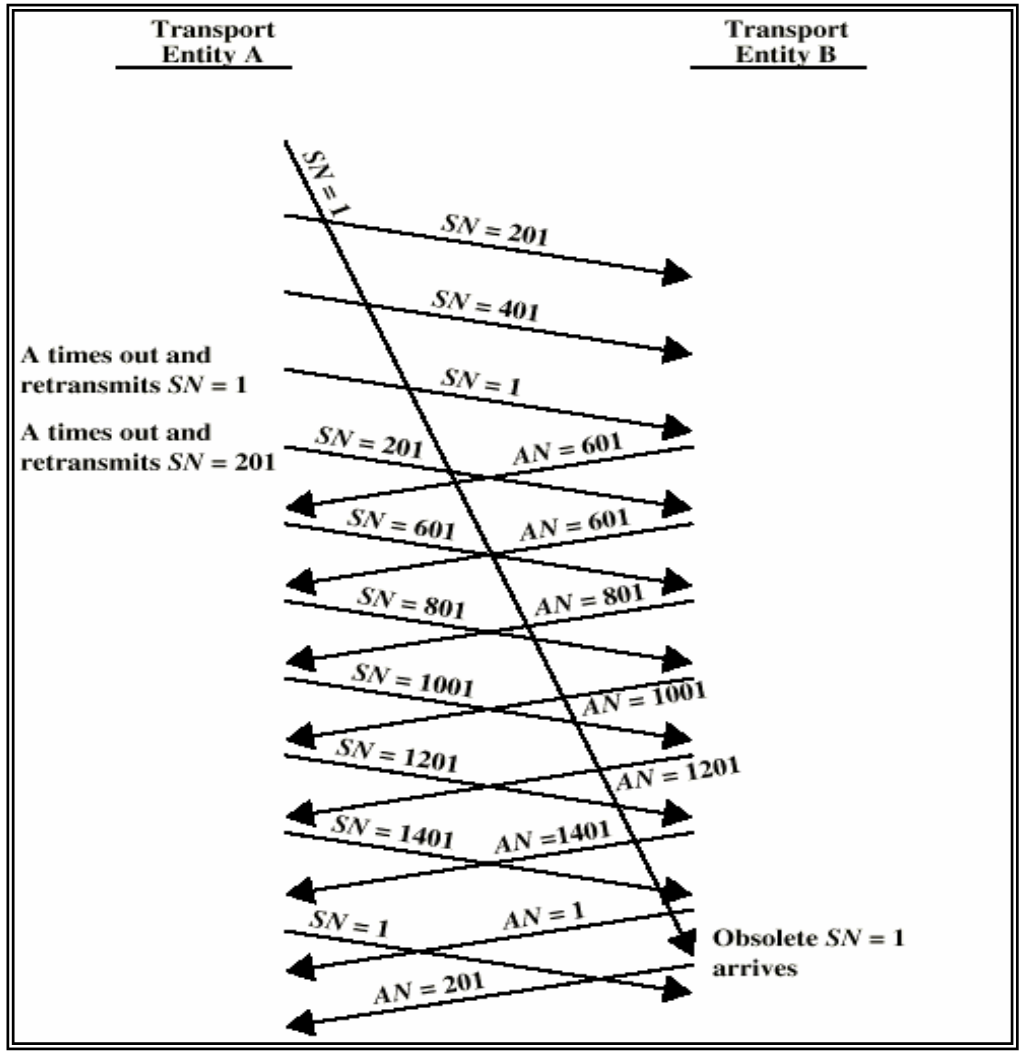
- **Set timer value**
  - Based on understanding of network behavior
  - Should adapt to **changing network conditions**
    - Fixed timer is not suitable
    - Too small leads to unnecessary re-transmissions
    - Too large and response to lost segments is slow
  - Can be set a bit longer than round-trip time
- Problems in **round-trip time calculation**
  - Receiver may not *ACK* immediately
  - Sender can not distinguish between *ACK* of original segment and re-transmitted segment
  - Network conditions may change suddenly

(3) Duplication Detection

- If re-transmission Timer timeout, sender re-transmits segment
- If segment just delayed, receiver must **recognize duplicates**
- **Duplicate received within a connection**
  - Receiver assumes *ACK* lost and re-transmits *ACKs*
  - Sender must not get confused with multiple *ACKs*
  - Space of *seq number* should be large enough to not cycle within maximum life of segment
- **Duplicate received after connection closed**

Incorrect Duplicate Detection (1)

- *Seq number* cycled within life of a segment.
- Suppose the sequence space is **1600**.



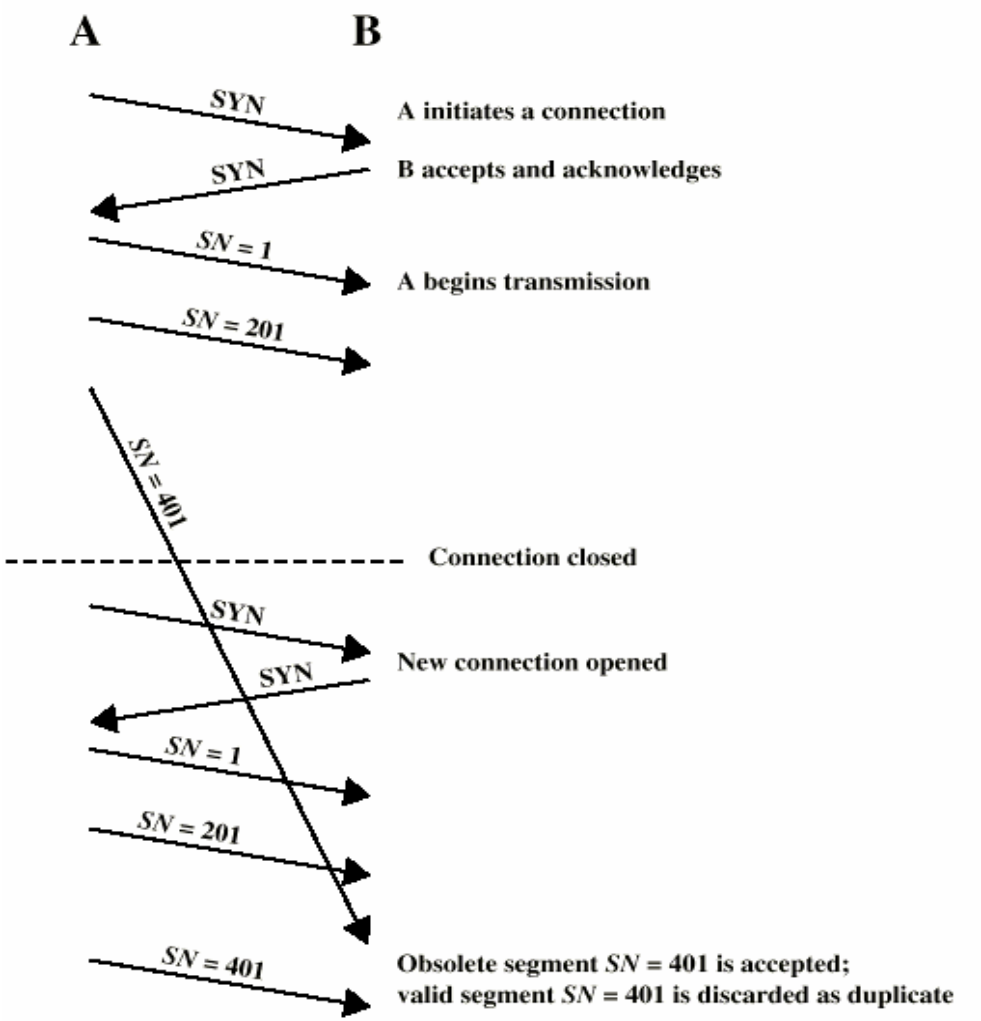
(4) Flow Control

- **Credit allocation deadlock**
  - Segment with AN=i, W=0 closing rcv-window
  - Should send AN=i, W=j to reopen, but this maybe lost
  - Now sender thinks window is closed, receiver thinks it is open and wait
- Handle
  - Use **window timer**
  - If timer expires without any receiving, send something
  - Could be re-transmission of previous segment

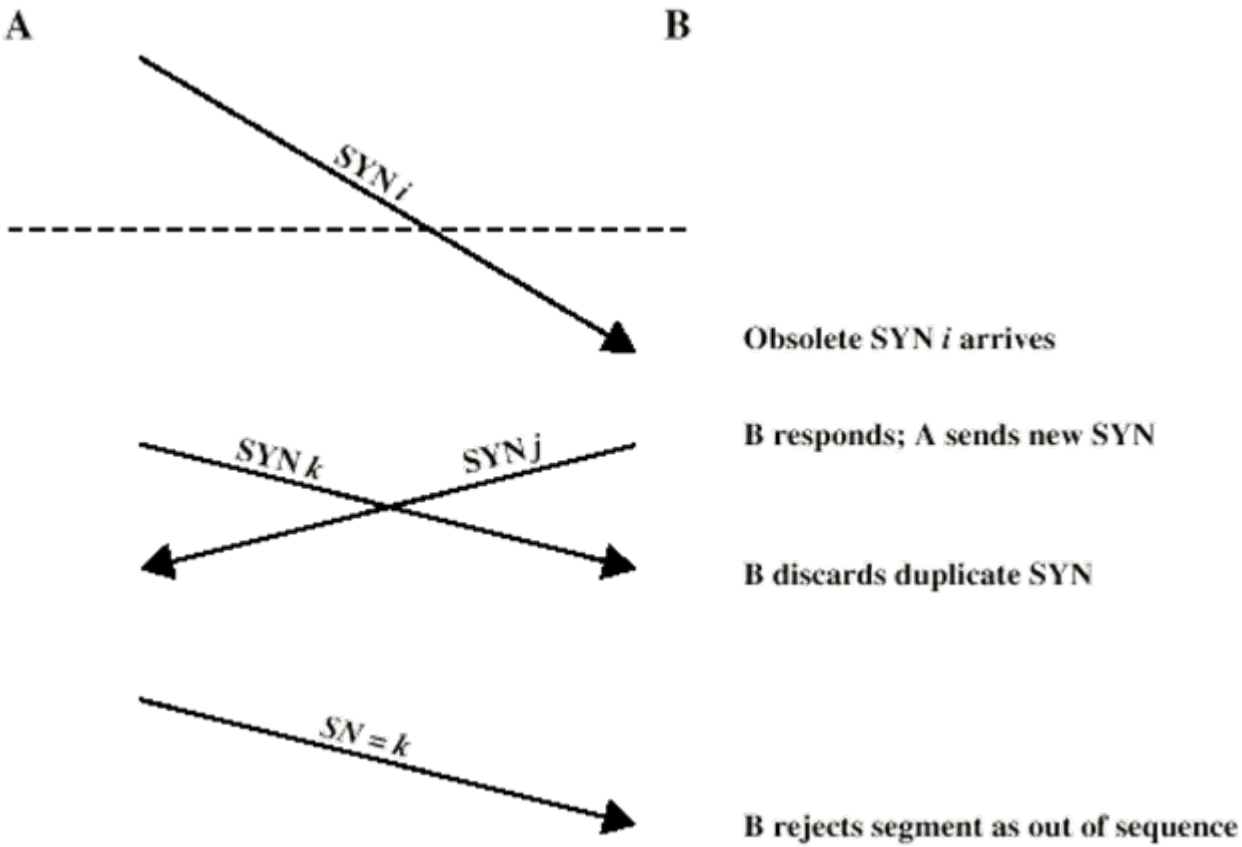
(5) Connection Establishment

- **2-way handshake**
  - A sends SYN, B replies with SYN
  - Lost SYNs handled by re-transmission
  - Ignore duplicate SYNs once connected
- Problem
  - How to recognize **slipped segments from old connection**
  - How to recognize duplicated **obsolete SYN**
- Handle
  - SYN with start seq number i far removed from previous connection
  - ACK SYN, use 3-way handshake

