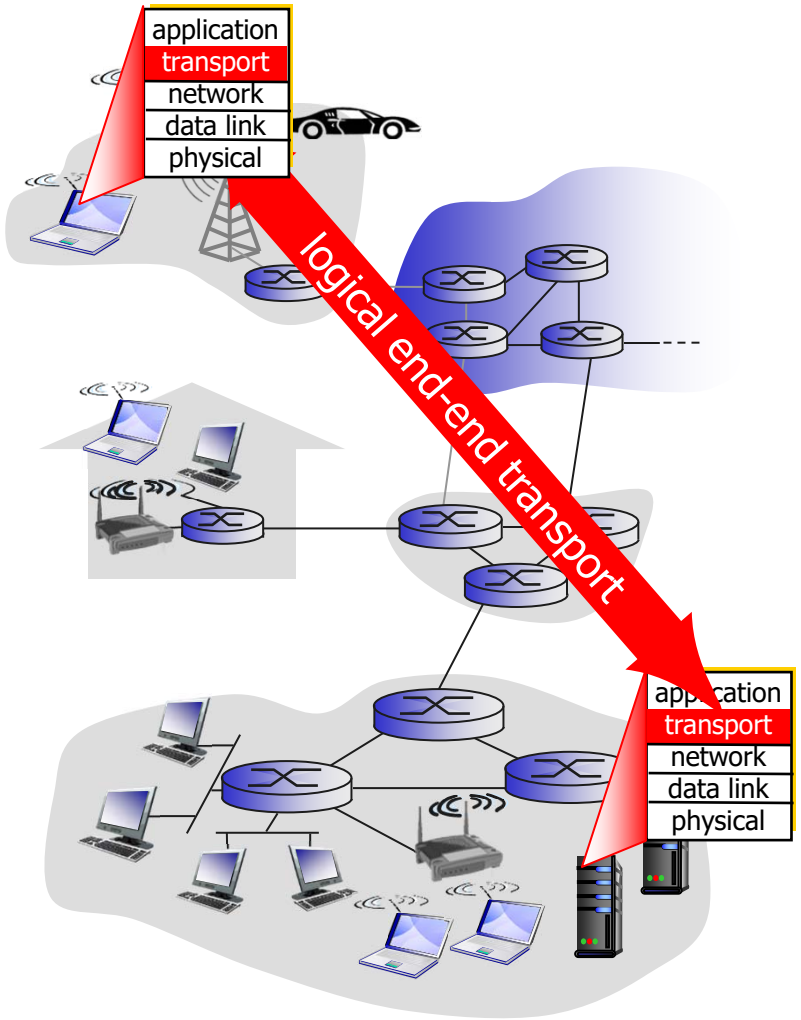
Transport
& Congestion Control:

William Stallings, *Data and Computer Communications*, 10th Edition, Pearson, 2014:
Chapter 15 Transport Protocols
Chapter 20 Congestion Control.

Jim Kurose, Keith Ross, *Computer Networking: A Top Down Approach*, 7th Edition, Pearson, 2017
Chapter 3 Transport Layer

Transport services and protocols

- provides logical communication between app processes running on different hosts
- relies on, enhances, network layer services
- (recall: network layer: logical communication between hosts)
- transport protocols run in end systems
  - Tx (send) side: breaks app messages into segments, passes to network layer
  - Rx side: reassembles segments into messages, passes to app layer
- Internet transport protocols: TCP and UDP
- Transmission Control Protocol (TCP) provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating by an IP network (congestion control, flow control, connection setup). Major Internet applications such as the World Wide Web, email, remote administration, and file transfer rely on TCP.
- Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that emphasizes reduced latency over reliability.(extension of "best-effort" IP, unreliable, unordered delivery)



Internet: application & transport protocols

TCP service:

- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *connection-oriented:* setup required between client and server processes
- *does not provide:* timing, minimum throughput guarantee, security

UDP service:

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Multiplexing/demultiplexing

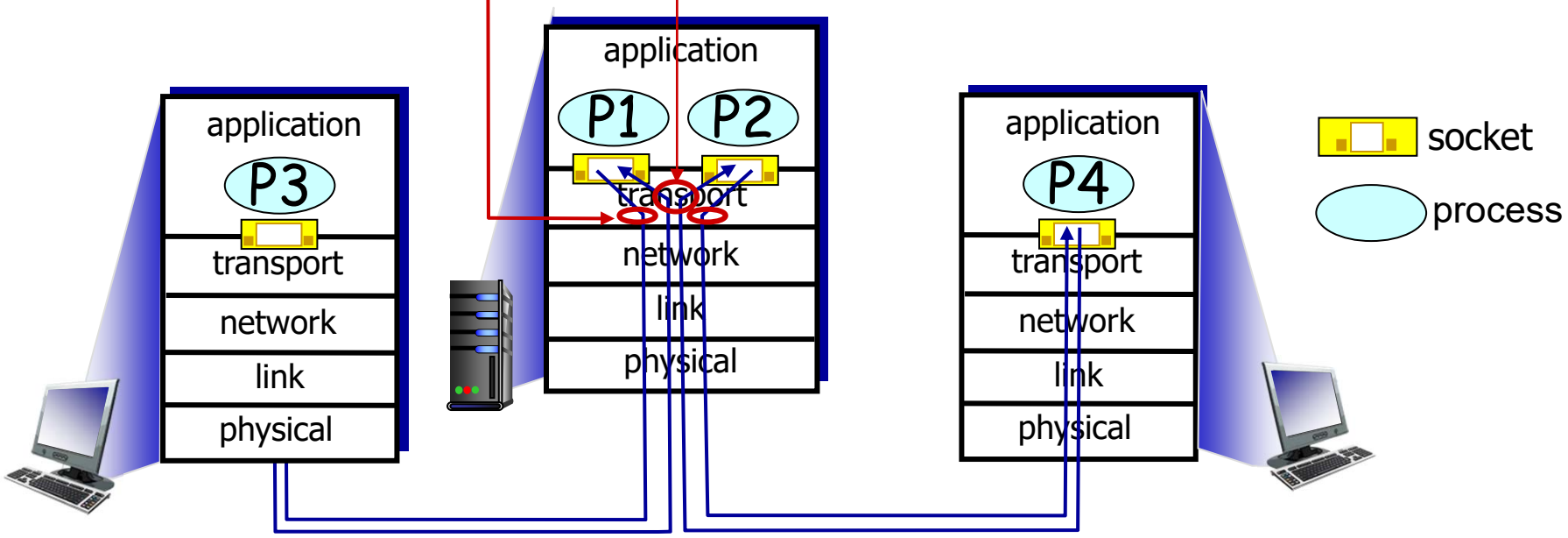
Multiple users employ the same transport protocol and are distinguished by port numbers or service access points

multiplexing at sender:

handle data from multiple sockets, add transport header (later used for demultiplexing)

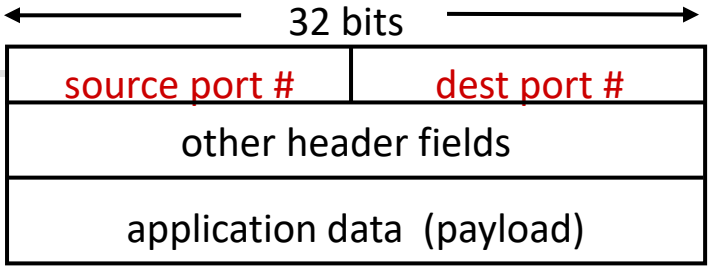
demultiplexing at receiver:

use header info to deliver received segments to correct socket



demultiplexing

- host receives IP datagrams
  - ✓ each datagram has source/destination IP address
  - ✓ each datagram carries one transport-layer segment
  - ✓ each segment has source/destination port number



TCP/UDP segment format

Connectionless demultiplexing:

- when host receives UDP segment, it only checks destination port number in segment, and directs UDP segment to socket with that port number
  - IP datagrams with *same dest. port #,* but different source IP addresses and/or source port numbers will be directed to *same socket* at destination

Connection-oriented demultiplexing:

- TCP socket identified by 4-tuple: Source/ dest IP address/port number
- receiver uses all four values to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

User Datagram Protocol (UDP)

- Transport-level protocol that is commonly used as part of the TCP/IP protocol suite (RFC 768)
- Provides a connectionless service for application-level procedures
  - each UDP segment handled independently of others
  - no connection state no handshaking between UDP sender, receiver: simple
  - no connection establishment (which can add delay)
  - small header size
- "best effort" service, UDP segments may be lost or delivered out-of-order to app
- Unreliable service; delivery and duplicate protection are not guaranteed
- Reduces overhead and may be adequate in many cases

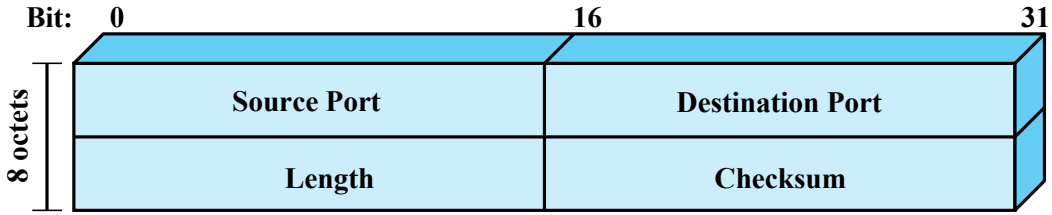


Figure 15.11 UDP Header

- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

UDP checksum

Goal: detect "errors" in transmitted segment

sender:

- treat segment contents, including header fields, as sequence of 16-bit integers
- checksum: one's complement sum of segment contents
- sender puts checksum value into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value: NO (error detected)

example:

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Transmission Control Protocol (TCP)

RFCs: 793,1122,1323, 2018, 2581

TCP MECHANISMS:

Connection establishment:

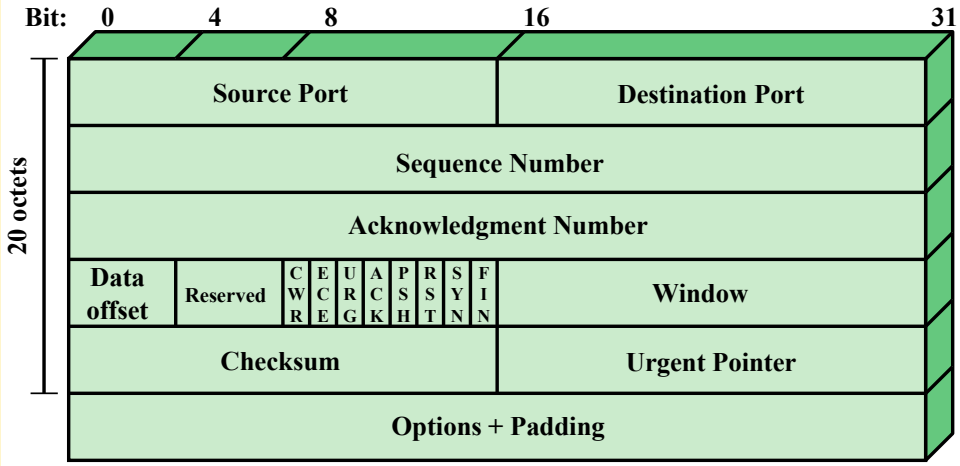
- Always uses a three-way handshake
- Connection is determined by host and port

Data transfer:

- Viewed logically as consisting of a stream of octets
- Flow control is exercised using credit allocation

Connection termination:

- Each TCP user must issue a CLOSE primitive
- An abrupt termination occurs if the user issues an ABORT primitive



TCP header:

- Sequence Number (32 bits): Sequence number of the first data octet in this segment except when the SYN flag is set. If SYN is set, this field contains the initial sequence number (ISN) and the first data octet in this segment has sequence number ISN + 1.
- Flags (8 bits): For each flag, if set to 1, the meaning is
  - CWR: congestion window reduced.
  - ECE: ECN-Echo; the CWR and ECE bits, defined in RFC 3168, are used for the explicit congestion notification function; a discussion of this function is beyond our scope.
  - URG: urgent pointer field significant.
  - ACK: acknowledgment field significant.
  - PSH: push function.
  - RST: reset the connection.
  - SYN: synchronize the sequence numbers.
  - FIN: no more data from sender.

TCP services:

TCP service:

- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *connection-oriented:* setup required between client and server processes

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.