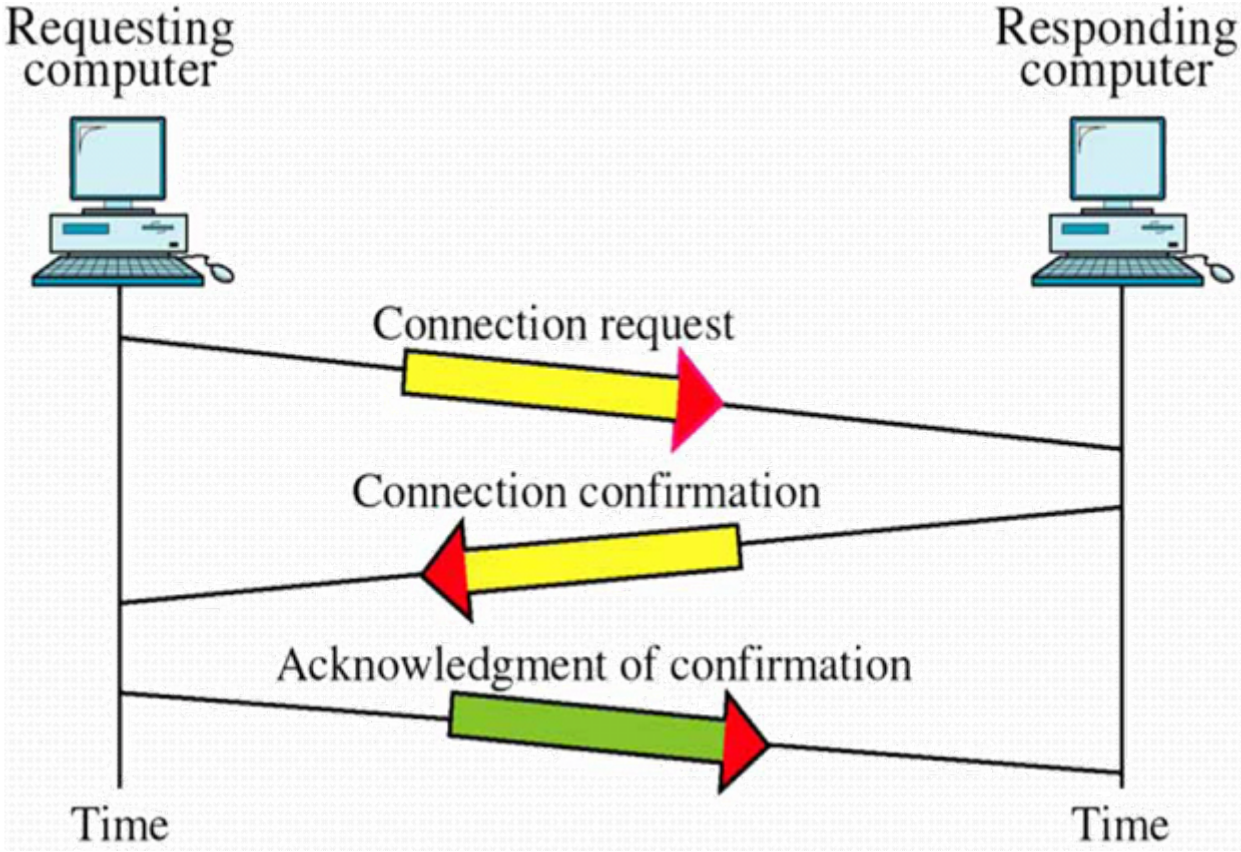
2-Way Handshake: Slipped Data Segment



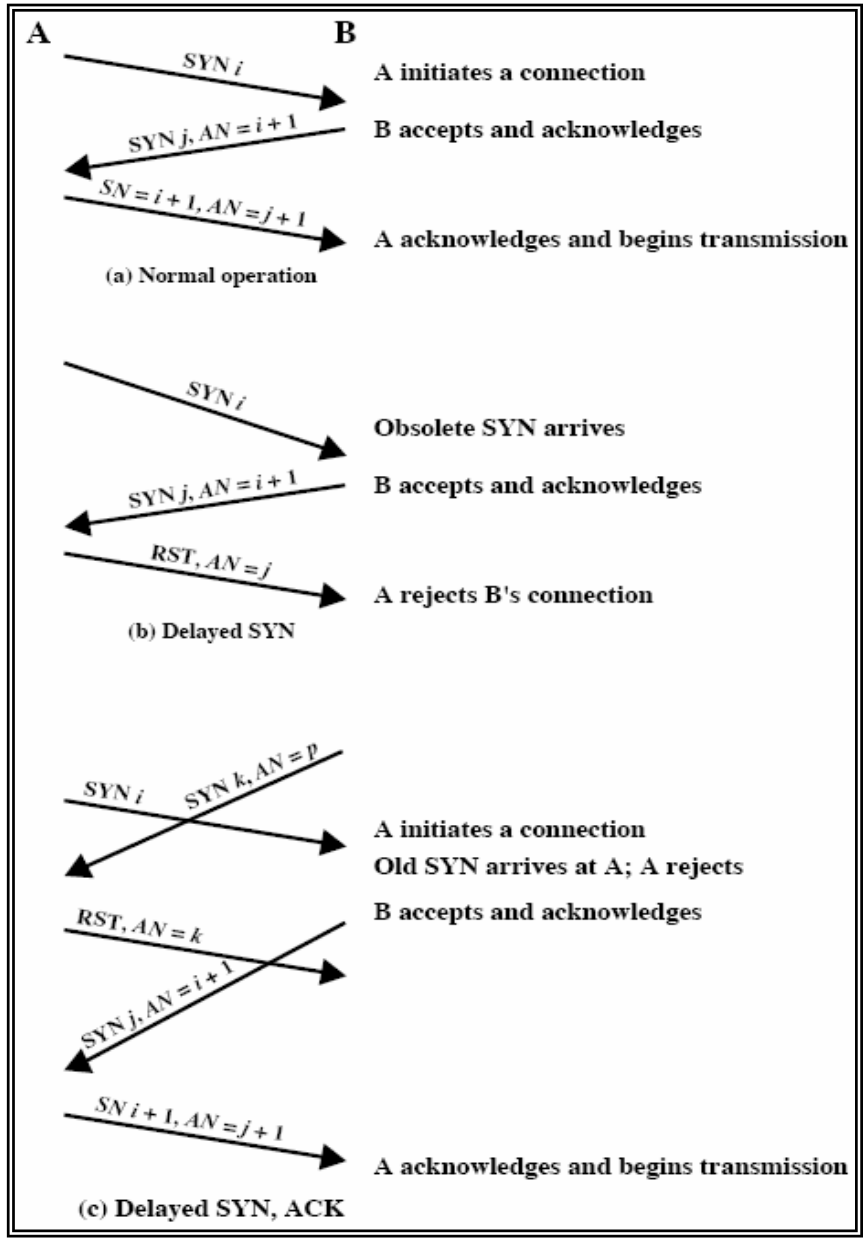
2-Way Handshake: Obsolete SYN



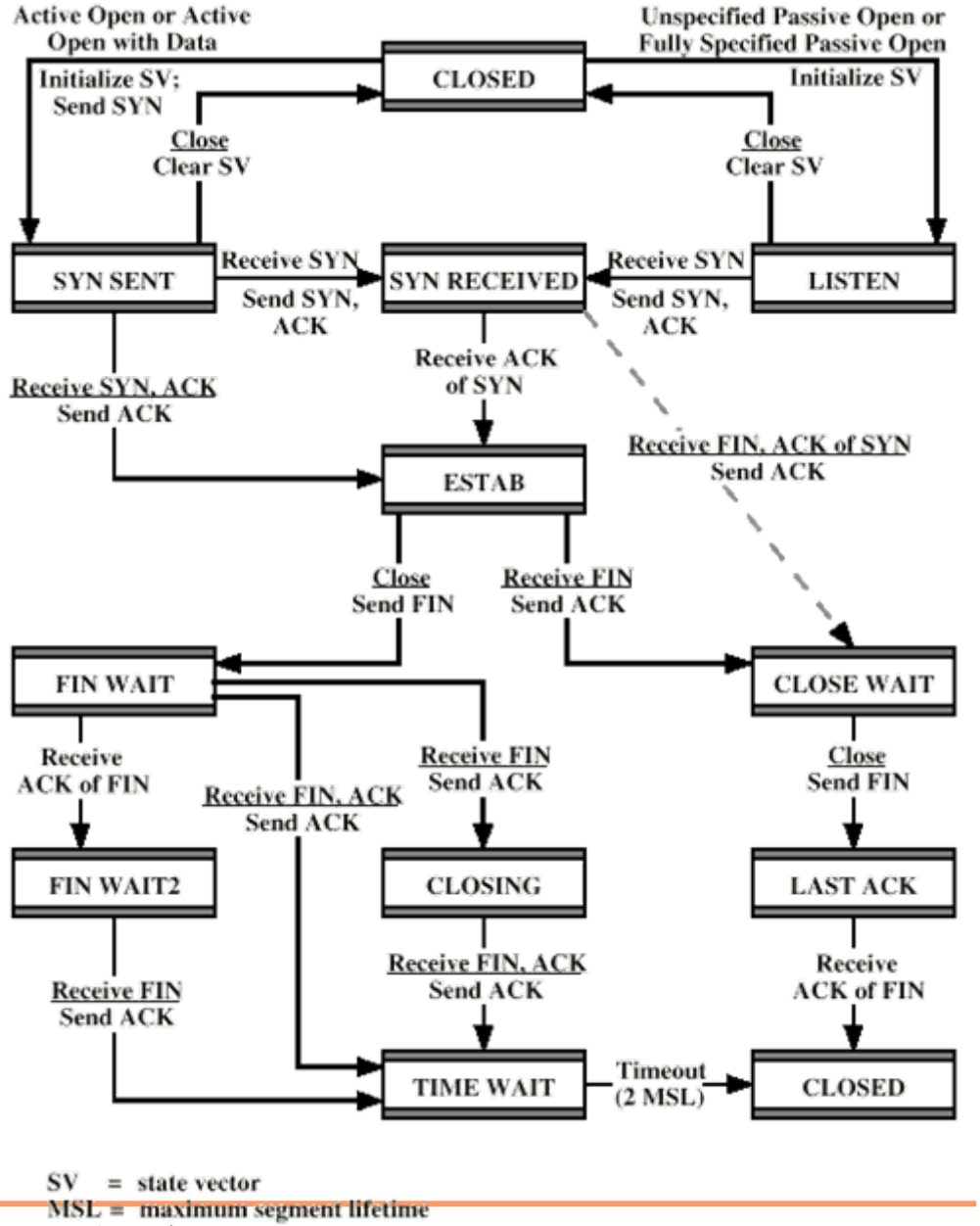
3-Way Handshake



3-Way Handshake: Examples

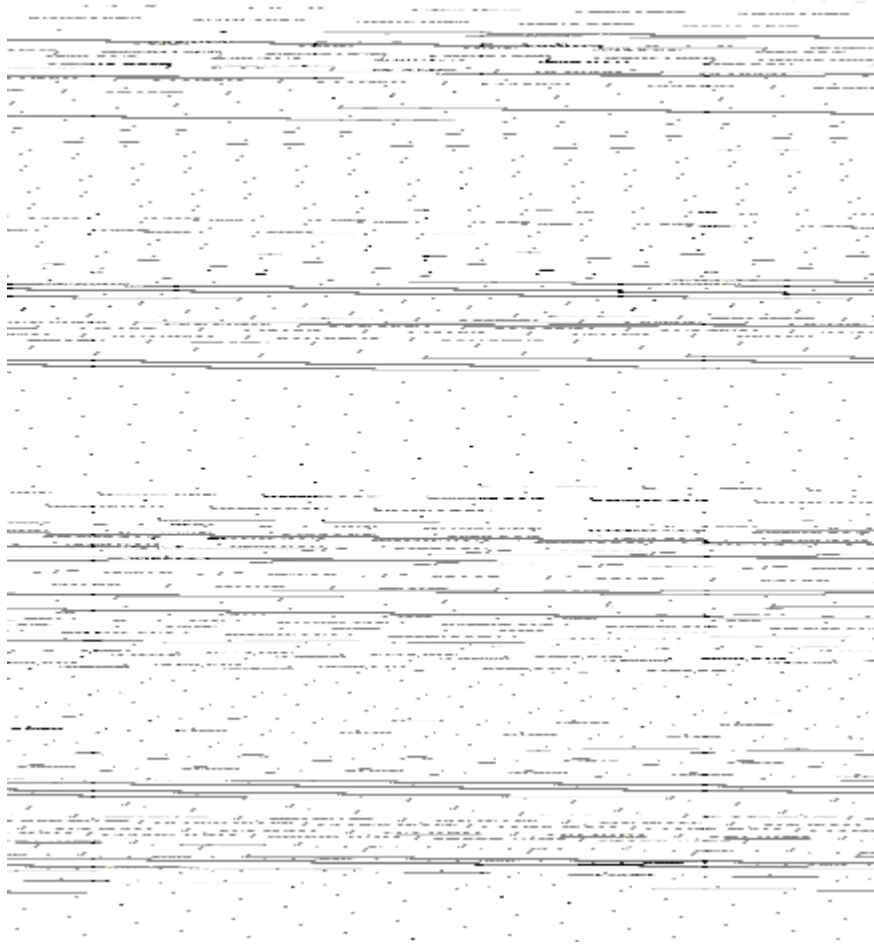


3-Way Handshake: State Diagram

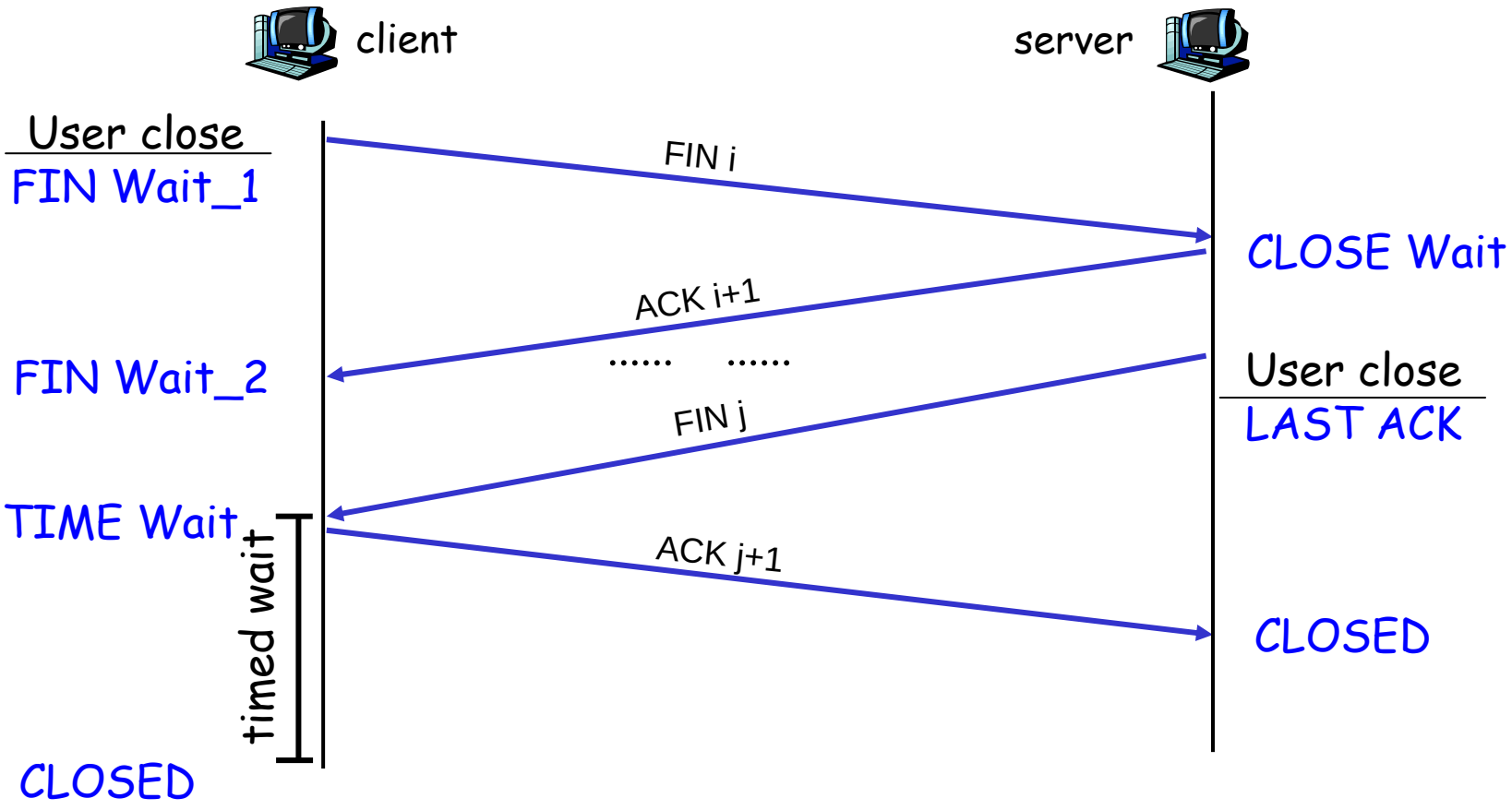


(6) Connection Termination

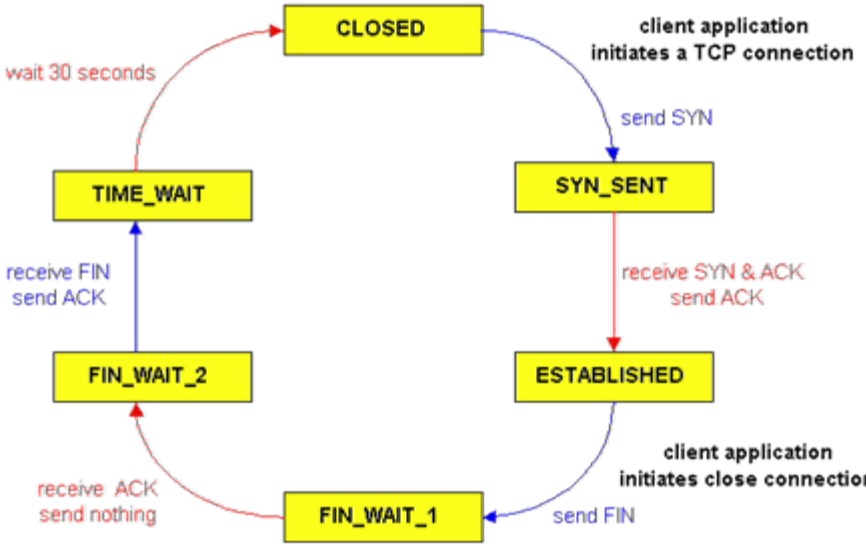
- A problem with 2-way termination
  - Entity in *CLOSE Wait* state sends last data segment, followed by FIN
  - FIN arrives before last data segment
  - Receiver accepts FIN and closes connection
  - Now last data segment lost
- Handle
  - Associate sequence number with FIN
  - Receiver waits for all segments before FIN seq number