Table 15.2: TCP Service Request 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

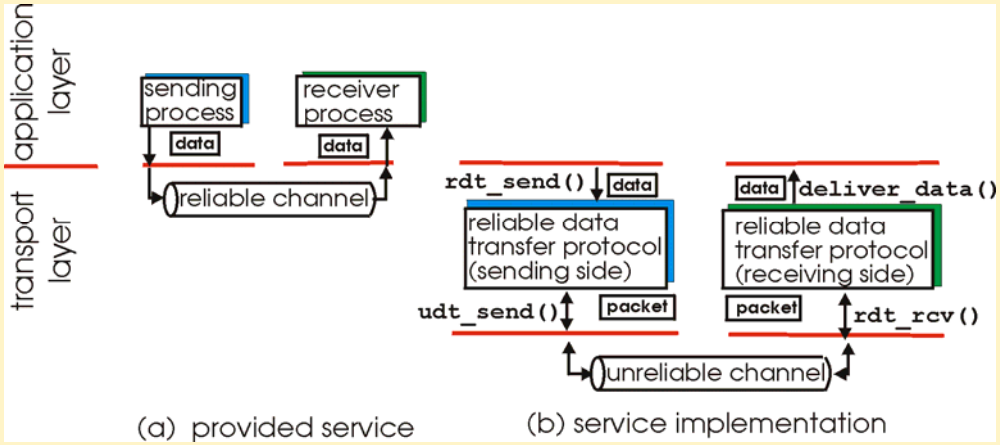
Table 15.3: TCP Service Response Primitives

Table 15.4: TCP Service Parameters

Source Port	Local TCP user
Timeout	Longest delay allowed for data delivery before automatic connection termination or error report; user specified
Timeout-action	Indicates whether the connection is terminated or an error is reported to the TCP user in the event of a timeout
Precedence	Precedence level for a connection. Takes on values zero (lowest) through seven (highest); same parameter as defined for IP
Security-range	Allowed ranges in compartment, handling restrictions, transmission control codes, and security levels
Destination Port	Remote TCP user
Destination Address	Internet address of remote host
Security	Security information for a connection, including security level, compartment, handling restrictions, and transmission control code; same parameter as defined for IP
Data	Block of data sent by TCP user or delivered to a TCP user
Data Length	Length of block of data sent or delivered
PUSH flag	If set, indicates that the associated data are to be provided with the data stream push service
URGENT flag	If set, indicates that the associated data are to be provided with the urgent data signaling service
Local Connection Name	Identifier of a connection defined by a (local socket, remote socket) pair; provided by TCP
Description	Supplementary information in a Terminate or Error primitive
Source Address	Internet address of the local host
Connection State	State of referenced connection (CLOSED, ACTIVE OPEN, PASSIVE OPEN, ESTABLISHED, CLOSING)
Receive Window	Amount of data in octets the local TCP entity is willing to receive
Send Window	Amount of data in octets permitted to be sent to remote TCP entity
Amount Awaiting ACK	Amount of previously transmitted data awaiting acknowledgment
Amount Awaiting Receipt	Amount of data in octets buffered at local TCP entity pending receipt by local TCP user
Urgent State	Indicates to the receiving TCP user whether there are urgent data available or whether all urgent data, if any, have been delivered to the user

TCP over unreliable network service

- Examples: Internetwork using IP; Frame relay network using only the LAPF core protocol; IEEE 802.3 LAN using the unacknowledged connectionless LLC service
- Segments are occasionally lost and may arrive out of sequence due to variable transit delays
- Issues to Address: Ordered delivery, Retransmission strategy, Duplicate detection, Flow control, Connection establishment, Connection termination, Failure recovery



characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

- **Ordered Delivery**: With an unreliable network service it is possible that segments may arrive out of order → Solution: number segments sequentially. TCP: each data octet is implicitly numbered
- **Retransmission Strategy**:
  - Segment may be damaged in transit but still arrives at its destination, or fails to arrive
  - Sending transport does not know transmission was unsuccessful
  - Receiver acknowledges successful receipt by returning a segment containing an ack number

Retransmission timer	Retransmit an unacknowledged segment
MSL (maximum segment lifetime) timer	Minimum time between closing one connection and opening another with the same destination address
Persist timer	Maximum time between ACK/CREDIT segments
Retransmit-SYN timer	Time between attempts to open a connection
`Keepalive timer`	Abort connection when no segments are received

- No acknowledgment will be issued if a segment does not arrive successfully
- A timer needs to be associated with each segment as it is sent
- If timer expires before acknowledgment is received, sender must retransmit

TCP: Flow/Congestion/Error Control

- **Flow Control**: to prevent that the sender overruns the receiver with information
- **Congestion Control**: to prevent that the sender overloads the network
- **Error Control**: to recover or conceal the effects from packet losses

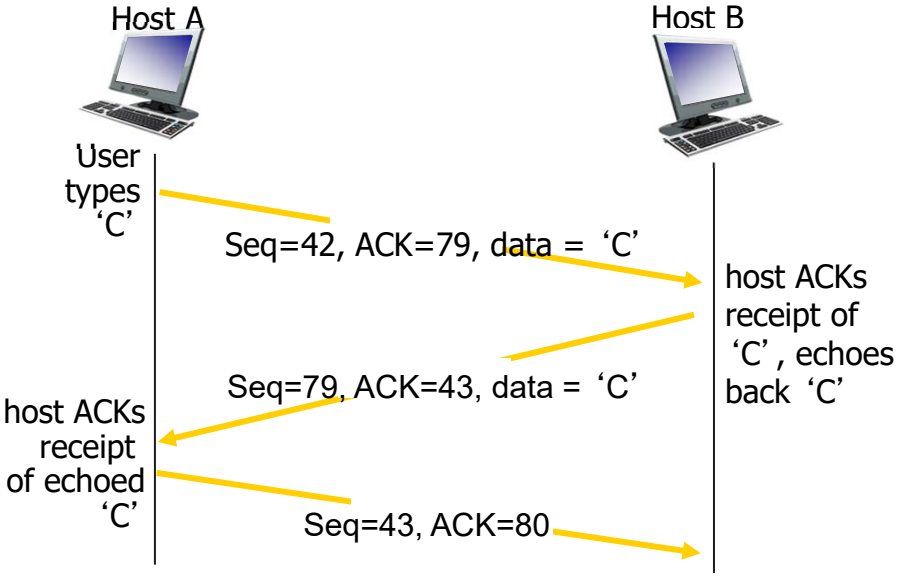
  - the control mechanisms have different goals.
  - However, the implementation is combined

TCP sequence numbers, ACKs

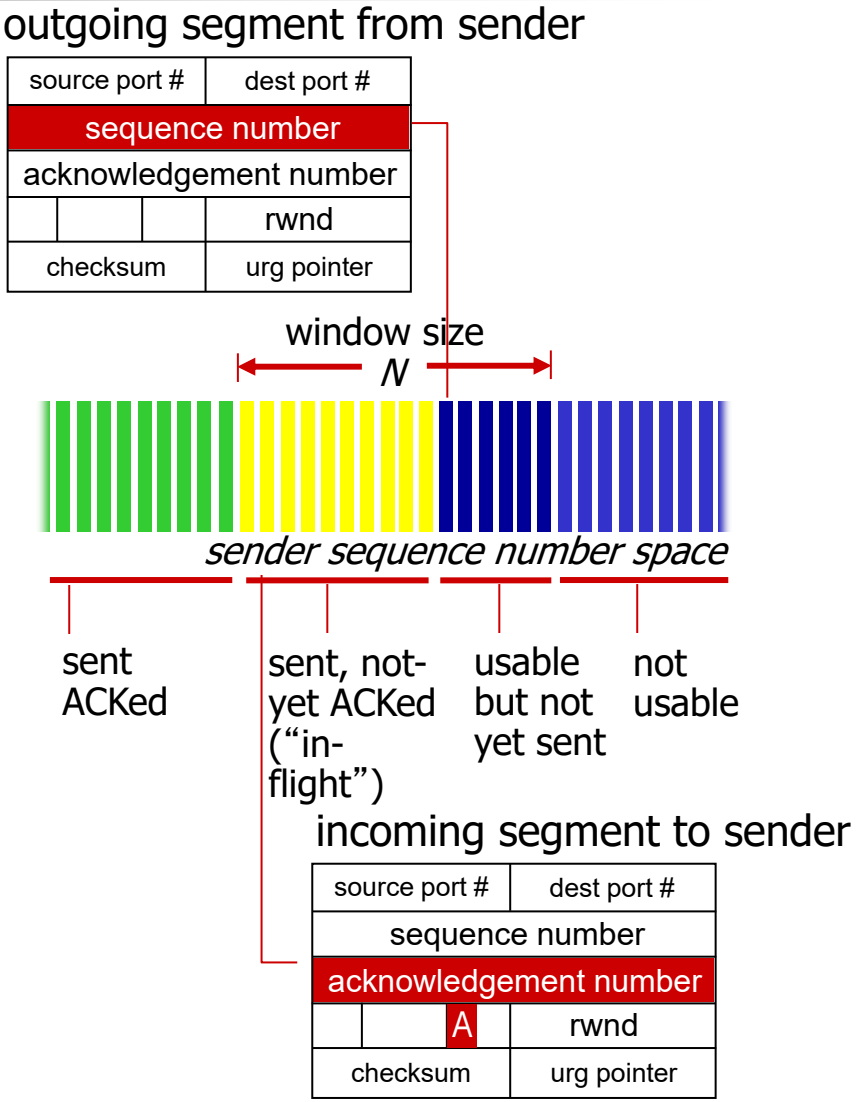
sequence numbers:
- byte stream "number" of first byte in segment's data

acknowledgements:
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments
- A: TCP spec doesn't say, - up to implementor

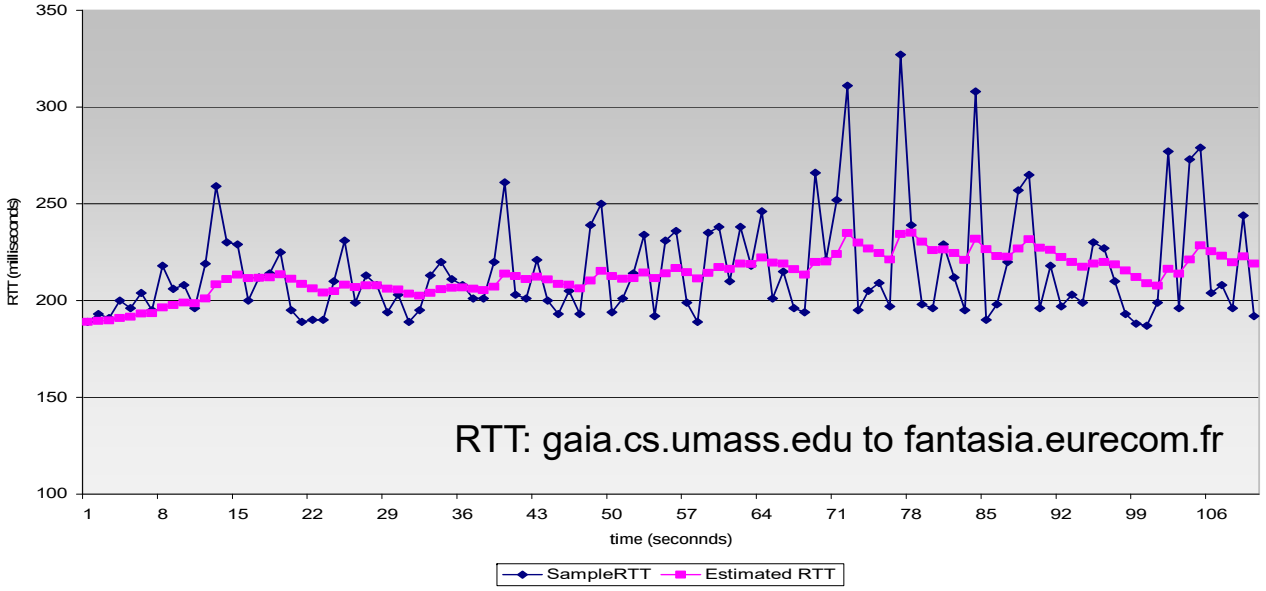


simple telnet scenario



TCP round trip time (RTT), timeout

- Ideally, TCP timeout interval must be larger than RTT. But RTT varies:
  - If set *too short* → premature timeout introducing unnecessary retransmissions
  - If set *too long* → slow reaction to segment loss
- SampleRTT: measured time from segment transmission until ACK receipt (ignore retransmissions). SampleRTT will vary, want Estimated RTT "smoother"
- EstimatedRTT=(1-α)*EstimatedRTT+α*SampleRTT, typical value: α = 0.125
- timeout interval : TimeoutInterval=EstimatedRTT+4*DevRTT
  - large variation in EstimatedRTT → larger safety margin
- estimate SampleRTT deviation from Estimated RTT:
- DevRTT =(1-β)*DevRTT+β*|SampleRTT-EstimatedRTT|, (typical β=0.25)



TCP sender events:

data received from app:

- create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unacked segment
  - expiration interval: **`TimeOutInterval`**

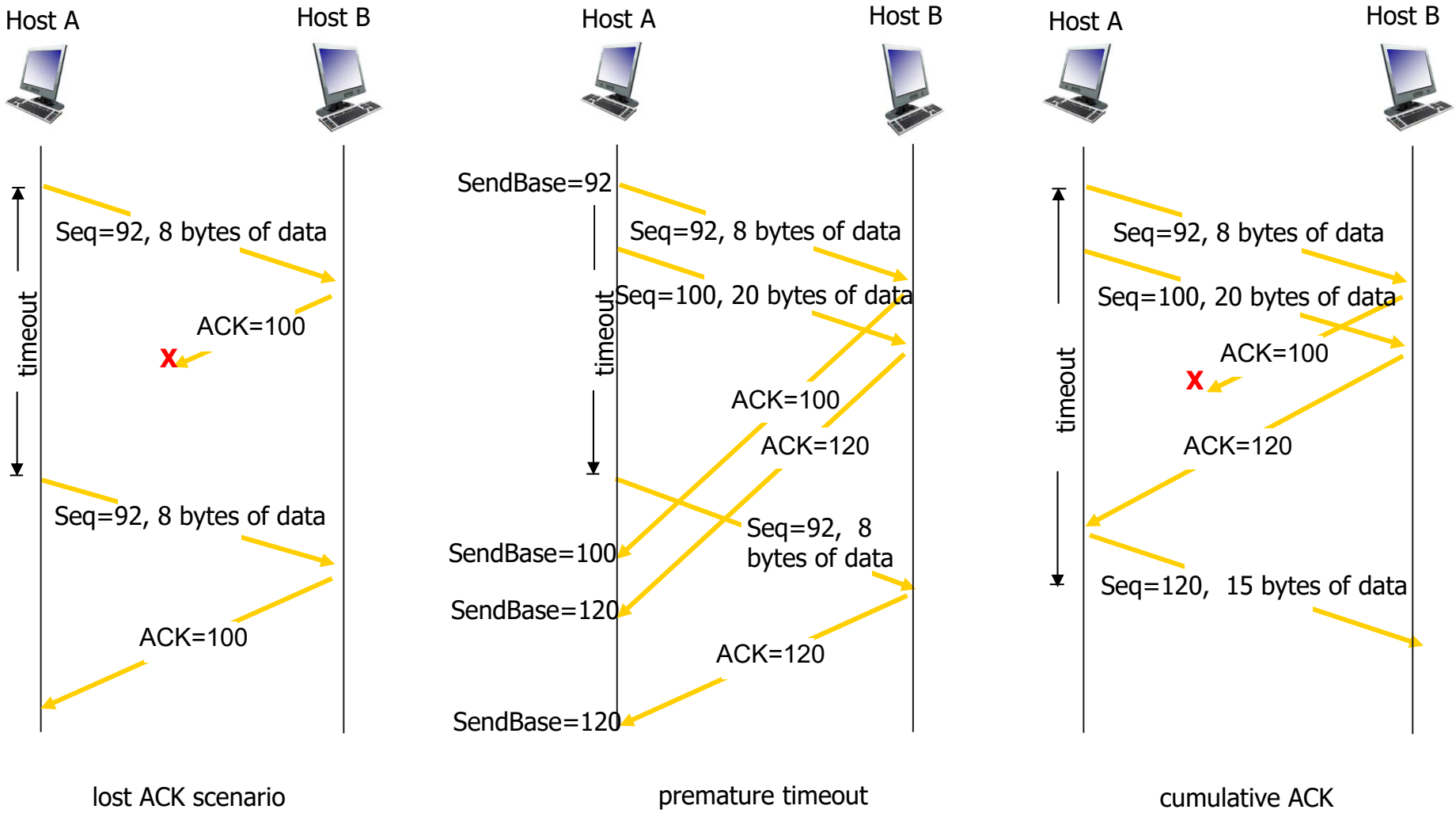
timeout:

- retransmit segment that caused timeout
- restart timer

ack received:

- if ack acknowledges previously unacked segments
  - update what is known to be ACKed
  - start timer if there are still unacked segments

TCP: retransmission scenarios



TCP ACK generation [RFC 1122, RFC 2581]

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 starts at lower end of gap

Duplicate Detection

- If a segment is lost and then retransmitted, no confusion will result.
- If, one or more segments in sequence are successfully delivered, but the corresponding ACK is lost, then the sending transport entity will time out and one or more segments will be retransmitted. If these retransmitted segments arrive successfully, they will be duplicates of previously received segments.
- Receiver must be able to recognize duplicates
- Segment sequence numbers help
- Complications arise if:
  1. A duplicate is received prior to the close of the connection:
     - Sender must not get confused if it receives multiple acknowledgments to the same segment
     - Sequence number space must be long enough
  2. A duplicate is received after the close of the connection

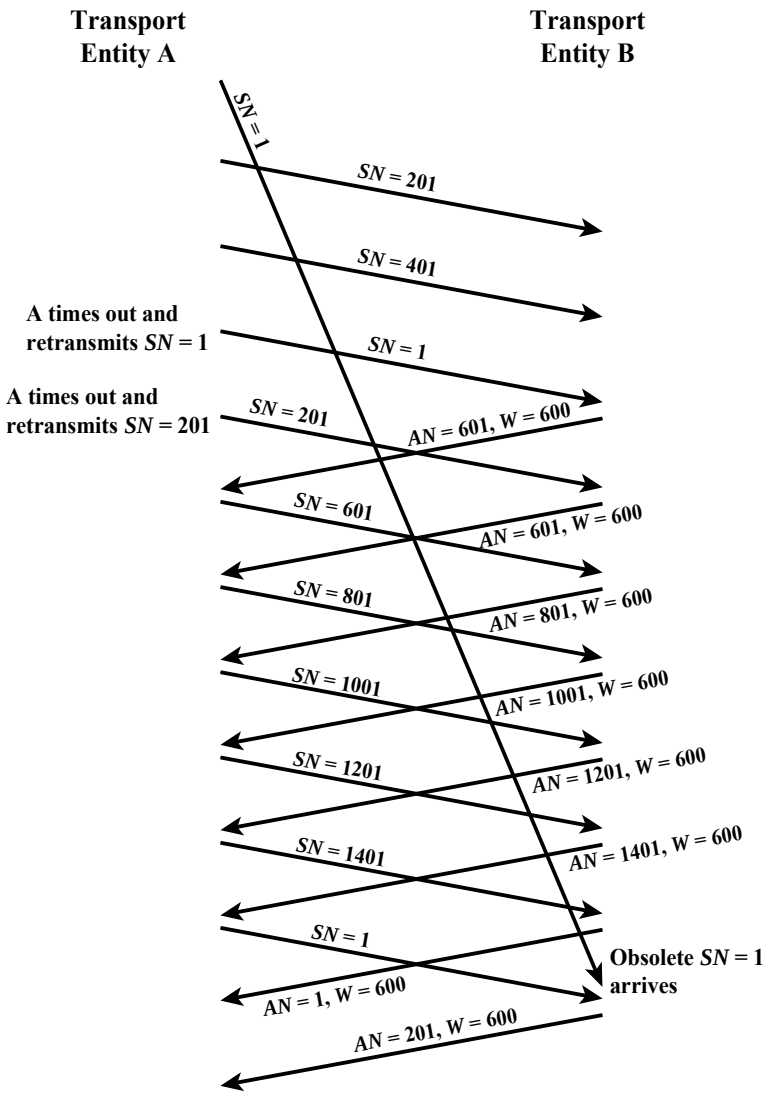
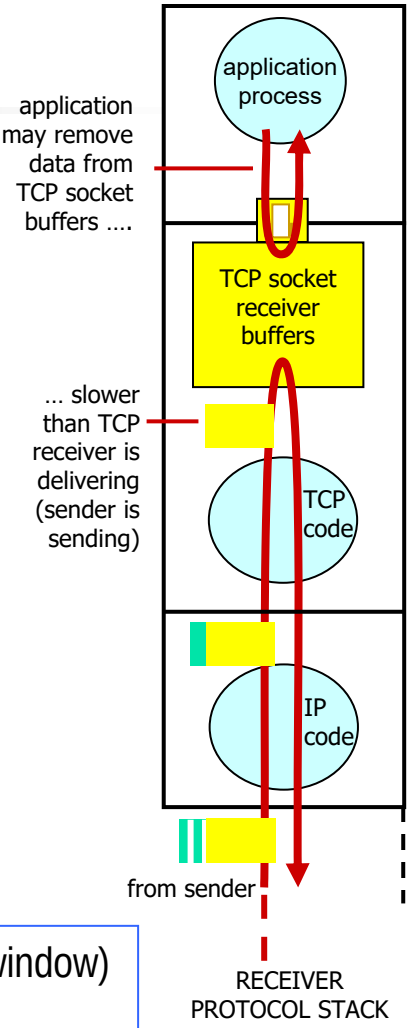
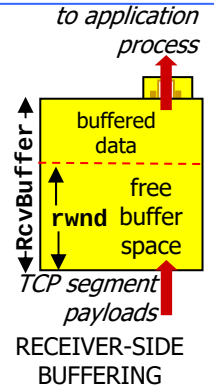


Figure 15.5 Example of Incorrect Duplicate Detection

TCP: flow control

- Reasons: Receiving transport entity itself cannot keep up with the flow of segments. Receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast
- Complex at the transport layer:
  - Considerable delay in the communication of flow control information
  - Amount of the transmission delay may be highly variable, making it difficult to effectively use a timeout mechanism for retransmission of lost data
- The receiving transport entity can
  1. Do nothing:
     - Segments that overflow the buffer are discarded
     - Sending transport entity will retransmit
  2. Refuse to accept further segments from the network service: Relies on network service to do the work
  3. Use a fixed sliding-window protocol: With a reliable network service this works quite well
  4. Use a credit scheme: A more effective scheme to use with an unreliable network service





RECEIVER-SIDE BUFFERING

- receiver "advertises" free buffer space by including rwnd (Rx window) value in TCP header of receiver-to-sender segments
- RcvBuffer size set via socket options (typical default is 4096 bytes)
- many operating systems autoadjust RcvBuffer
- sender limits amount of unacked ("in-flight") data to receiver's rwnd value
- guarantees receive buffer will not overflow

TCP: credit allocation, example

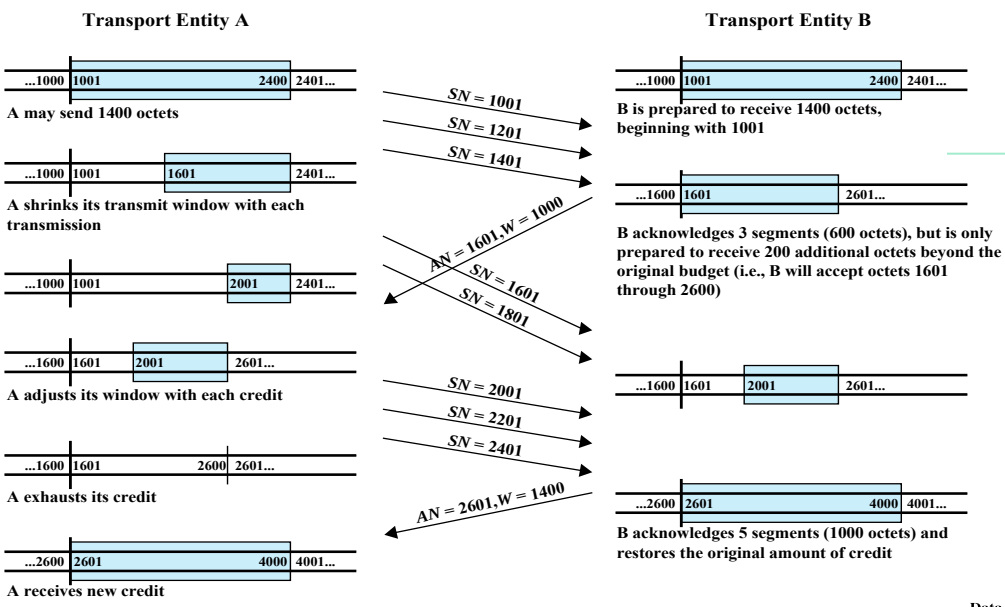


Figure 15.1 Example of TCP Credit Allocation Mechanism

in Figure 15.1, the first credit message implies that B has 1000 available octets in its buffer, and the second message that B has 1400 available octets.

- A conservative flow control scheme may limit the throughput of the transport connection in long-delay situations.
- The receiver could potentially increase throughput by optimistically granting credit for space it does not have. For example, if a receiver's buffer is full but it anticipates that it can release space for 1000 octets within a round-trip propagation time, it could immediately send a credit of 1000.
  - If the receiver can keep up with the sender, this scheme may increase throughput and can do no harm.
  - If the sender is faster than the receiver, however, some segments may be discarded, necessitating a retransmission. Because retransmissions are not otherwise necessary with a reliable network service (in the absence of internet congestion), an optimistic flow control scheme will complicate the protocol.

Initially, through the connection establishment process, the sending and receiving sequence numbers are synchronized and A is granted an initial credit allocation of 1400 octets, beginning with octet number 1001. The first segment transmitted by A contains data octets numbered 1001 through 1200. After sending 600 octets in three segments, A has shrunk its window to a size of 800 octets (numbers 1601 through 2400). After B receives these three segments, 600 octets out of its original 1400 octets of credit are accounted for, and 800 octets of credit are outstanding.

Now suppose that, at this point, B is capable of absorbing 1000 octets of incoming data on this connection. Accordingly, B acknowledges receipt of all octets through 1600 and issues a credit of 1000 octets. This means that A can send octets 1601 through 2600 (5 segments).

However, by the time that B's message has arrived at A, A has already sent two segments, containing octets 1601 through 2000 (which was permissible under the initial allocation). Thus, A's remaining credit upon receipt of B's credit allocation is only 600 octets (3 segments). As the exchange proceeds, A advances the trailing edge of its window each time that it transmits and advances the leading edge only when it is granted credit.