  - *ACK* FIN, use 3-way termination



Graceful Close

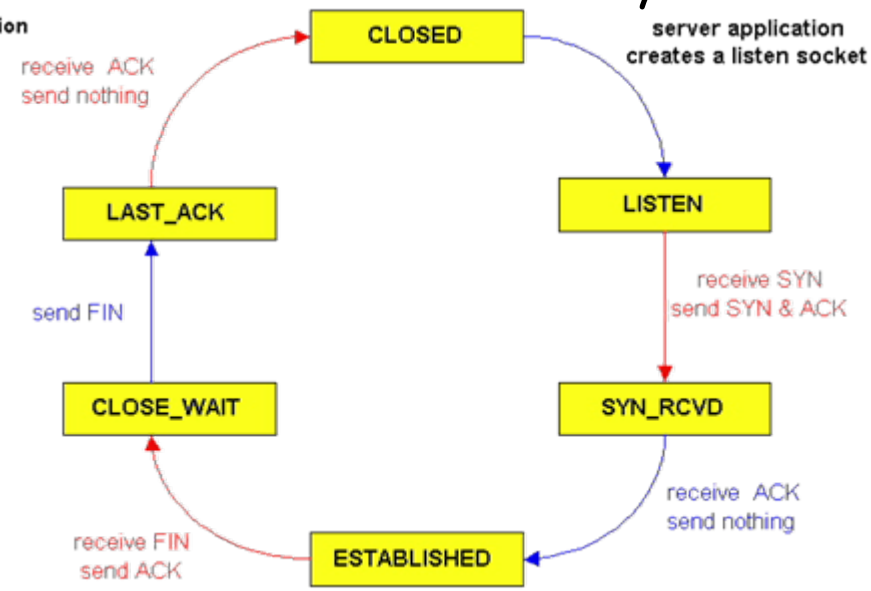


State Diagram: Client and Server



TCP client lifecycle

TCP server lifecycle



(7) Crash Recovery

- Problem
  - If a side restarts, all state info is lost
  - Connection is half open now
    - Side that did not crash still thinks it is connected
- Handle
  - Close connection using Persistence Timer
    - Wait for *ACK* for (timeout) × (number of retries)
    - When expired, close connection and inform user
  - Restarted side **sends RST i** in response to any i segment arriving
    - Other side verifies RST i, then closes connection
- Restarted user can reconnect immediately
  - May take the risk of slipped segments from just crashed connection

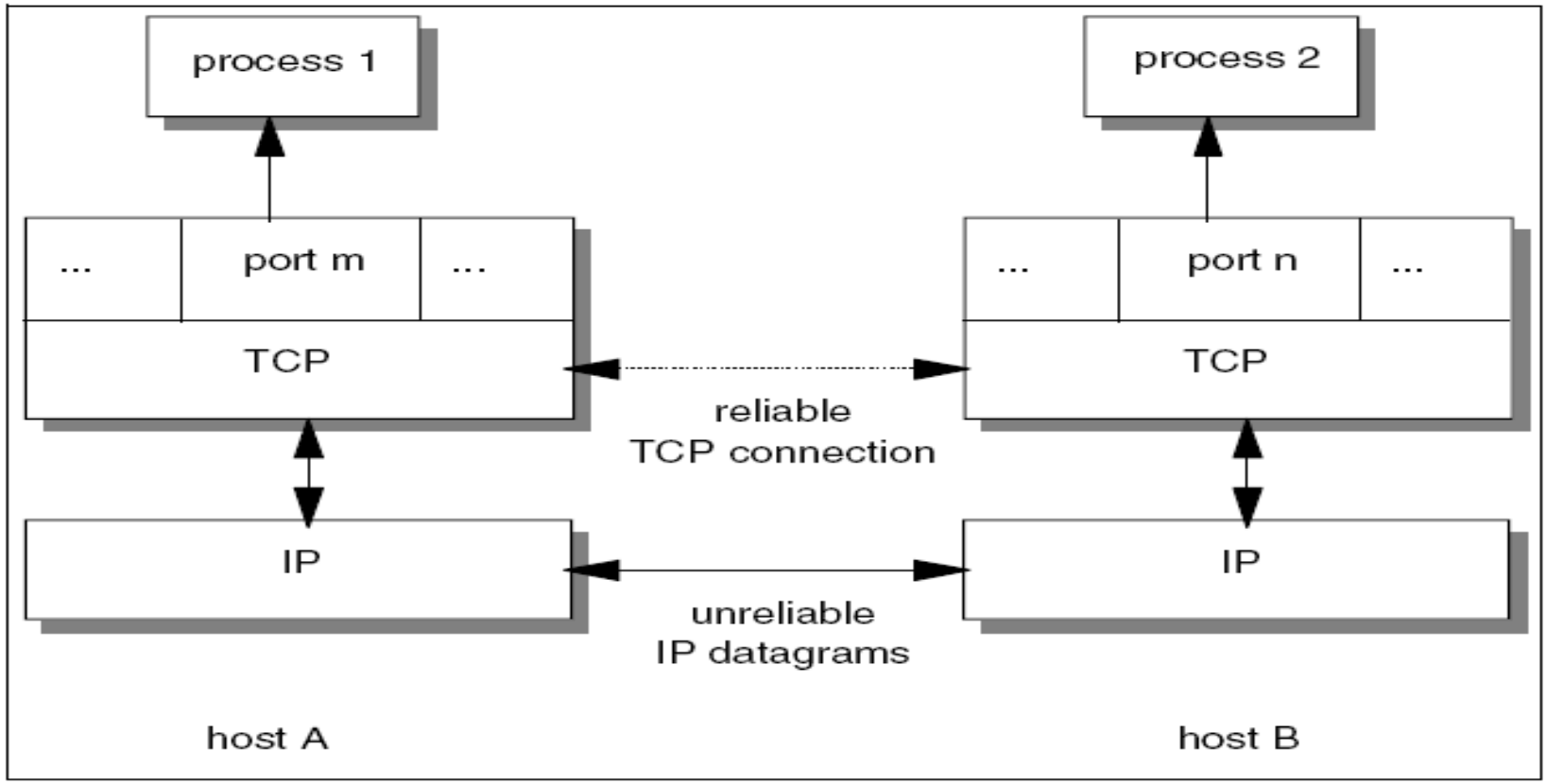
2. Transmission Control Protocol

Transmission Control Protocol

1. TCP Services
2. TCP Format
3. TCP mechanisms
4. Implementation Policy Options

TCP Services

- Reliable communication between pairs of processes
  - Across variety of reliable and unreliable networks and internets



TCP Services

- Stream Data Transfer
  - From the application's viewpoint, TCP transfers a contiguous stream of bytes through the network.
  - The application does not have to bother with chopping the data into basic blocks or datagrams.
  - TCP does this by grouping the bytes in TCP segments, which are passed to IP for transmission to the destination.
  - Also, TCP itself decides how to segment the data and it can forward the data at its own convenience.
  - **Sometimes, an application needs to be sure that all the data passed to TCP has actually been transmitted to the destination.**
    - For that reason, a push function is defined.
    - It will push all remaining TCP segments still in storage to the destination host.
    - The normal close connection function also pushes the data to the destination.

TCP Services

- Reliability
  - TCP assigns a sequence number to each byte transmitted and expects a positive acknowledgment (ACK) from the receiving TCP.
  - If the ACK is not received within a timeout interval, the data is retransmitted.
  - Since the data is transmitted in blocks (TCP segments), only the sequence number of the first data byte in the segment is sent to the destination host.
  - The receiving TCP uses the sequence numbers to rearrange the segments when they arrive out of order, and to eliminate duplicate segments.

TCP Services

- Flow Control
  - The receiving TCP, when sending an ACK back to the sender, also indicates to the sender the number of bytes it can receive beyond the last received TCP segment, without causing overrun and overflow in its internal buffers.
- Multiplexing
- Logical Connections
- Full Duplex

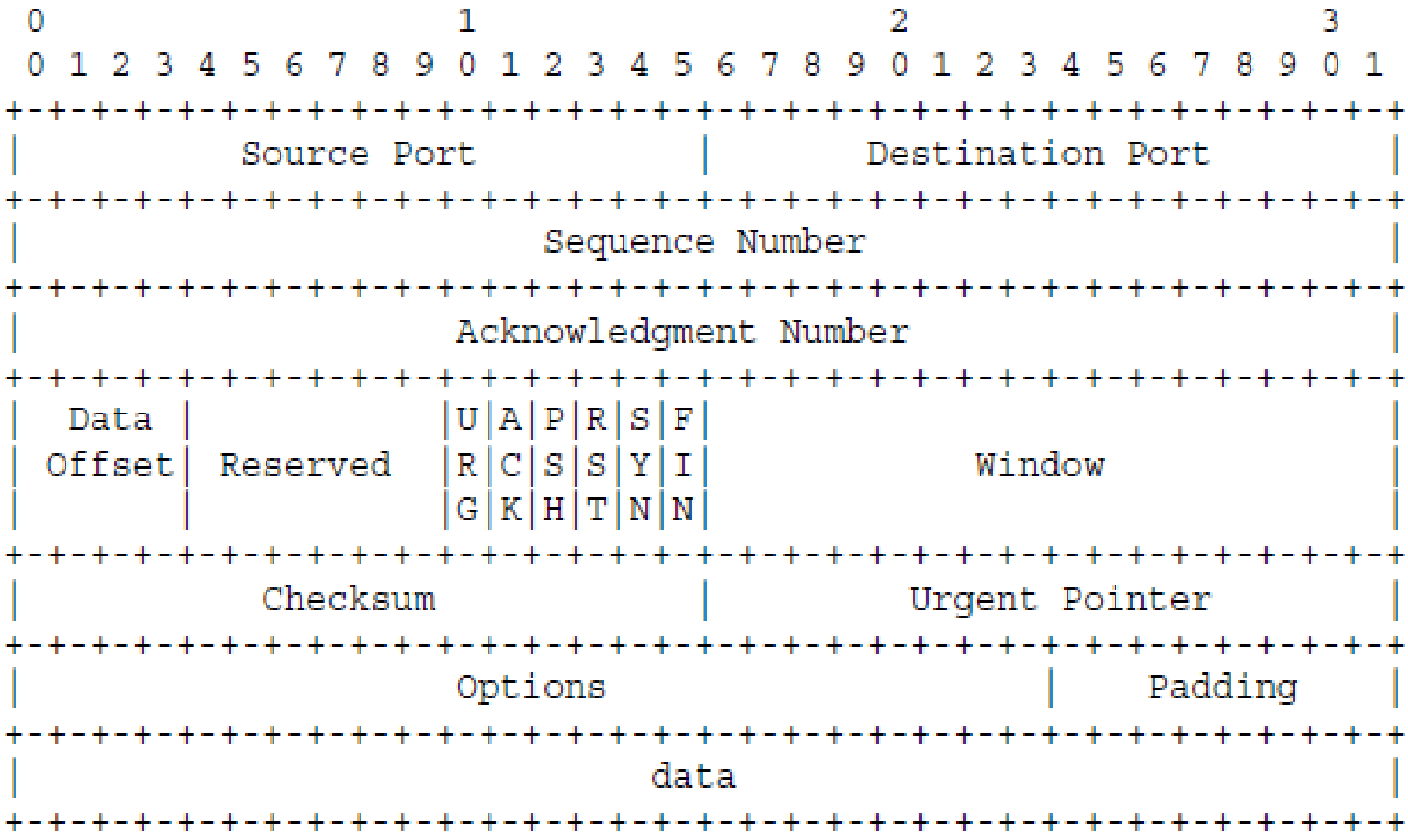
TCP Service Request Primitives

Primitive	Parameters	Description
Unspecified Passive Open	source-port, [timeout], [timeout-action], [precedence], [security-range]	Listen for connection attempt at specified security and precedence from any remote destination.
Fully Specified Passive Open	source-port, destination-port, destination-address, [timeout], [timeout-action], [precedence], [security-range]	Listen for connection attempt at specified security and precedence from specified destination.
Active Open	source-port, destination-port, destination-address, [timeout], [timeout-action], [precedence], [security]	Request connection at a particular security and precedence to a specified destination.
Active Open with Data	source-port, destination-port, destination-address, [timeout], [timeout-action], [precedence], [security], data, data-length, PUSH-flag, URGENT-flag	Request connection at a particular security and precedence to a specified destination and transmit data with the request
Send	local-connection-name, data, data-length, PUSH-flag, URGENT-flag, [timeout], [timeout-action]	Transfer data across named connection
Allocate	local-connection-name, data-length	Issue incremental allocation for receive data to TCP
Close	local-connection-name	Close connection gracefully
Abort	local-connection-name	Close connection abruptly
Status	local-connection-name	Query connection status

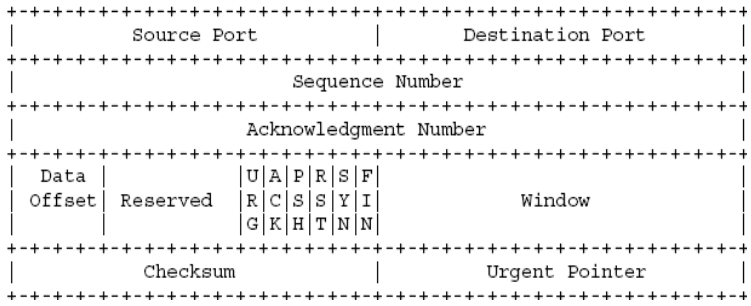
TCP Service Response Primitives

Primitive	Parameters	Description
Open ID	local-connection-name, source-port, destination-port*, destination-address*,	Informs TCP user of connection name assigned to pending connection requested in an Open primitive
Open Failure	local-connection-name	Reports failure of an Active Open request
Open Success	local-connection-name	Reports completion of pending Open request
Deliver	local-connection-name, data, data-length, URGENT-flag	Reports arrival of data
Closing	local-connection-name	Reports that remote TCP user has issued a Close and that all data sent by remote user has been delivered
Terminate	local-connection-name, description	Reports that the connection has been terminated; a description of the reason for termination is provided
Status Response	local-connection-name, source-port, source-address, destination-port, destination-address, connection-state, receive-window, send-window, amount-awaiting-ACK, amount-awaiting-receipt, urgent-state, precedence, security, timeout	Reports current status of connection
Error	local-connection-name, description	Reports service-request or internal error

TCP Header

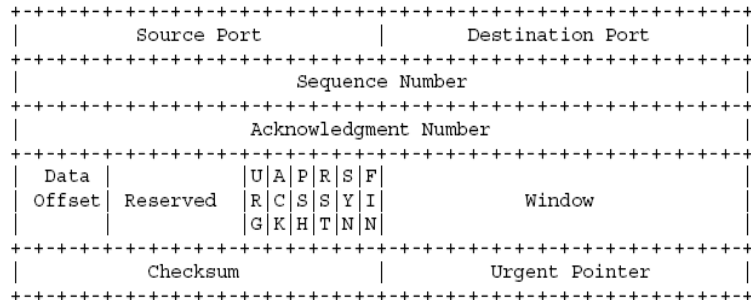

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Source Port
Destination Port
Sequence Number
Acknowledgment Number
Data Offset
Reserved
U R G
A C K
P S H
R S T
S Y N
F I N
Window
Checksum
Urgent Pointer
Options
Padding
data

TCP Header Fields



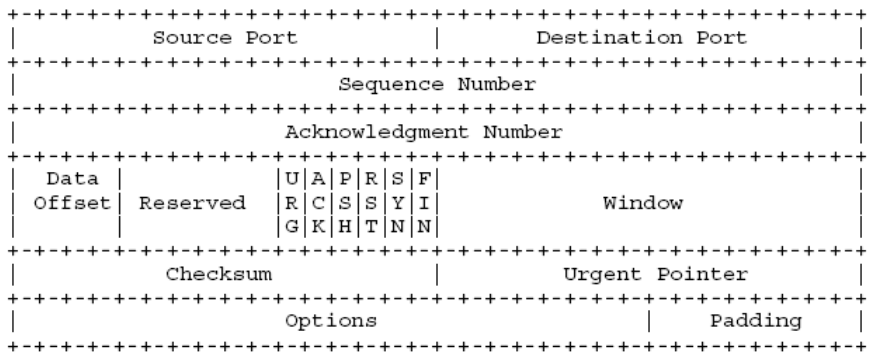
- Source port (16 bits)
- Destination port (16 bits)
  - Identify src and dest TCP user
- Sequence number (32 bits)
  - Seq number of first data octet
  - If SYN is set, it is ISN and first data octet is ISN+1
- ACK number (32 bits)
  - Piggybacked ACK
- Window (16 bits)
  - Credit allocation in octets, i.e. rcv_window of sender