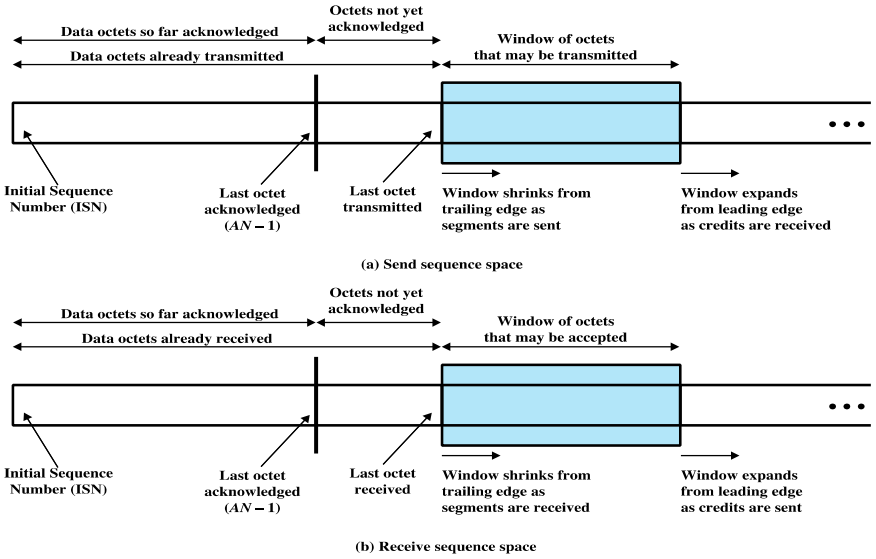
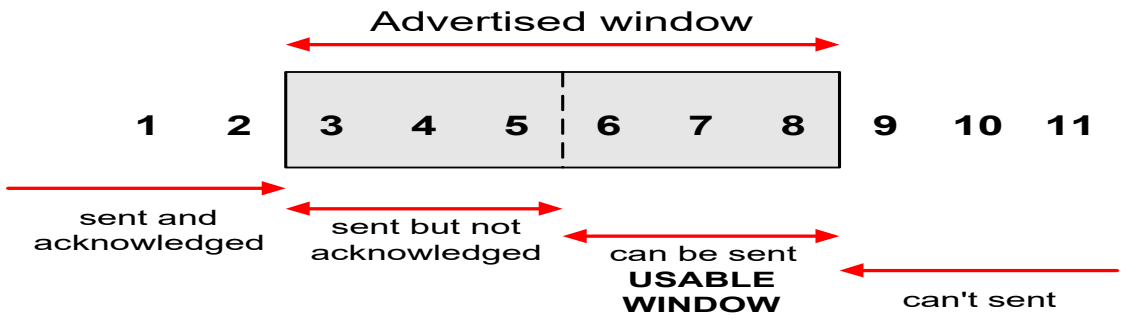


Figure 15.2 Sending and Receiving Flow Control Perspectives

Flow Control

- TCP implements sliding window flow control
  - Sending acknowledgements is separated from setting the window size at sender.
  - Acknowledgements do not automatically increase the window size
  - Acknowledgements are cumulative
- Sliding Window Protocol is performed at the byte level:



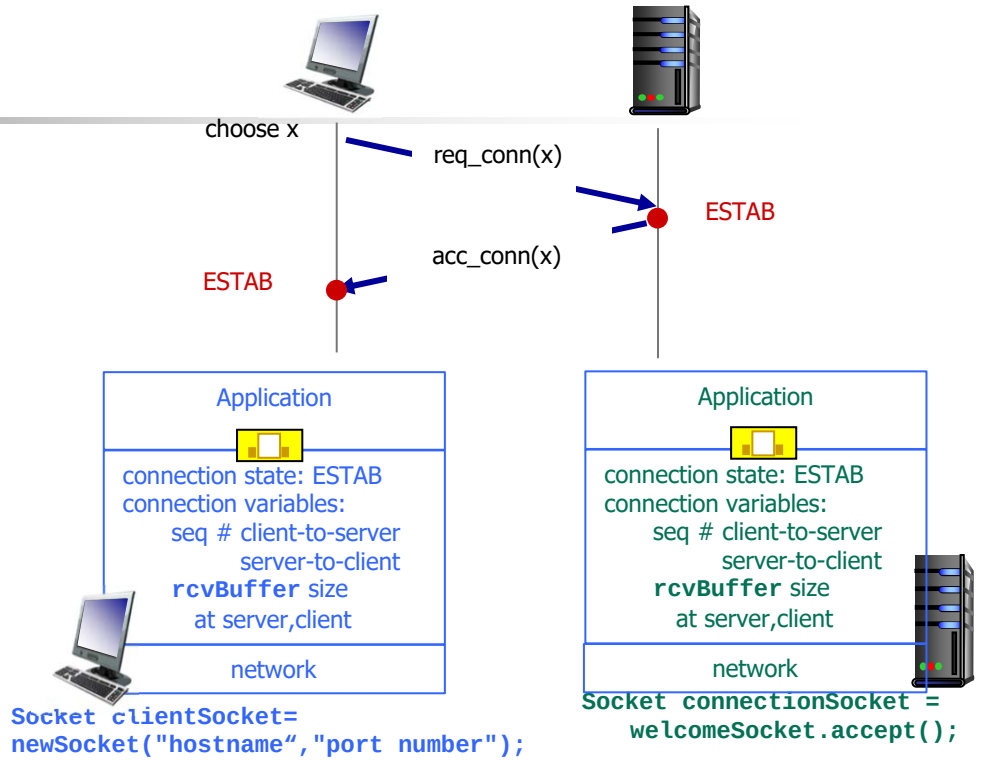
- Future acknowledgments will resynchronize the protocol if an ACK/CREDIT segment is lost
- If no new acknowledgments are forthcoming the sender times out and retransmits a data segment which triggers a new acknowledgment
- Still possible for deadlock to occur

Connection Establishment

before exchanging data, sender/receiver "handshake" serves three main purposes:

- Allows each end to assure that the other exists
- Allows exchange or negotiation of optional parameters
- Triggers allocation of transport entity resources

- Connection establishment must take into account the unreliability of a network service
  - will 2-way handshake always work in network?
  - variable delays
  - retransmitted messages (e.g. req_conn(x)) due to message loss
  - message reordering
  - can't "see" other side
  - Calls for the exchange of SYNs (2-way handshake) could result in: Duplicate SYNs, data segments



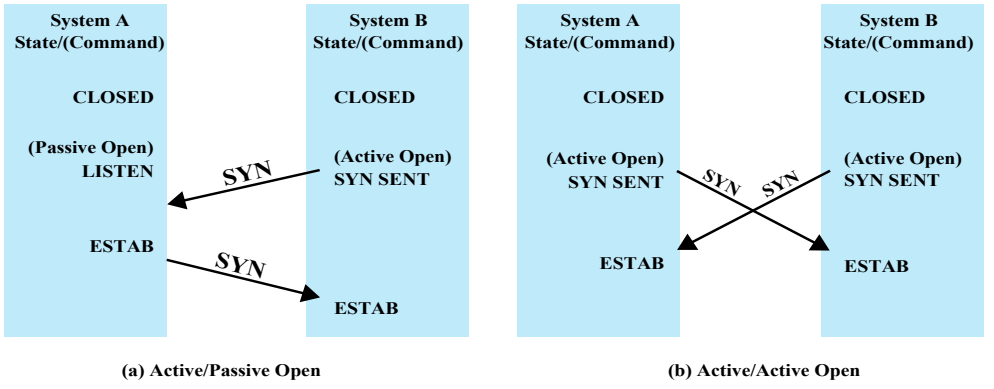


Figure 15.4 Connection Establishment Scenarios

2-way handshake failure scenarios

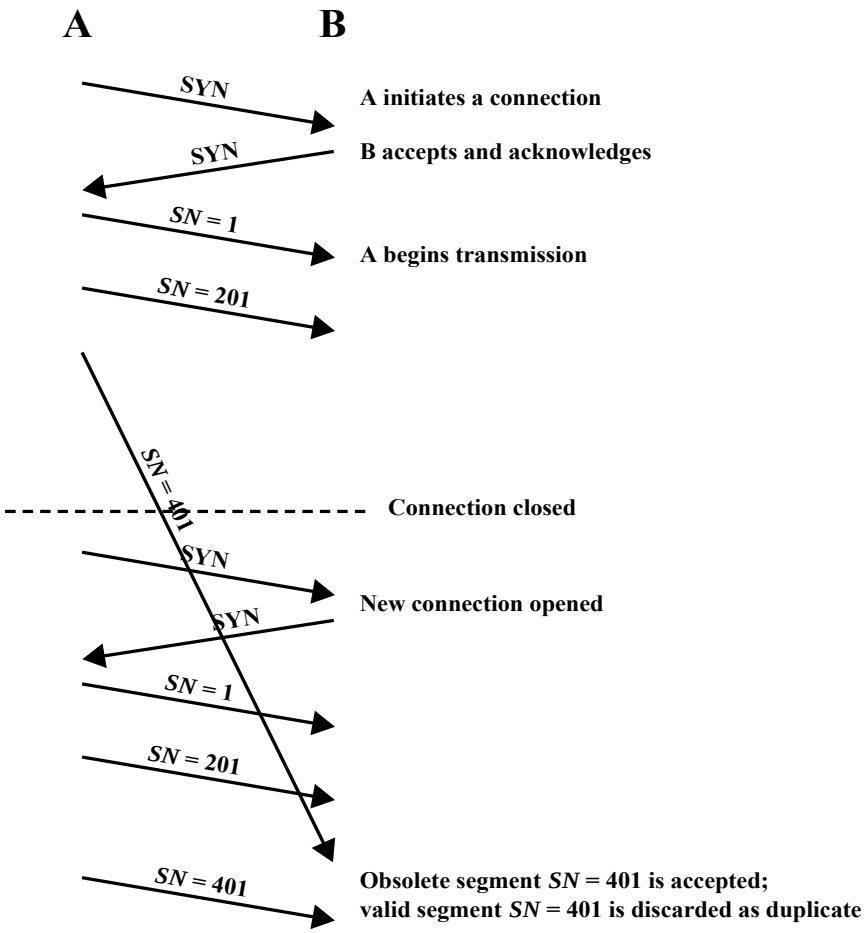


Figure 15.6 The Two-Way Handshake: Problem with Obsolete Data Segment

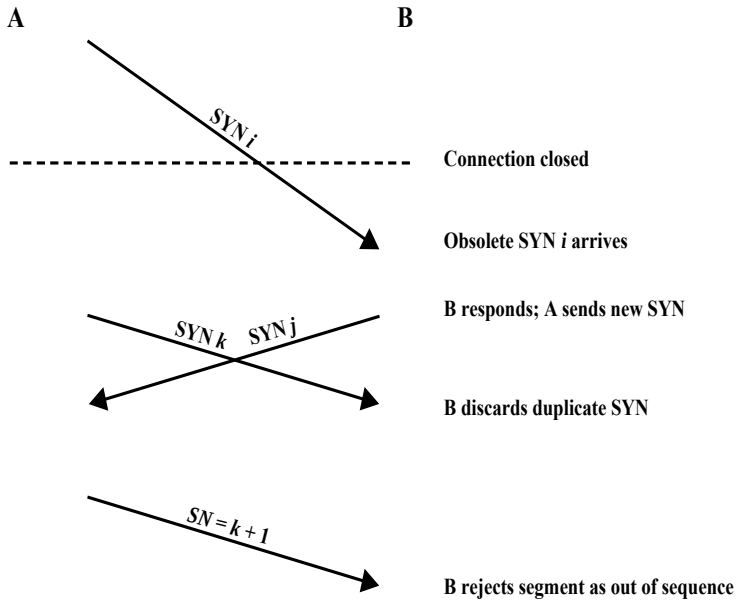
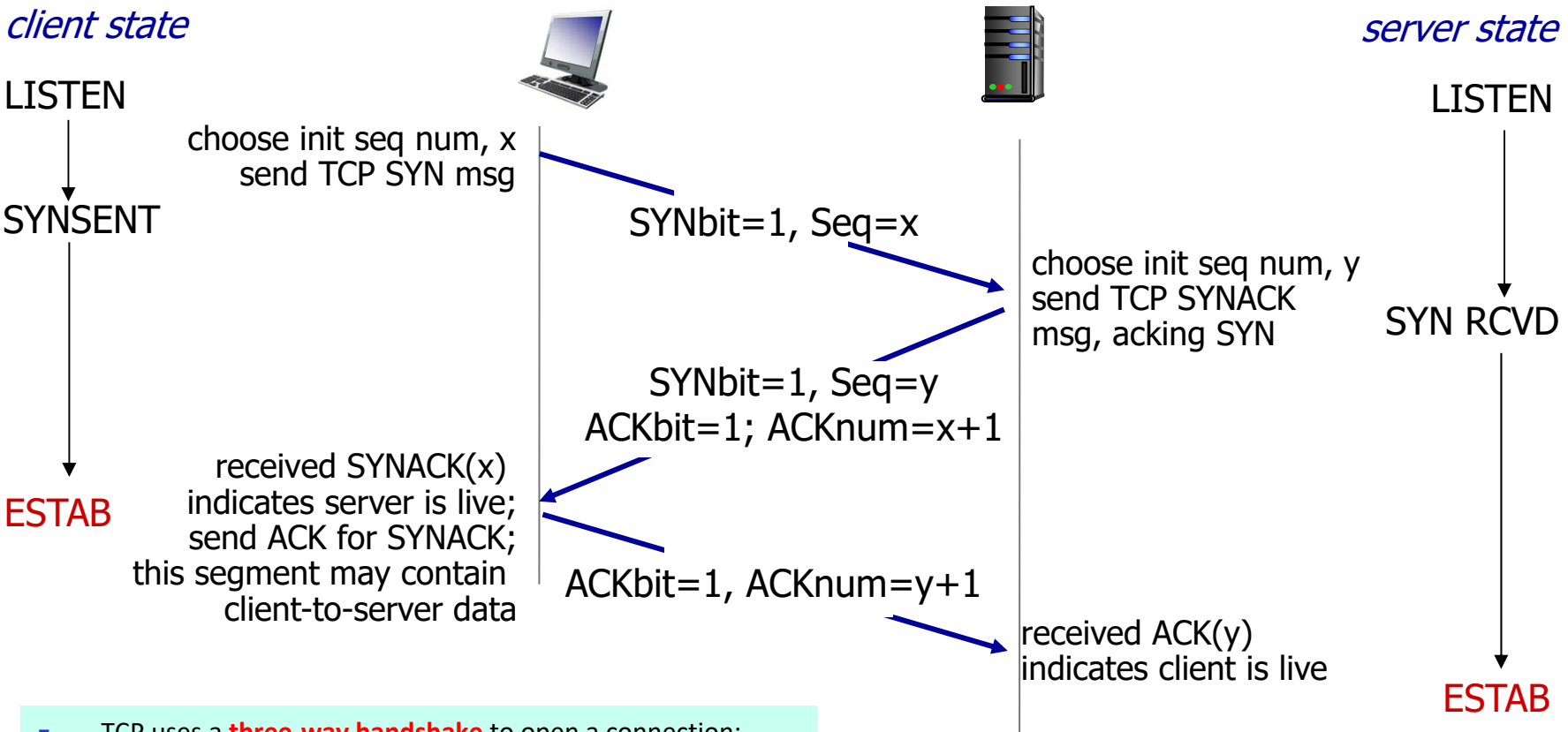