TCP Header Fields



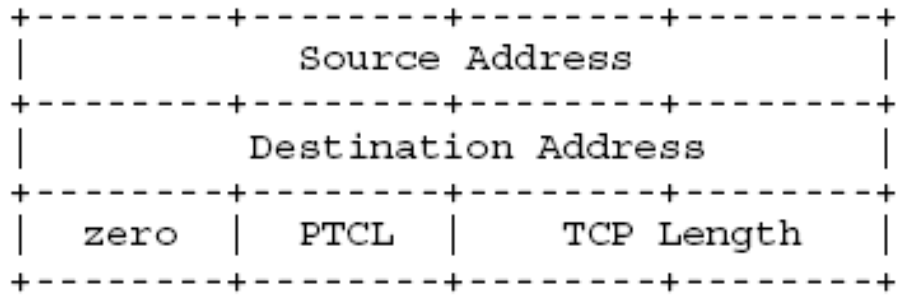
- Data offset (4 bits)
  - Number of 32-bit words in the header
  - Largest data offset is 15×4=60 octets
- Reserved (6 bits)
- Flags (6 bits):
  - *ACK*: acknowledgment field meaningful
  - *RST*: reset the connection
  - *SYN*: synchronize the sequence number
  - *FIN*: no more data from sender
  - *PSH*: push function
  - *URG*: urgent pointer field meaningful

TCP Header Fields



- Checksum (16 bits)
  - Header + Data + Pseudo-header

This gives the TCP protection against misrouted segments.



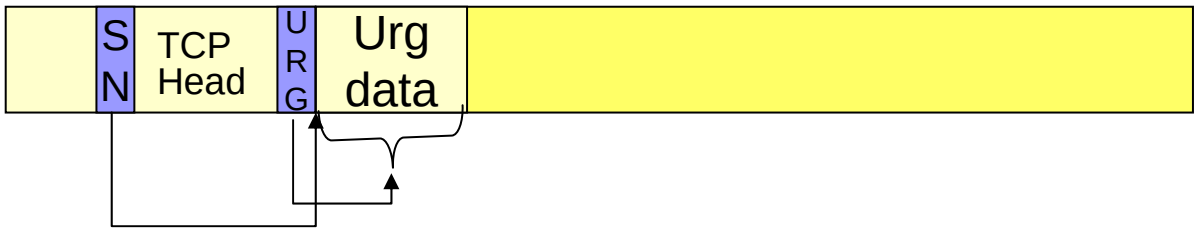
- Urgent Pointer (16 bits)
  - Points to last octet in a sequence of urgent data

TCP Header Fields — PUSH fuction

- Sometimes, an application needs to be sure that all the data passed to TCP has actually been transmitted to the destination.
  - For that reason, a push function is defined.
  - It will push all remaining TCP segments still in storage to the destination host.
  - The normal close connection function also pushes the data to the destination.

TCP Header Fields — Urgent Function

- It is a means of signaling the destination application that urgent data have arrived.
- The urgent signal is given to the application in advance of delivery of the data to the application.
- TCP does not attempt to define what the user specifically does upon being notified of pending urgent data, but the general notion is that the receiving process will take action to process the urgent data quickly.
- The Telnet synch signal is designed to allow the user to communicate some urgent command to a server process, such as an Interrupt Process.



TCP Header Fields

- Options: variable
  - There are two cases for the format of an option:
    - Case 1: A single octet of option-kind.
    - Case 2: An octet of *option-kind, an octet of option-length, and the actual option-data octets*. The option-length counts the two octets of option-kind and option-length as well as the option-data octets.
  - Note that the list of options may be shorter than the data offset field might imply. The content of the header beyond the End-of-Option option must be header padding (i.e., zero).

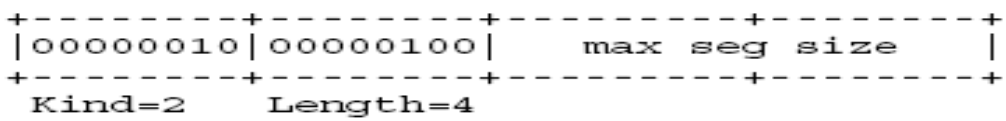
TCP Header Fields

- There are currently seven options defined:

Kind	Length	Meaning
0	-	End of option list
1	-	No-Operation
2	4	Maximum segment size
3	3	Window scale
4	2	Sack-Permitted
5	X	Sack
8	10	Timestamps

TCP Header Fields

- Maximum Segment Size



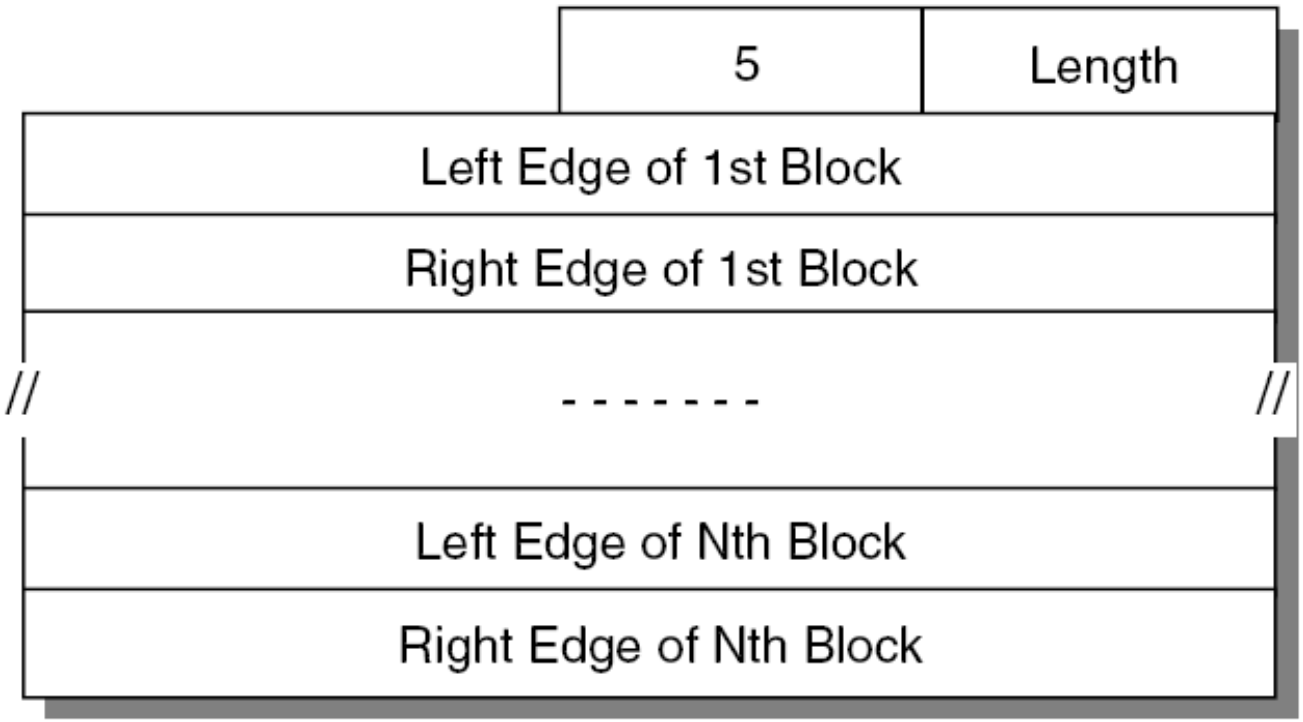
  - **It communicates the maximum receive segment size at the TCP which sends this segment.**
  - **This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set).**
  - **If this option is not used, any segment size is allowed.**

- Window Scale option
  - Both sides must send the Windows Scale Option in their SYN segments to enable windows scaling in their direction.
  - It defines the 32-bit window size by using scale factor in the SYN segment over standard 16-bit window size.
  - This option is determined while handshaking. There is no way to change it after the connection has been established.

Selective Acknowledgment

- SACK-Permitted option
  - This option is set when selective acknowledgment is used in that TCP connection.
- SACK option
  - Selective Acknowledgment (SACK) allows the receiver to inform the sender about all the segments that are received successfully.
  - Thus, the sender will only send the segments that actually got lost.
  - The number of blocks that can be reported by the SACK option is limited to four. To reduce this, the SACK option should be used for the most recent received data.

Selective Acknowledgment

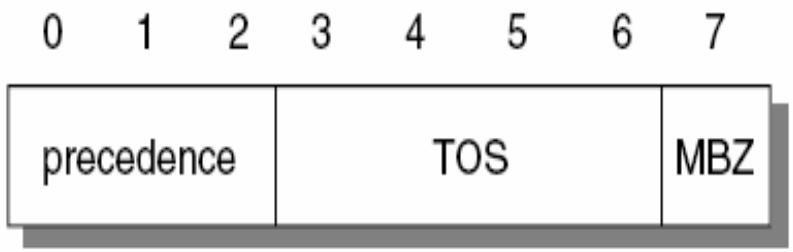


TCP Header Fields

- Padding
  - The TCP header padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary. The padding is composed of zeros.

Parameters Passed to IP

- TCP passes QOS parameters down to IP
  - Precedence
  - Normal delay / low delay
  - Normal throughput / high throughput
  - Normal reliability / high reliability
  - Security



1000: Minimize delay
0100: Maximize throughput
0010: Maximize reliability
0001: Minimize monetary cost
0000: Normal service

- IPv4 "Type of Service" or IPv6 "Traffic Class"

TCP Mechanisms (1)

- Connection establishment
  - 3-way handshake
  - Between pairs of ports
  - One port can connect to multiple destination ports
- Connection termination
  - Graceful termination
    - TCP users issues CLOSE primitive
    - Transport entity sets FIN flag on last segment sent
  - Abrupt termination by ABORT primitive
    - Entity abandons all attempts to send or receive data
    - RST segment transmitted

TCP Mechanisms (2)

- Data transfer
  - Logical stream of octets
  - Octets numbered modulo 2^{32}
  - Flow control by credit allocation of number of octets
  - Data buffered at sender and receiver
  - User sets PUSH to force data transmission immediately
  - User may specify a block of data as urgent
- Congestion control

Implementation Policy Options

- Send
- Deliver
- Accept
- Retransmit
- Acknowledge

Send

- If no PUSH or CLOSE, TCP entity transmits at its own convenience
- Data issued by TCP user buffered at transmit buffer
  - May construct segment per data batch
  - May wait for certain amount of data

Deliver

- In absence of PUSH, TCP entity delivers data at own convenience
- May deliver as each segment in order received
  - Deliveries (I/O interrupts) are frequent and small
- May buffer data from more than one segment
  - Deliveries are infrequent and large

Accept

- Segments may arrive out of order
- In order
  - Only accept segments in order
  - Discard out of order segments
  - Makes for a simpler implementation
- In windows
  - Accept all segments within receive window
  - Can reduce retransmission

Retransmit

- TCP entity maintains queue of segments transmitted but not acknowledged
- TCP will retransmit if not ACKed in given time
  - First only: one timer a queue, reset the timer after retransmission of first segment in queue
  - Batch: one timer a queue, reset after retransmission of all segments in queue
  - Individual: one timer each segment, reset after retransmission