Figure 15.7 Two-Way Handshake: Problem with Obsolete SYN Segments

Connection Establishment: TCP 3-way handshake

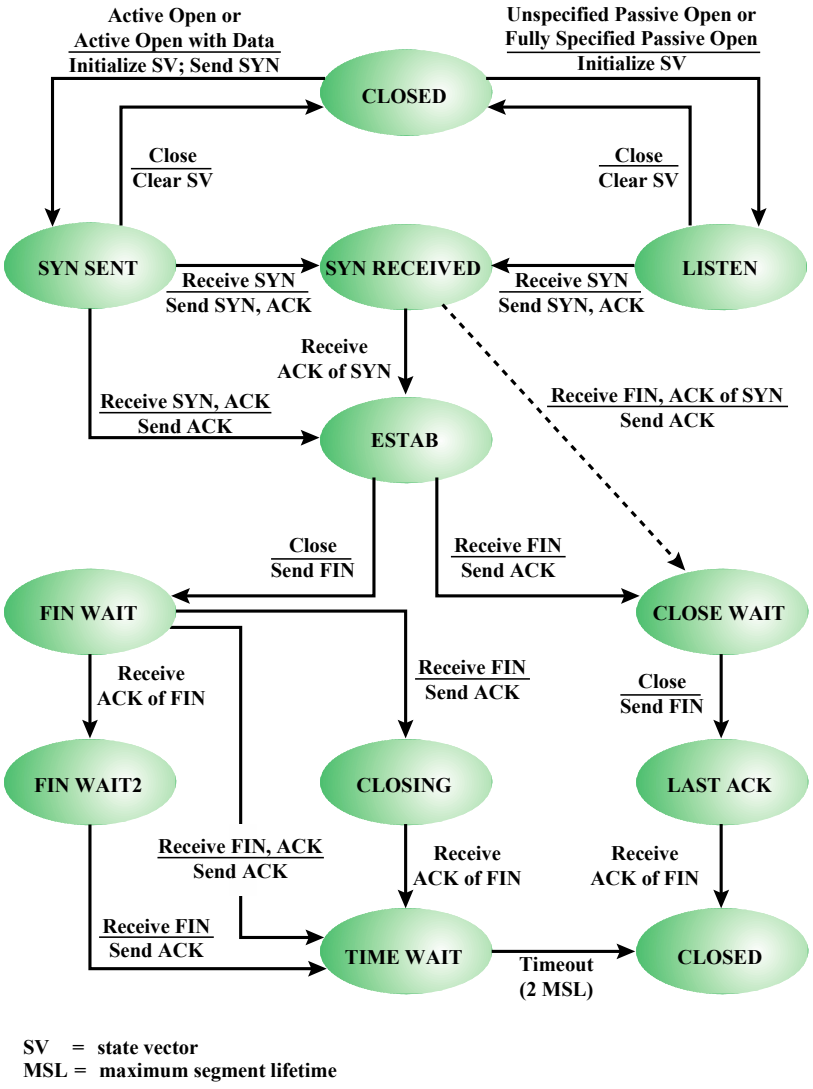


- TCP uses a **three-way handshake** to open a connection:
  (1) ACTIVE OPEN: Client sends a segment with
  - SYN bit set *
  - port number of client
  - initial sequence number (ISN) of client

  (3) Client acknowledges by sending a segment with:
  - ACK ISN of server

(2) PASSIVE OPEN: Server responds with a segment with
- SYN bit set (* counts as one byte)
- initial sequence number of server
- ACK for ISN of client

three-way handshake



SV = state vector
MSL = maximum segment lifetime

Figure 15.8 TCP Entity State Diagram

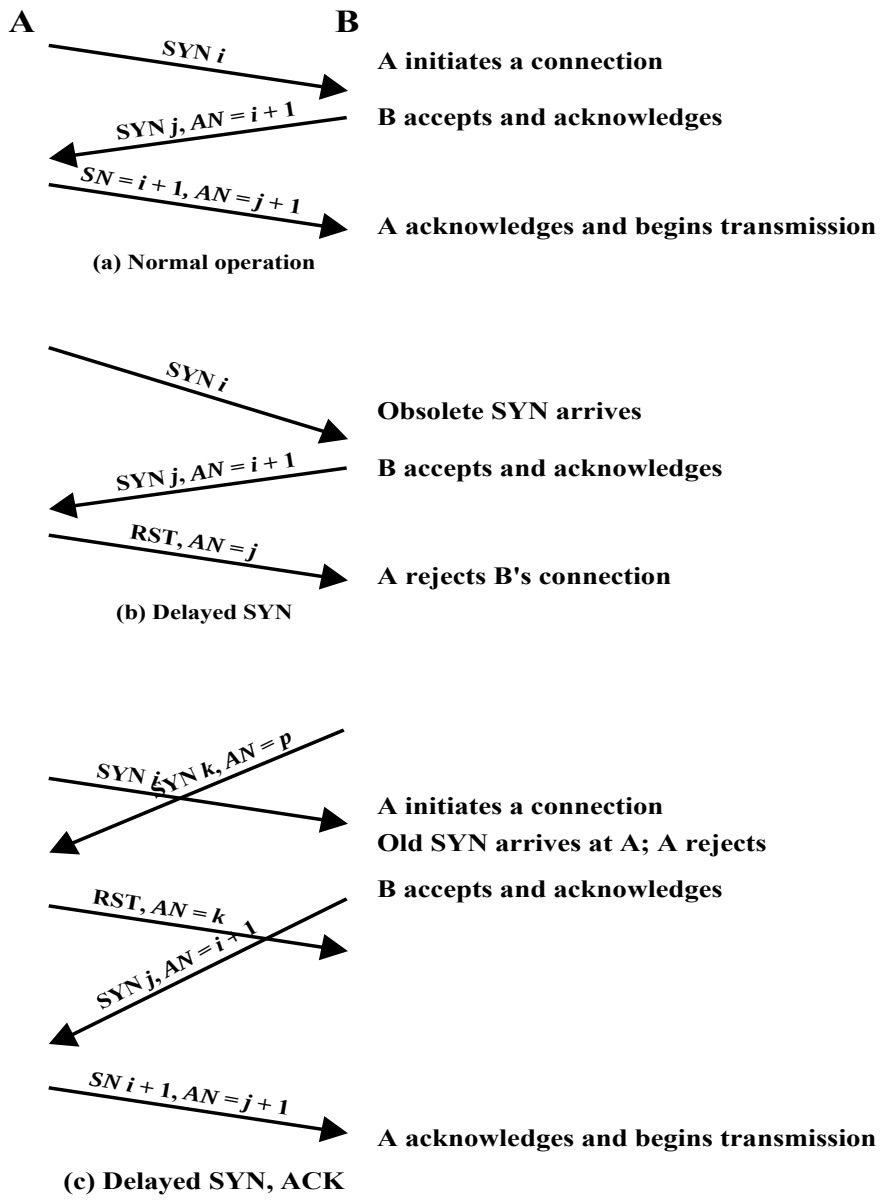
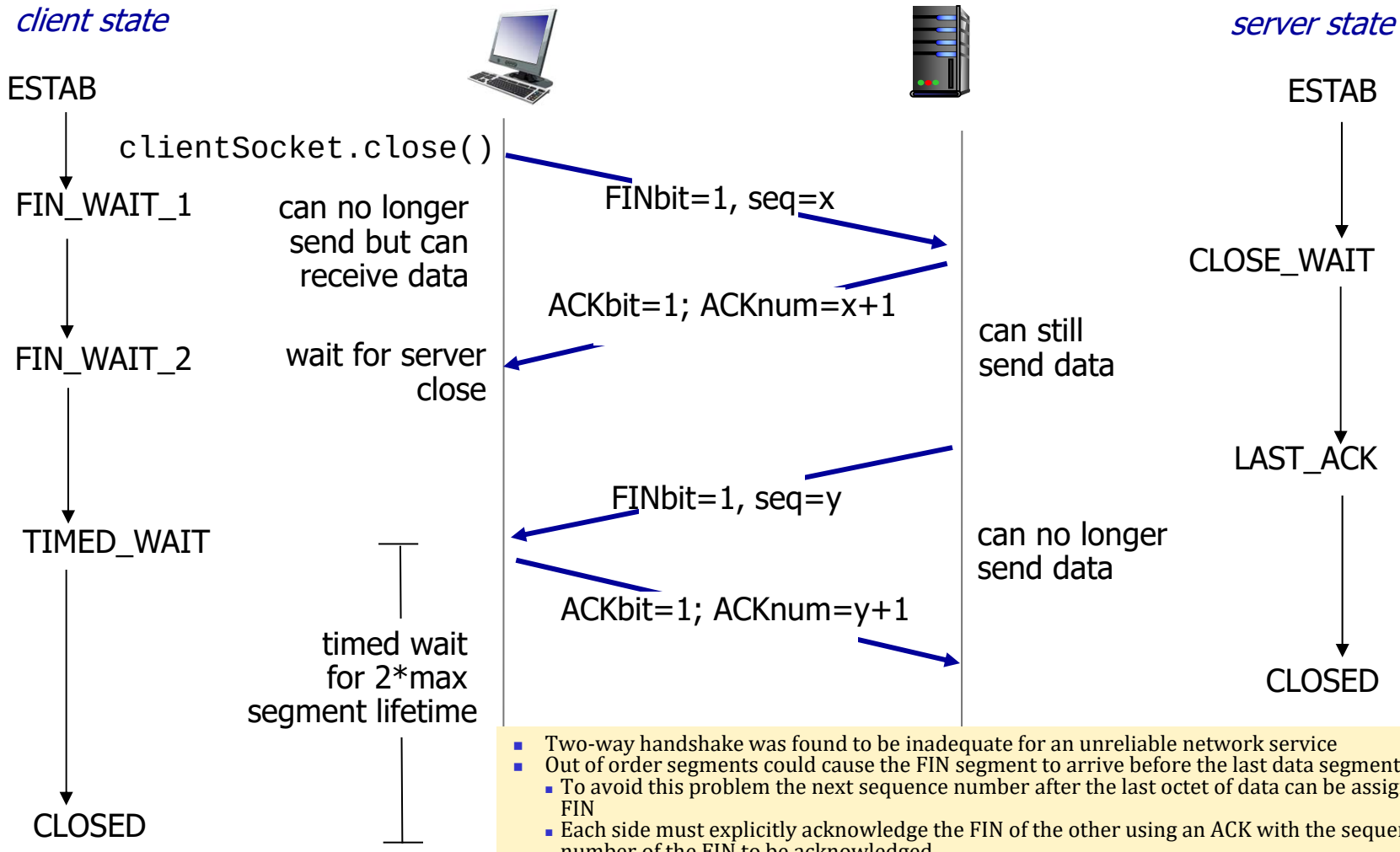


Figure 15.9 Examples of Three-Way Handshake

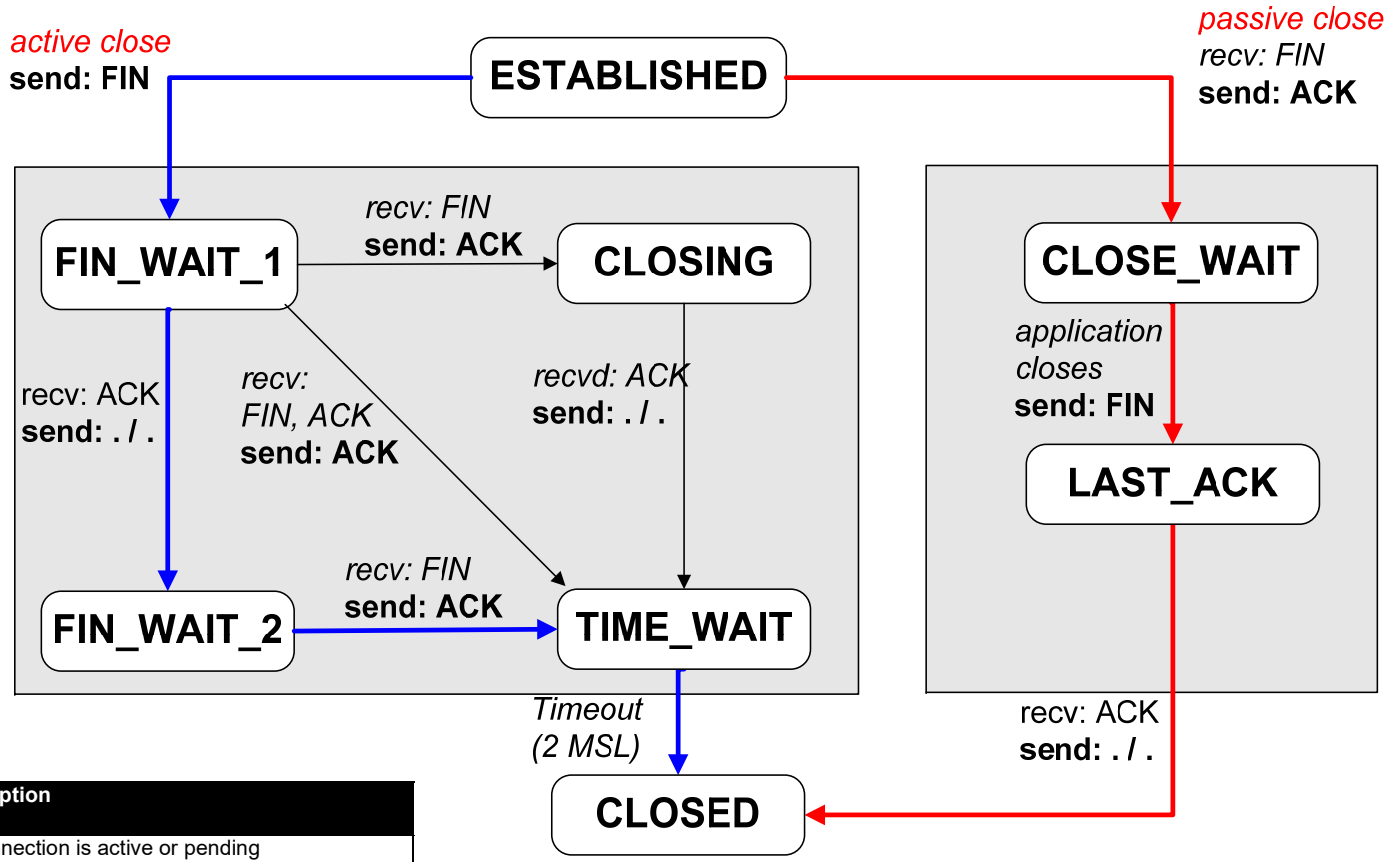
TCP: closing a connection



- Two-way handshake was found to be inadequate for an unreliable network service
- Out of order segments could cause the FIN segment to arrive before the last data segment
  - To avoid this problem the next sequence number after the last octet of data can be assigned to FIN
  - Each side must explicitly acknowledge the FIN of the other using an ACK with the sequence number of the FIN to be acknowledged