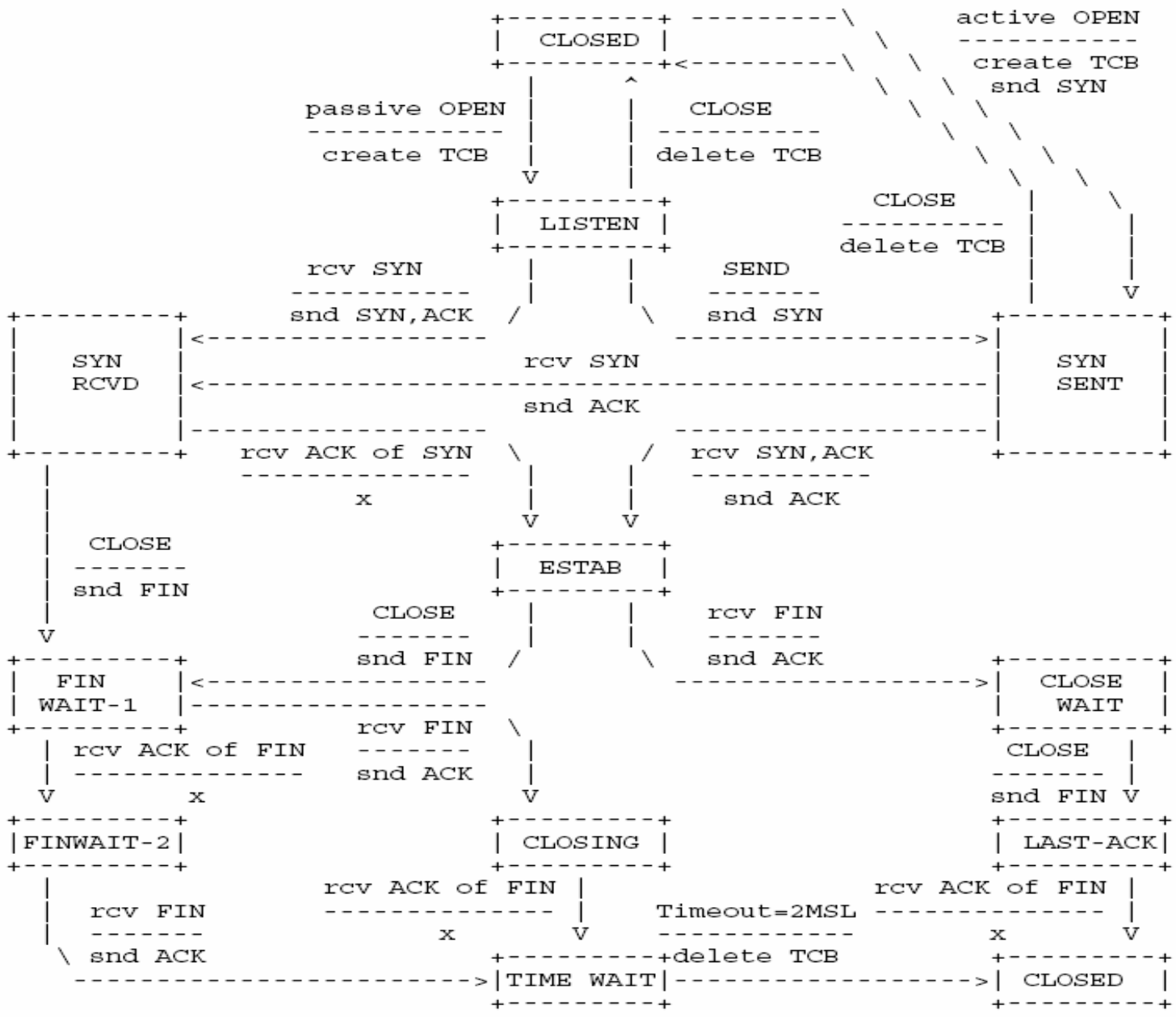
Fast Retransmit

- Time-out period often relatively long
- Detect lost segments via duplicate *ACKs*
  - If segment is lost, there will likely be many duplicate *ACKs*
- If a *TCP* entity receives 3 *ACKs* for the same data, then segments after ACKed data must be lost
  - Trigger fast retransmit: resend segment before timer expires

Acknowledgement

- Immediate or Cumulative

Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	Immediately send duplicate ACK, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediate send ACK, provided that segment sat at lower end of gap


CLOSED
active OPEN
create TCB
snd SYN
passive OPEN
create TCB
CLOSE
delete TCB
LISTEN
CLOSE
delete TCB
rcv SYN
snd SYN,ACK
SEND
snd SYN
SYN RCVD
rcv SYN
snd ACK
SYN SENT
rcv ACK of SYN
x
rcv SYN,ACK
snd ACK
CLOSE
snd FIN
ESTAB
CLOSE
snd FIN
rcv FIN
snd ACK
FIN WAIT-1
CLOSE WAIT
rcv FIN
snd ACK
rcv ACK of FIN
x
CLOSE
snd FIN
FINWAIT-2
CLOSING
LAST-ACK
rcv ACK of FIN
x
rcv FIN
snd ACK
Timeout=2MSL
delete TCB
rcv ACK of FIN
x
TIME WAIT
CLOSED

3. TCP Congestion Control

TCP Congestion Control

- Congestion control
  - Too many sources sending too much data too fast for Internet to handle
  - RFC 1122, Requirements for Internet hosts —Communication Layers，1989.
- End to end control, no network assistance
  - Retransmission timer
  - Window management

Retransmission Timer Management

(1) Estimate round trip delay by observing delay pattern
- Simple average
- Exponential average

(2) Set timer to value somewhat greater than estimate
- RFC 793
- RTT Variance Estimation (Jacobson's algorithm)

(3) How to set timer after retransmission
- Exponential RTO backoff algorithm

(4) When to sample the round trip delay
- Karn's Algorithm

Simple Average

- Term
  - RTT(i): round-trip time observed for the i^{th} transmitted segment
  - ARTT(k): **average round-trip time** for the first k segments
- Expression

$$ARTT(k+1) = \frac{1}{k+1}\sum_{i=1}^{k+1} RTT(i)$$

$$ARTT(k+1) = \frac{1}{k+1}(k \times ARTT(k) + RTT(k+1))$$

$$= \frac{k}{k+1} ARTT(k) + \frac{1}{k+1} RTT(k+1)$$

Exponential Average

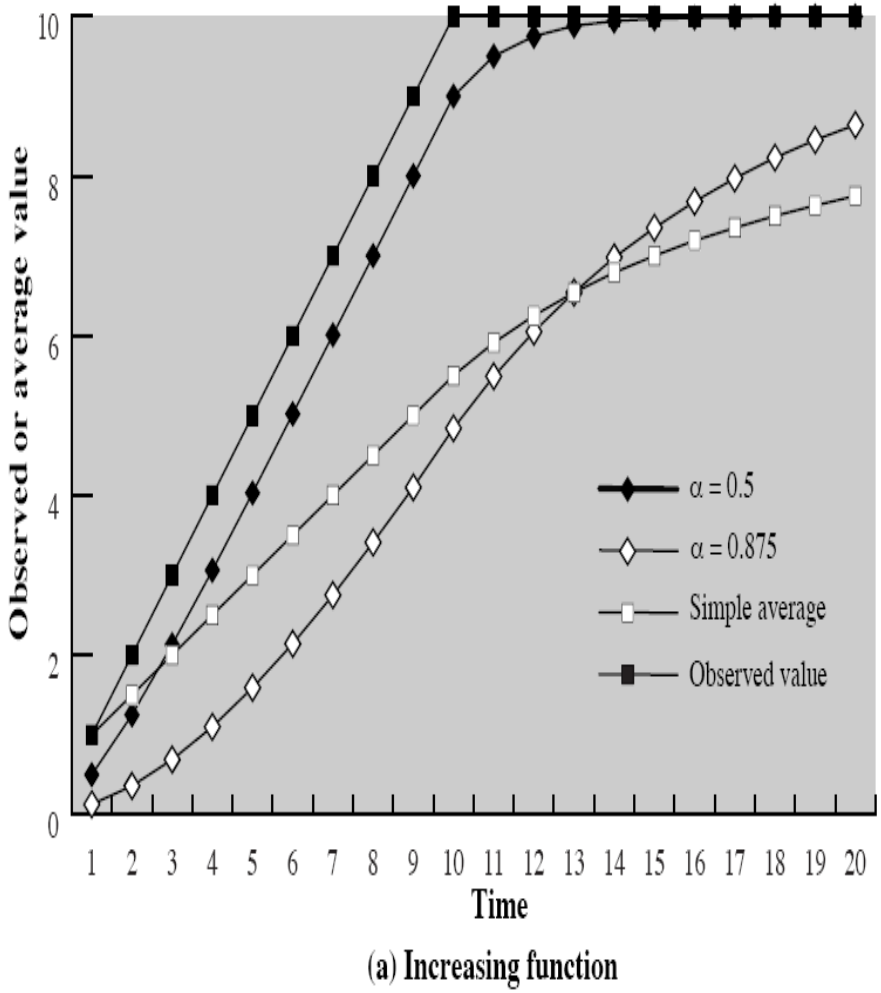
- Term
  - SRTT(k): **smoothed round-trip time** estimate for the first k segments
- Expression

$$SRTT(k+1) = \alpha \times SRTT(k) + (1-\alpha) \times RTT(k+1) \quad \text{i.e.}$$

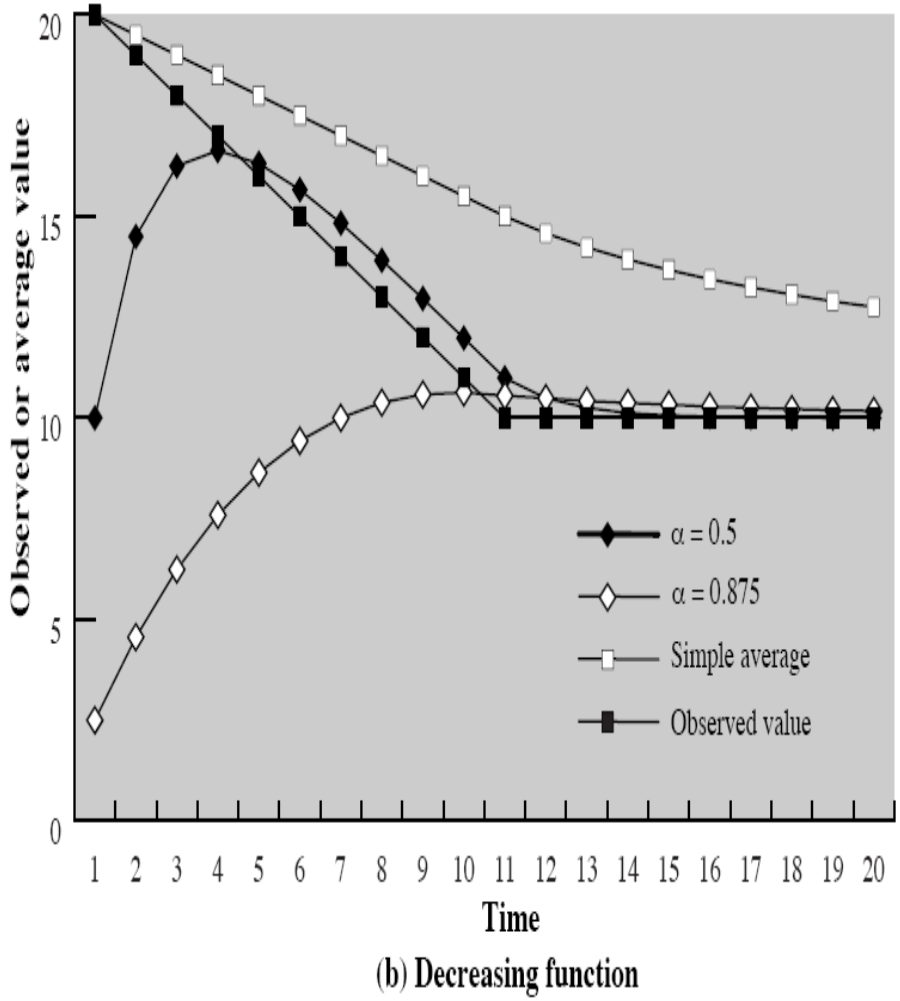
$$SRTT(k+1) = (1-\alpha) \times RTT(k+1) + \alpha(1-\alpha) \times RTT(k) +$$
$$\alpha^2(1-\alpha) \times RTT(k-1) + \ldots + \alpha^k(1-\alpha) \times RTT(1)$$