- client, server each close their side of connection: send TCP segment with FIN bit = 1
- respond to received FIN with ACK: on receiving FIN, ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled

TCP: closing a connection, State Transition Diagram



State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for Ack
SYN SENT	The client has started to open a connection
ESTABLISHED	Normal data transfer state
FIN WAIT 1	Client has said it is finished
FIN WAIT 2	Server has agreed to release
TIMED WAIT	Wait for pending packets (“2MSL wait state”)
CLOSING	Both Sides have tried to close simultanesously
CLOSE WAIT	Server has initiated a release
LAST ACK	Wait for pending packets

Failure Recovery

- When the system that the transport entity is running on fails and subsequently restarts, the state information of all active connections is lost
  - Affected connections become half open because the side that did not fail does not realize the problem
    - Still active side of a half-open connection can close the connection using a ***keepalive*** timer
- In the event that a transport entity fails and quickly restarts, half-open connections can be terminated more quickly by the use of the RST segment
  - Failed side returns an RST *i* to every segment *i* that it receives
  - RST *i* must be checked for validity on the other side
    - If valid an abnormal termination occurs
- There is still the chance that some user data will be lost or duplicated

Congestion & congestion control

- congestion occurs when the number of packets being transmitted through a network begins to approach the packet-handling capacity of the network:"too many sources sending too much data too fast for network to handle"
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- The objective of congestion control is to maintain the number of packets within the network below the level at which performance falls off dramatically.

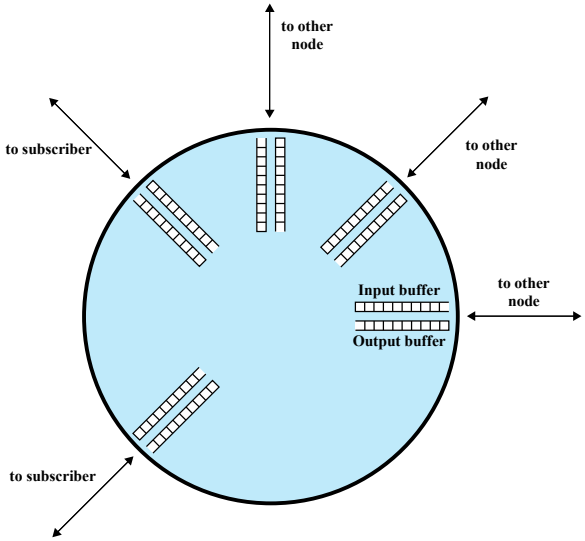


Figure 20.1 Input and Output Queues at Node

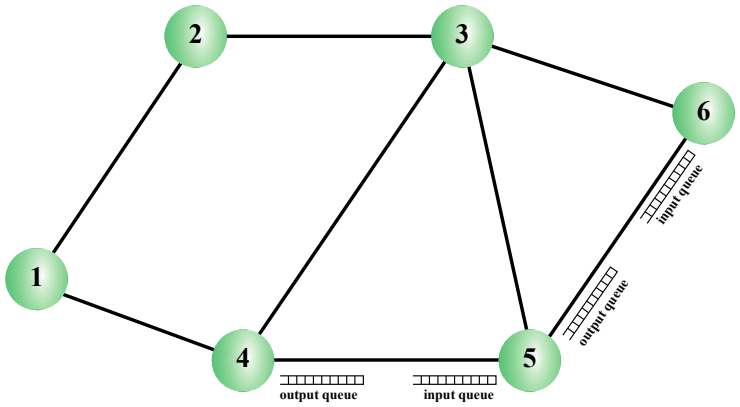
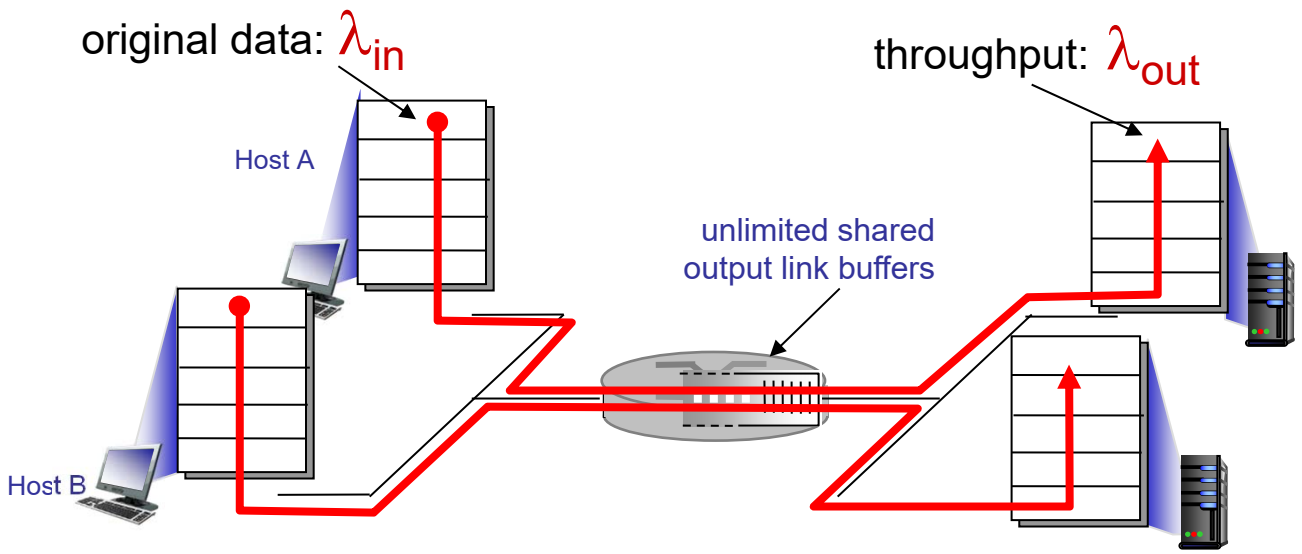


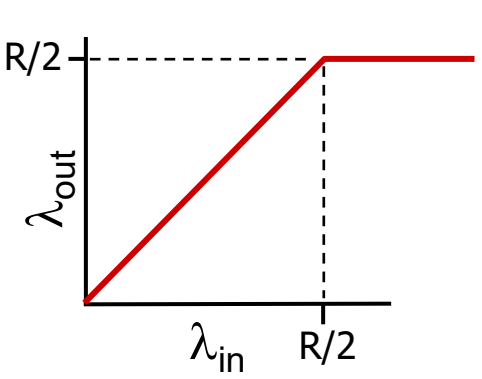
Figure 20.2 Interaction of Queues in a Data Network

Causes/costs of congestion: scenario 1

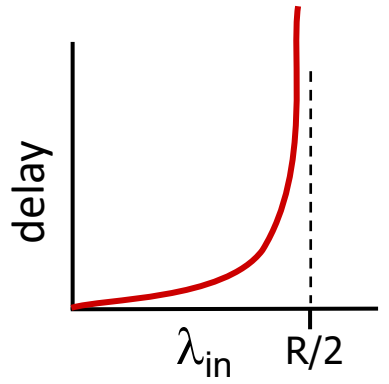
scenario 1:

- two senders,
- two receivers
- one router, infinite buffers
- output link capacity: R
- **no** retransmission





maximum per-connection throughput: R/2

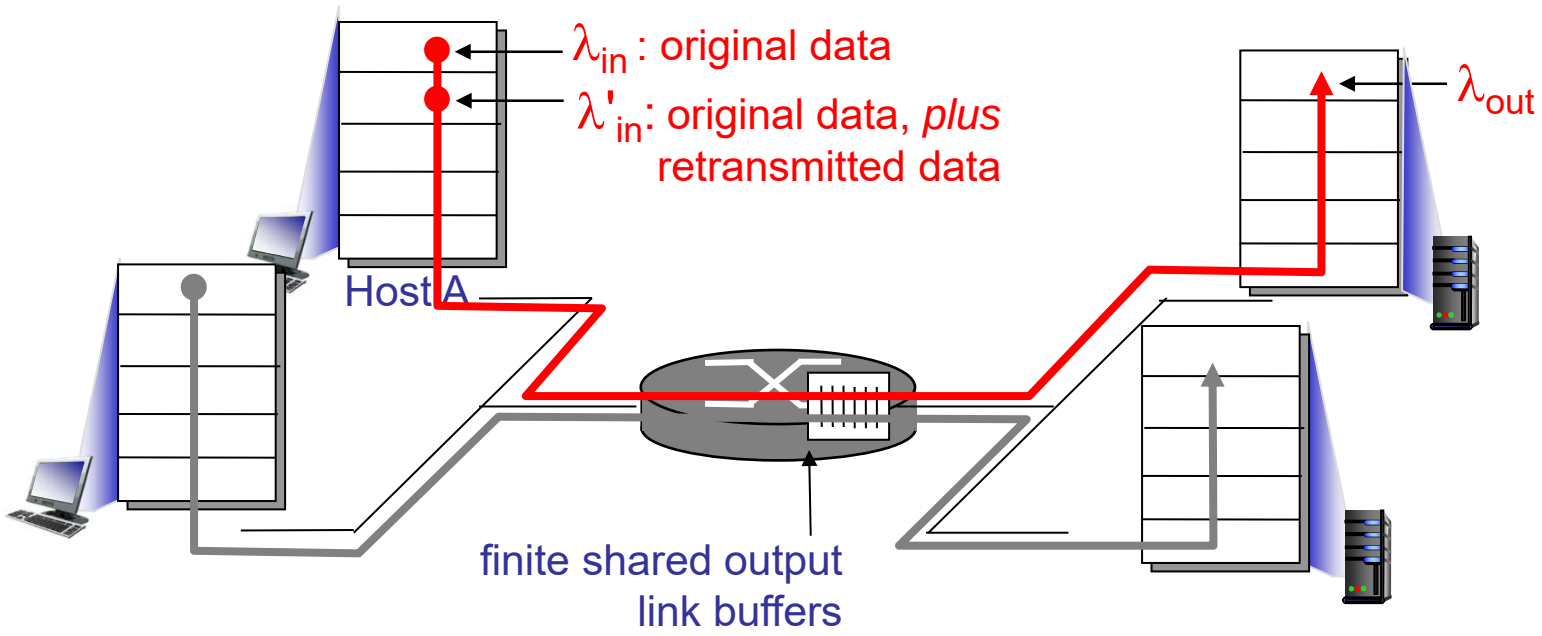


large delays $\rightarrow\infty$ as arrival rate, λ_{in}, approaches capacity

Causes/costs of congestion: scenario 2

scenario 2:

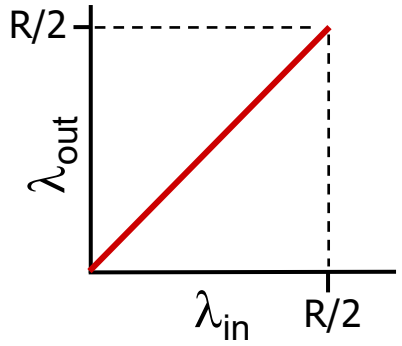
- one router, *finite* buffers
- sender retransmission of *timed-out* packet
  - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
  - transport-layer input includes *retransmissions* : $\lambda'_{in} \geq \lambda_{in}$

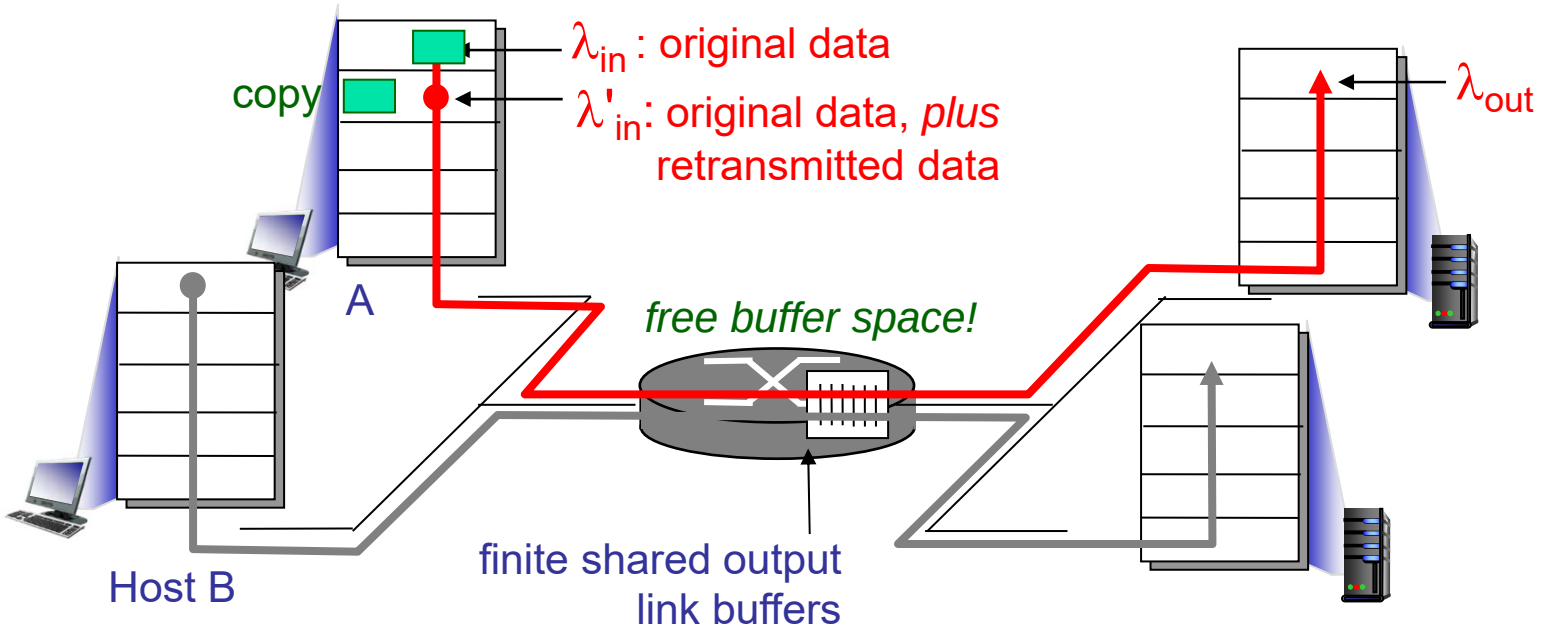


Causes/costs of congestion: scenario 2a

scenario 2a (ideal): sender has perfect knowledge of router buffer status

- sender sends *only* when router buffers *available* → **no** retransmission needed, $\lambda'_{in} = \lambda_{in}$