- small α
  - **quickly reflect a rapid change**
  - **jerky change in the average**

Simple and Exponential Averaging



(a) Increasing function



(b) Decreasing function

RFC 793

- Term
  - RTO(k): **retransmission timeout**, i.e. the timer after the first k segments
- Expression
  - Retransmission timer set between LBOUND~UBOUND

RTO(k+1) = Min (UBOUND, Max (LBOUND, $\beta \times$SRTT(K+1)))

  - Suggested values, α: 0.8~0.9, β: 1.3~2.0

Jacobson's Algorithm (**RTT Variance Estimation**)

- Problem in RFC 793
  - Not counting variance of RTT (network stability)
  - When network is stable, RTT variance is low, but β=1.3 gives a higher RTO
  - When network is unstable, RTT variance is high, β=2 is inadequate to protect against retransmissions.
- Three source of high variance
  - Data tansmisson delay is relatively large.
  - Internet trafic load and conditions may change abruptly.
  - The peer TCP entity may may not acknowlidge each segment immediately
    - processing delays
    - cummulative acknowledgements

Jacobson's Algorithm (**RTT Variance Estimation**)

- Term
  - SERR(k): smoothed error estimate, difference of round-trip time of segment k and the current SRTT
  - SDEV(k): standard deviation for round-trip time of first k segments
- Mean deviation MDEV(X) = E[| X-E(X) |]
- Expression

SRTT(K+1) =(1－g) •SRTT(k) + g•RTT(k+1)

SERR(k+1) = RTT(k+1) － SRTT(k)

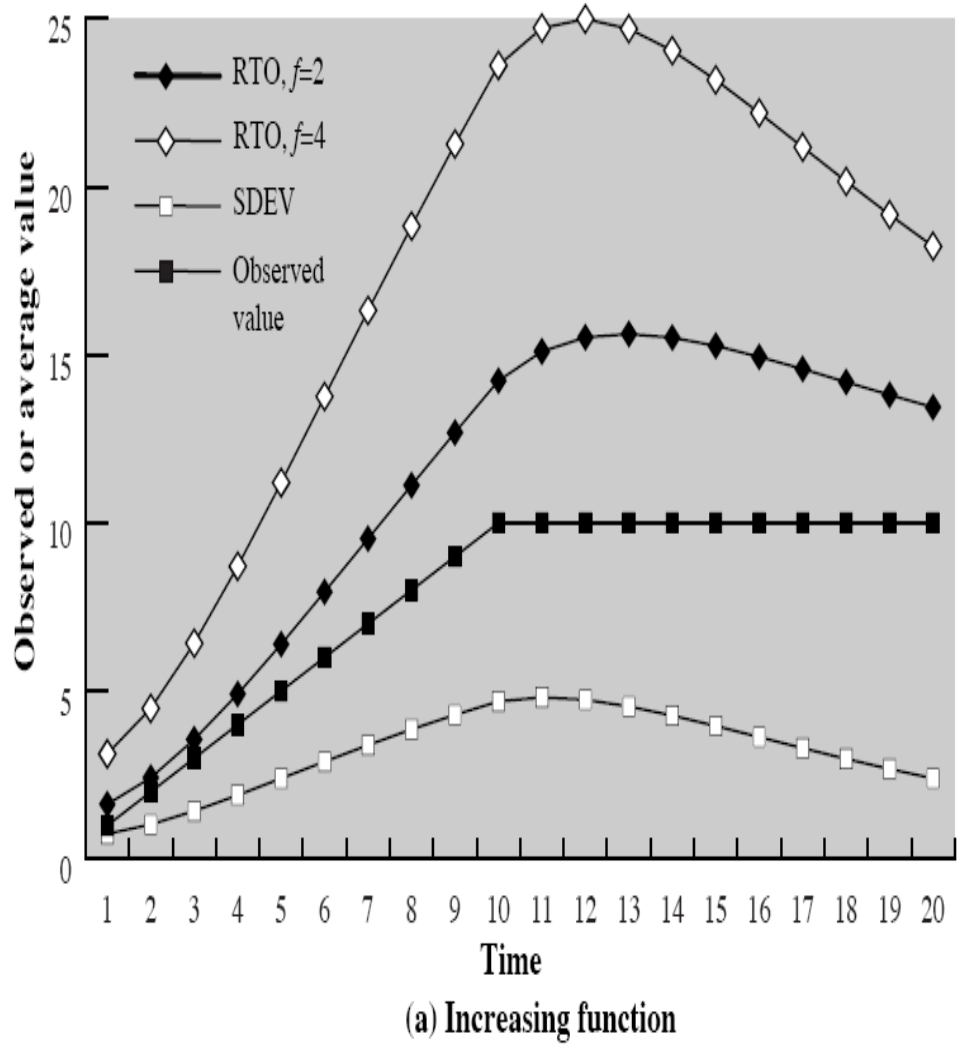
SDEV(K+1)=(1-h) •SDEV(K) + h • | SERR(k+1) |

RTO(k+1) = SRTT(k+1) + f • SDEV(K+1)

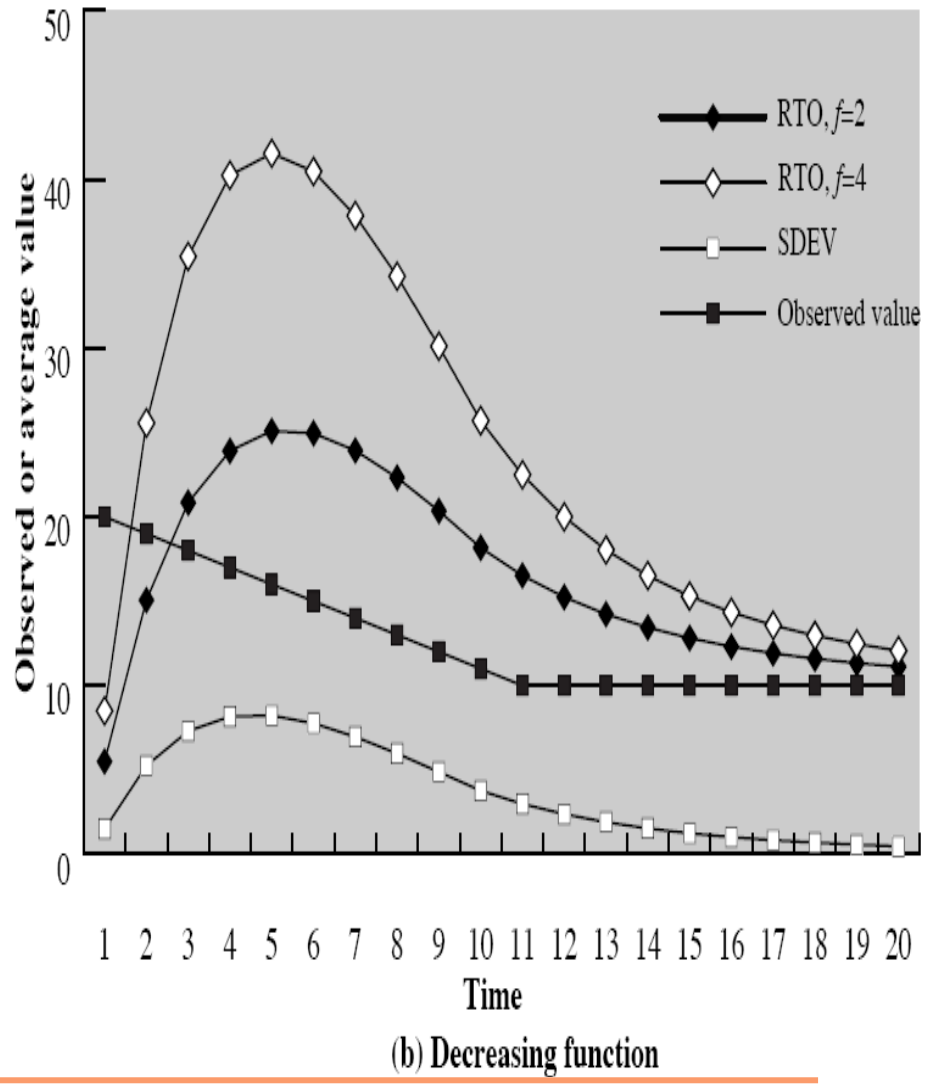
g=1/8=0.125, h=1/4=0.25, f= 2 or 4

RFC2988：RTO(k+1) = SRTT(k+1) + Max(G, f • SDEV(K+1))

Jacobson's RTO Calculation



(a) Increasing function



(b) Decreasing function

- Jacobson's algorithm can significantly improve TCP performance.

- However, it does not stand by itself.

  (1) What RTO value should be used on a retransmitted segment?

  The exponetial RTO backoff algorithm is used for this puroose.

  (2) Which round-trip samples shoud be used as input to Jacobsin's algorithm determines which samples to use.

  Karn's algorithm determines which samples to use.

Exponential RTO Backoff

- RFC 793 assumes that the same RTO will be used for retransmissioned segment.
- Because the timeout is often due to congestion by dropped packet or a long delay in RTT，maintainning the same RTO value is ill advised.
- Should slow down end system transmission

- Similar to Binary exponential backoff in Ethernet
  - multiply RTO for a re-transmitted segment by a constant value
  - $RTO = q \times RTO$
  - Commonly q=2

Karn's Algorithm

- The problem
  - If a segment is re-transmitted, the ACK arriving may be
    - For the first copy of the segment, or
    - For the second copy, or others
  - No way to tell
- RTT Sampling
  - Do not measure RTT for re-transmitted segments
  - Calculate RTO backoff when re-transmission occurs
  - Until ACK arrives for segment that has not been re-transmitted
    - Begin sampling, stop RTO backoff

Window Management

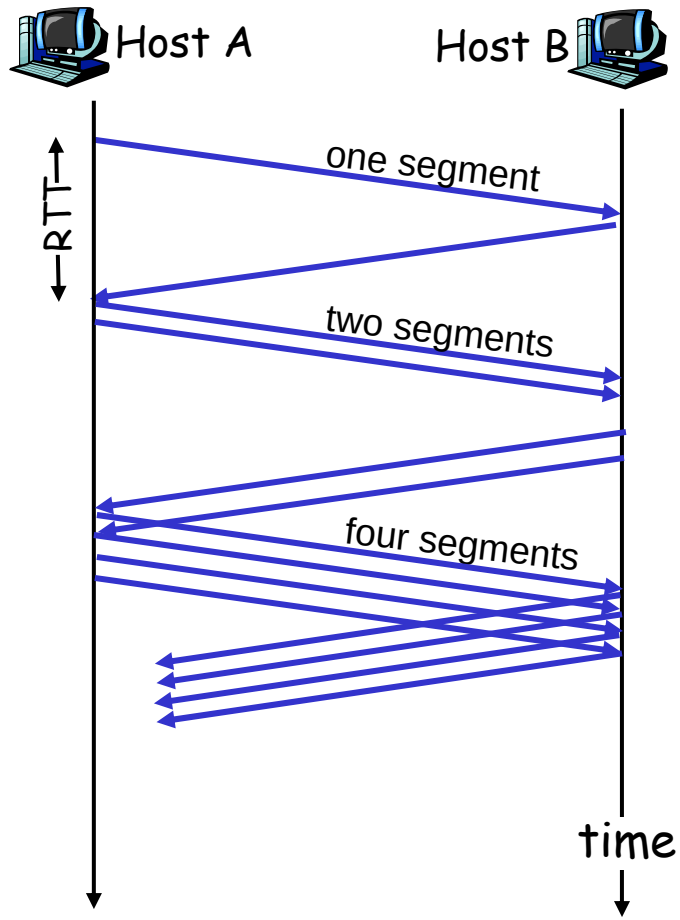
- **awnd**: allowed window in segments, in *MSS* (maximum segment size)
- *credit*: the amount of unused credit granted in last *ACK*, in *MSS*
- **cwnd**: congestion window in segments, in *MSS*.

awnd = Min { credit, cwnd}

- Manage congestion window
  - **Slow Start**: exponentially expending the *cwnd* at start of connection
  - **Dynamic Window on Congestion**: shrinking / expending the *cwnd* with stages when retransmission occurs

Slow Start

- When connection begins, **cwnd = 1 MSS**
- Each time an *ACK* received, **cwnd** increased by ACKed number of *MSSs* until Max value reached
- cwnd increased exponentially until first loss event occurs
  - Timeout or 3 duplicate *ACKs*



Dynamic Windows Sizing

- By Jacobson, set slow start threshold ssthresh = cwnd/2
- After timeout event
  - cwnd is set to 1
  - Same as Slow Start until cwnd reaches ssthresh
  - cwnd increases by 1 after each RTT or ACK received

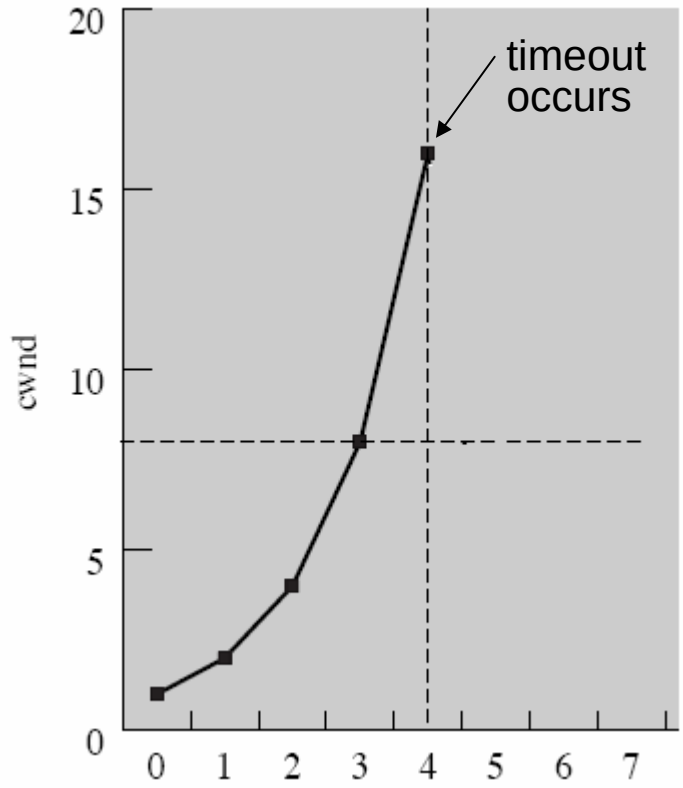
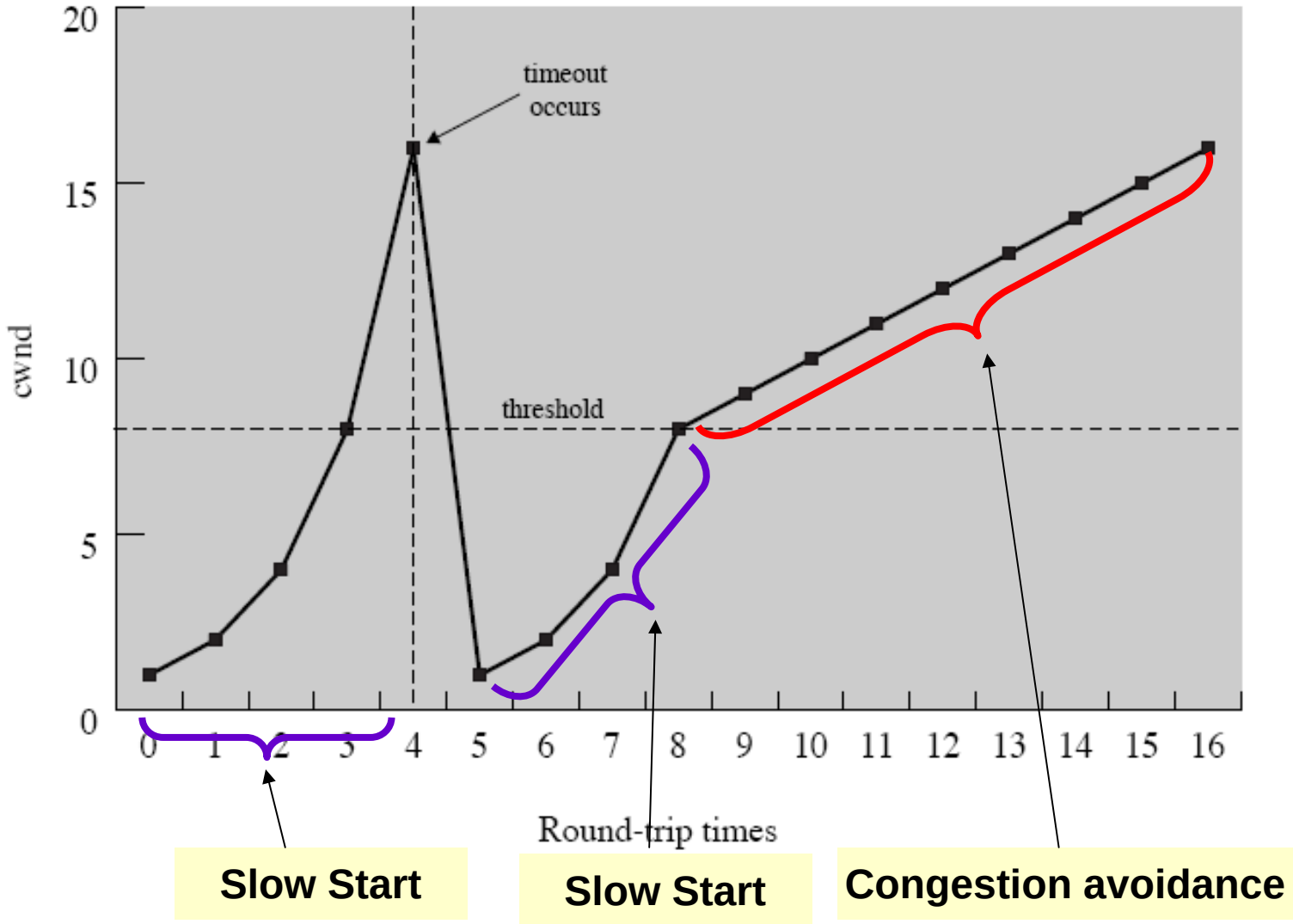


Illustration of Slow Start and Congestion Avoidance



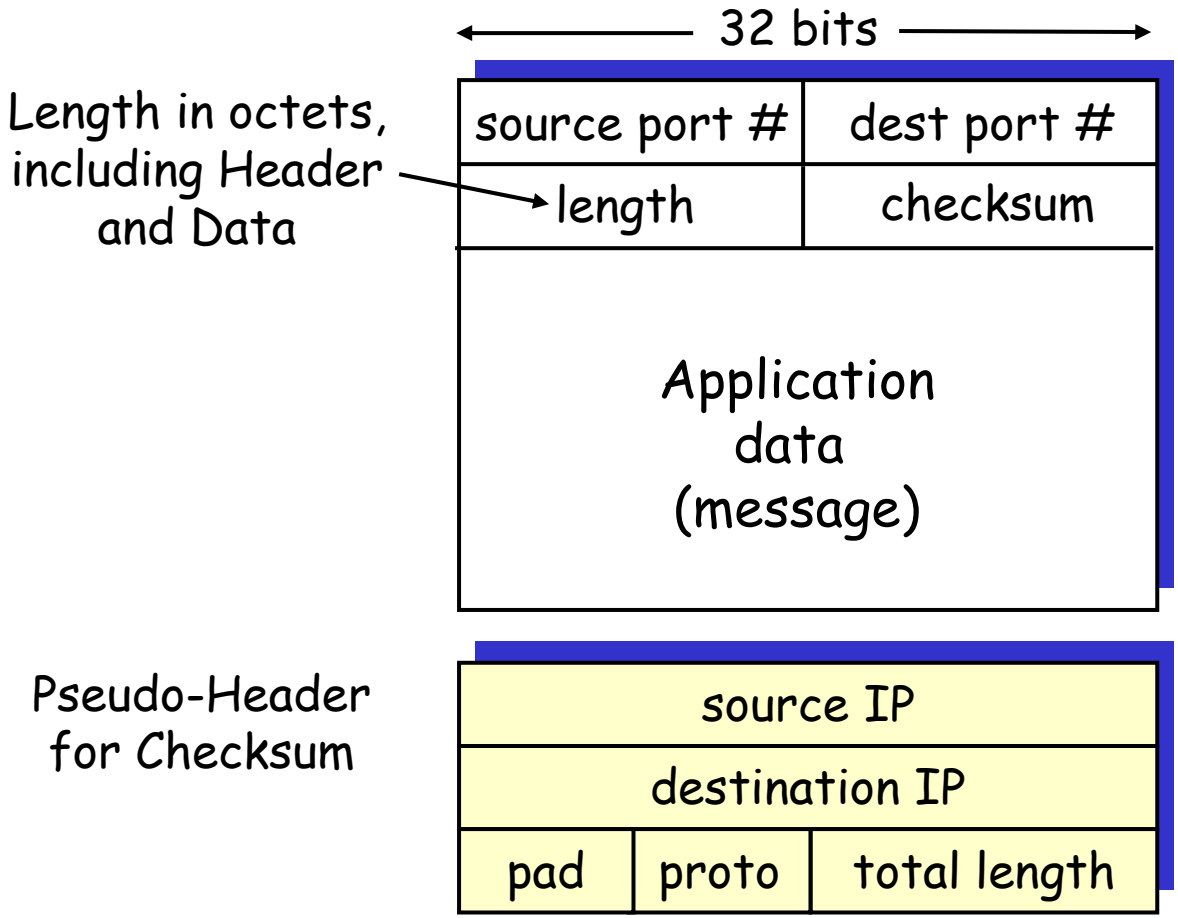
UDP

- User datagram protocol, RFC 768
- **Connectionless service** for application level processes
  - Unreliable, “best-effort” of IP
  - Each UDP segment handled independently of others
  - Delivery and duplication control not guaranteed
- **Simple and reduced overhead**
  - No connection establishment
  - No connection state at sender, receiver
  - Small segment header

UDP Uses

- Normal use
  - Inward data collection from sensors
  - Outward data dissemination
  - Real time applications
  - Request-Response (e.g. RPC), add reliability at application layer
- Example UDP uses
  - DNS
  - SNMP

UDP Segment Format



Summary

- **Transport Mechanisms**
  - Connection Oriented Mechanisms
  - Transport Protocol on Reliable Sequencing Network, Flow Control
  - Problems for Unreliable Networks, 3-way Handshake
- **TCP – Transmission Control Protocol**
  - TCP Services, Request and Response Primitives, TCP Header Fields
  - TCP Mechanisms and Implementation Policies
- **TCP Congestion Control**
  - Retransmission timer, Jacobson's Algorithm, Exponential RTO Backoff, Karn's Algorithm
  - Window Management, Slow Start, Dynamic Windows Sizing
- **UDP – User Datagram Protocol**
  - UDP Uses, UDP Segment Format

Exercises

- 20.1
- 20.3
- 20.5
- 20.8
- 20.9
- 20.14
- 20.15