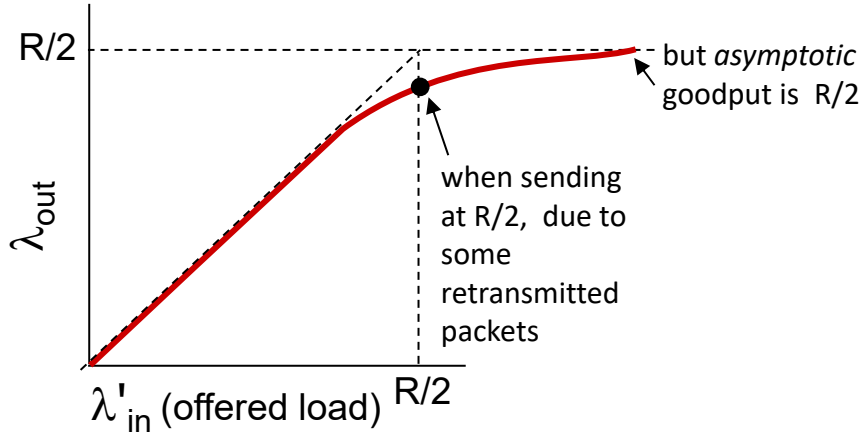


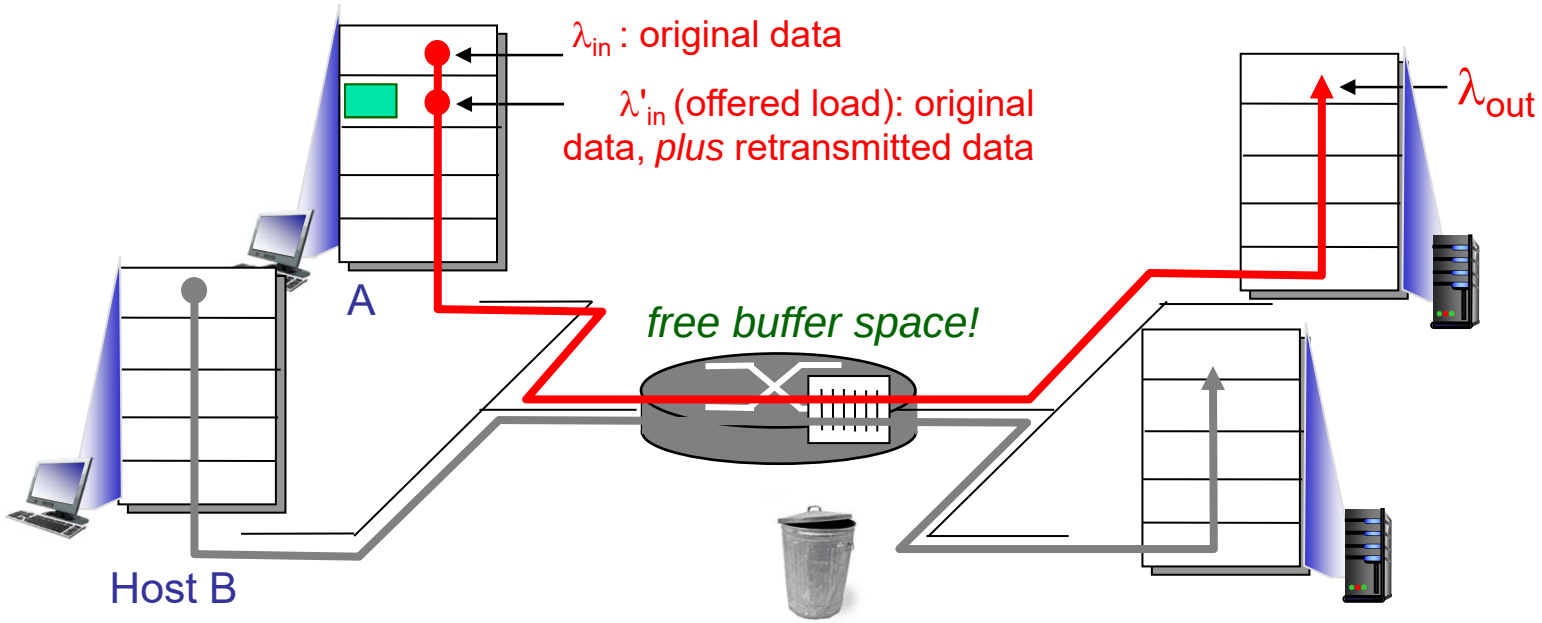


Causes/costs of congestion: scenario 2b

scenario 2b (rather ideal):

- *known loss* packets can be lost, dropped at router due to full buffers
- sender only resends if packet *known* to be lost

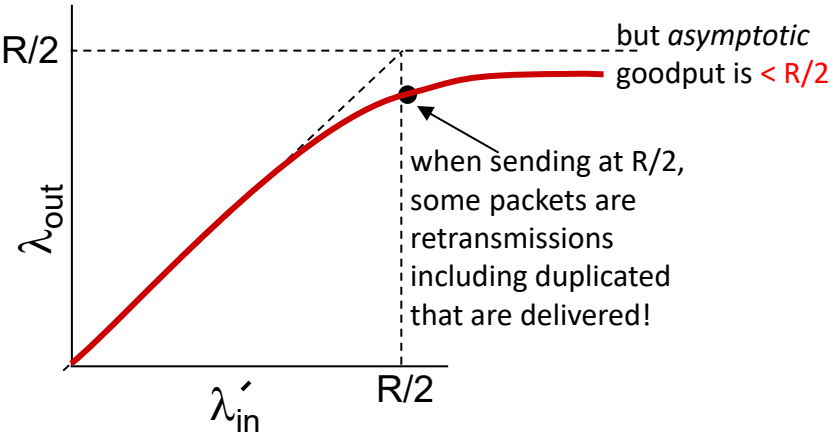


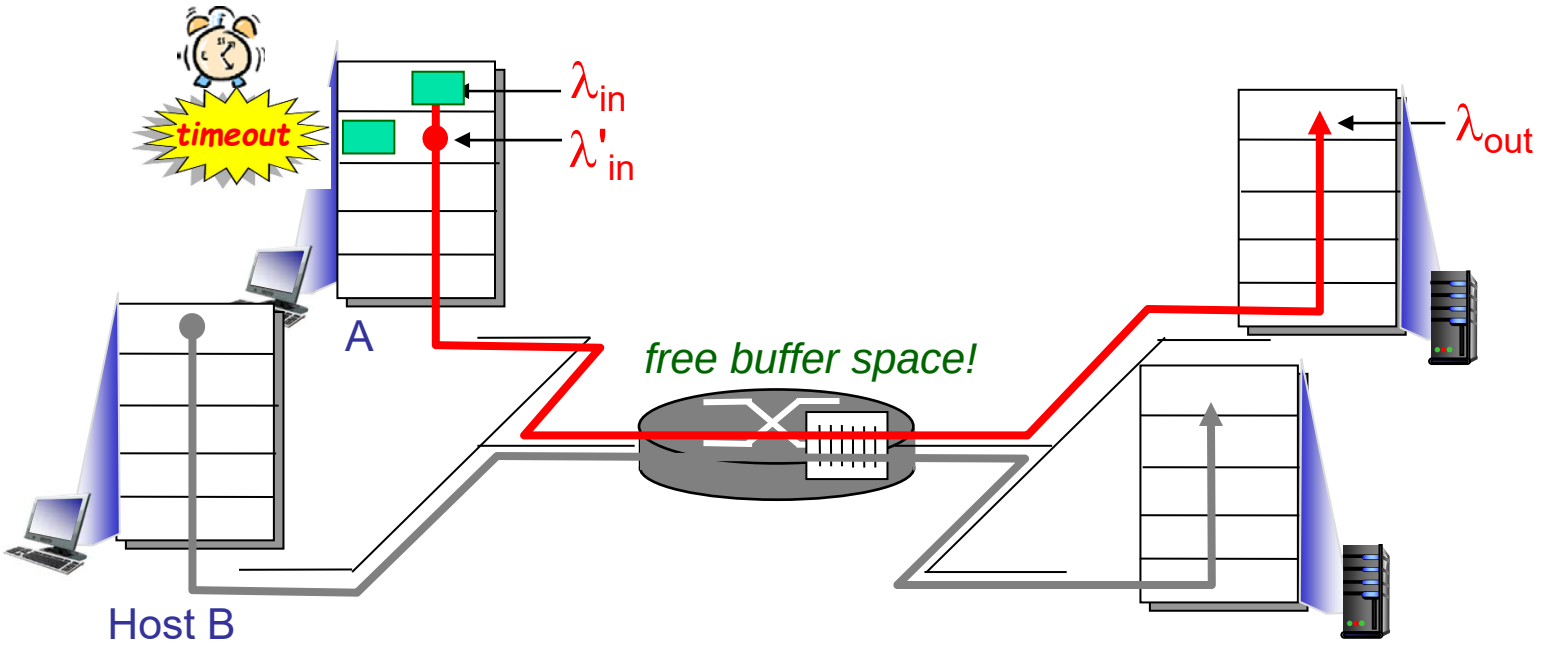


Causes/costs of congestion: scenario 2c

scenario 2c (*realistic):*

- packets can be lost, dropped at router due to full buffers
- sender also re-transmits if time-out
- Possible problem: sender times out prematurely, sending *two* copies, both of which are delivered
- "costs" of congestion: decreasing goodput

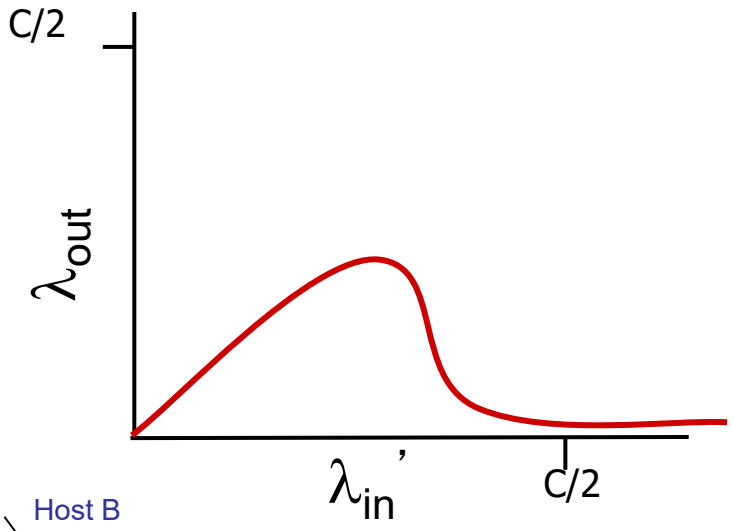


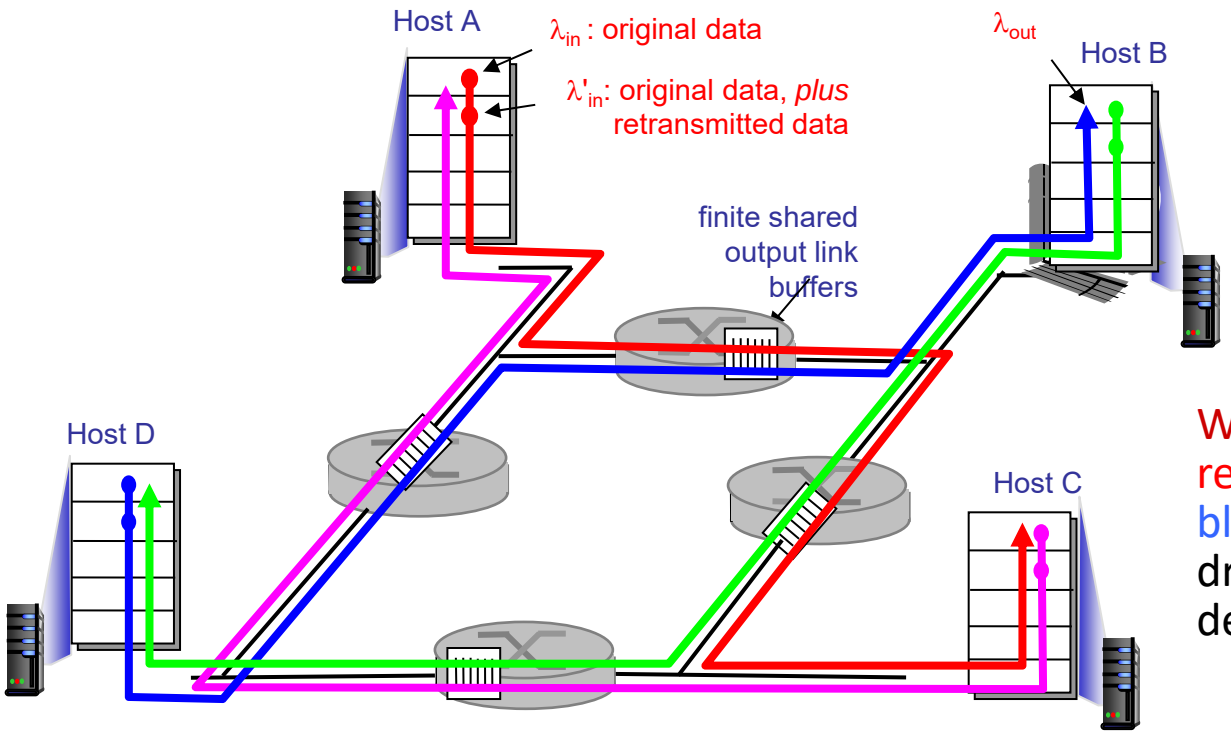


Causes/costs of congestion: scenario 3

scenario 3:

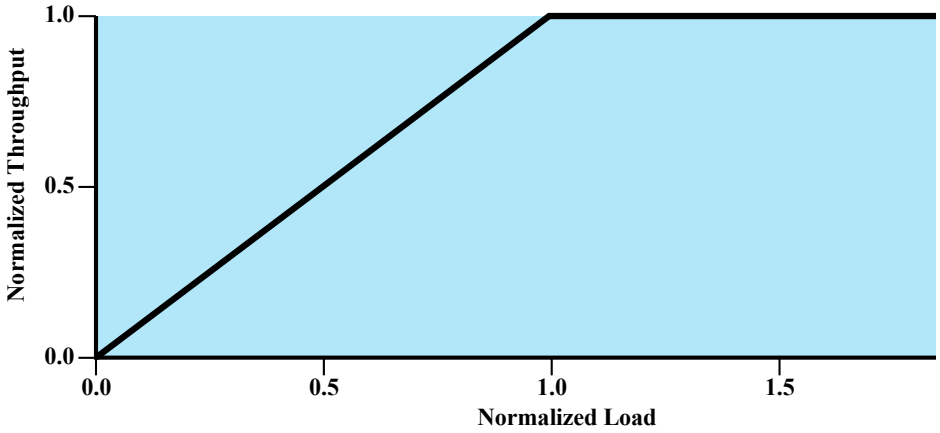
- four senders
- multihop paths
- Time-out/retransmit





When red offered load λ'_{in} of red flow increases, all arriving blue pkts at upper queue are dropped, blue throughput decreased

Effects of congestion:



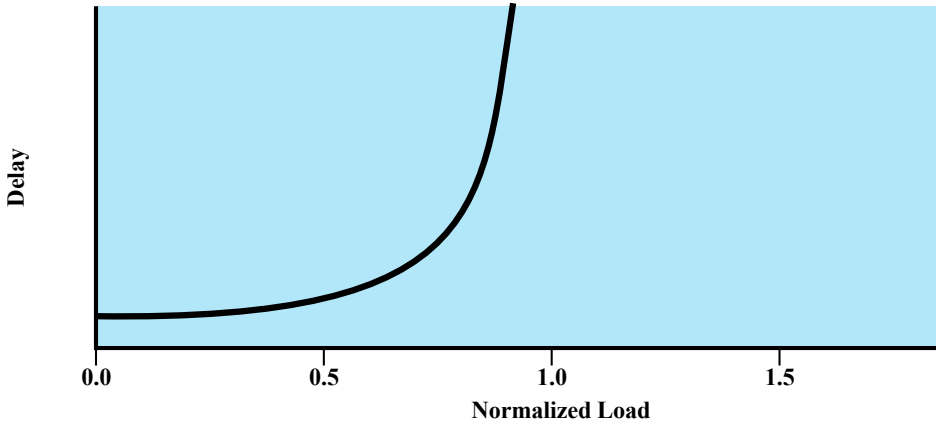
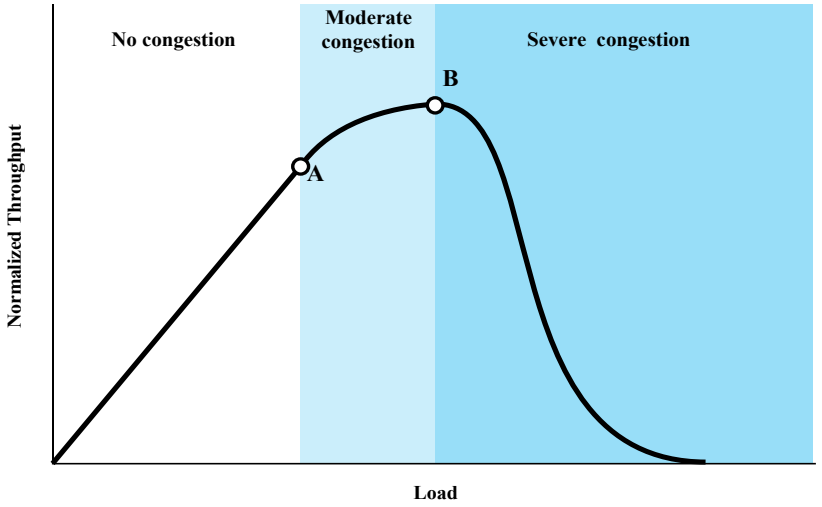


Figure 20.3 Ideal Network Utilization



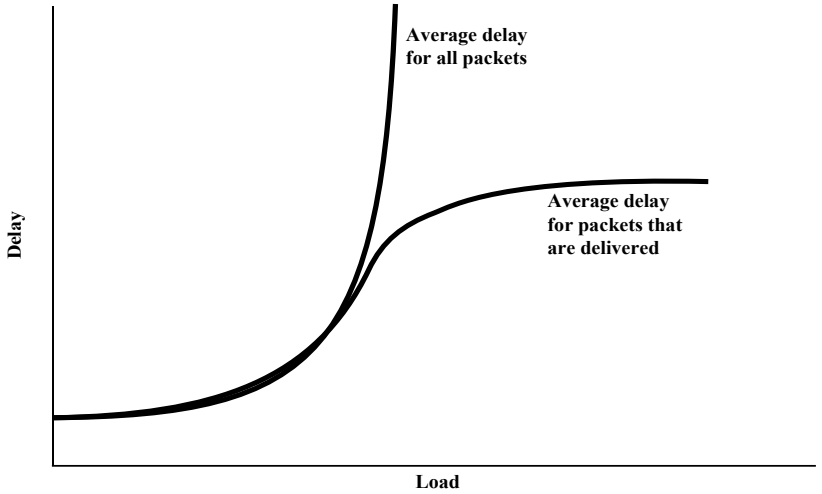
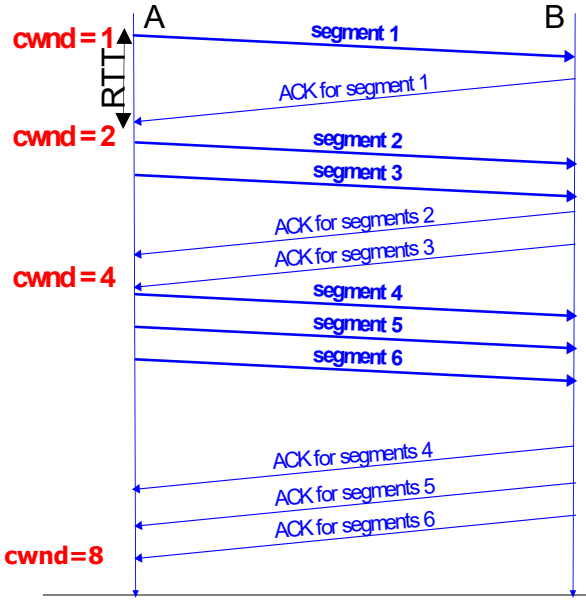


Figure 20.4 The Effects of Congestion

TCP Congestion Control

- TCP transmission is constrained by **awnd** = min[**credit, cwnd**]
- **awnd** = allowed window (i.e., number of segments that TCP is currently allowed to send without receiving further ACK's)
- **cwnd** = congestion window (A window determined by TCP during startup and reduced during periods of congestion)
- **credit** = the amount of unused credit granted in the most recent ACK, in segments.
- TCP congestion control mechanism is implemented at the sender using two parameters: **cwnd** and **Slow-start threshhold Value** (**ssthresh):** Initial value =advertised window size
- Congestion control works in two modes:
  - **slow start** (when cwnd < ssthresh)
  - **congestion avoidance** (when cwnd ≥ ssthresh)



slow start (when cwnd < ssthresh):

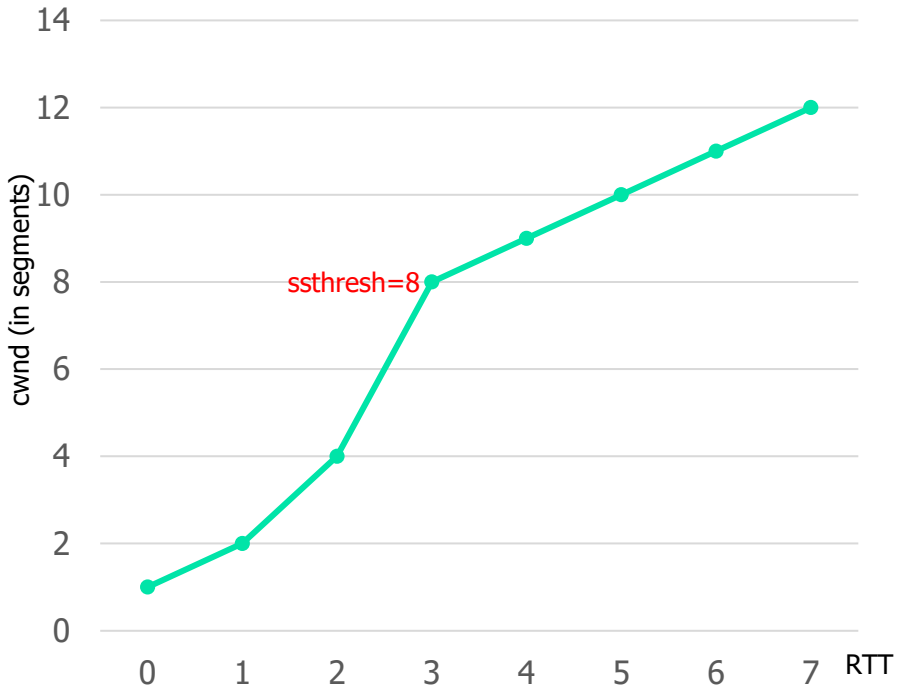
- Initial value: Set **cwnd = 1** MSS

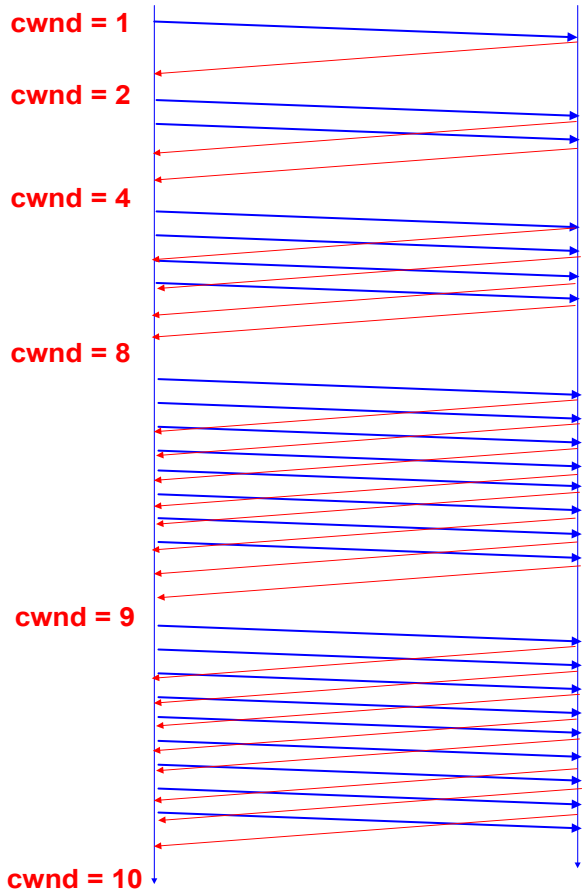
  (Unit is a segment size. TCP actually is based on bytes and increments by 1 MSS (maximum segment size)
- The TCP receiver sends an ACK for every other segment.
- Each time an ACK for 1 (even a segment smaller than MSS) or more segments is received by the sender, **cwnd = cwnd X 2**
  - i.e., cwnd is actually exponentially increased (doubled) until first **loss event**: initial rate is slow but ramps up exponentially fast

Congestion Avoidance

- Congestion avoidance phase is started if cwnd has reached the slow-start threshold (ssthresh) value
- If cwnd ≥ ssthresh then each time an ACK is received, increment cwnd by 1 only if *all cwnd* segments have been acknowledged

Slow Start/Congestion Avoidance: Example with ssthresh=8





Responses to Congestion

- TCP assumes there is congestion if it detects a segment loss
- A TCP sender can detect lost segments via:
  - Timeout of a retransmission timer
  - Receipt of a triple duplicate ACK
- Under congestion, the sender
  - sets ssthresh = *current* cwnd/2
    and then cwnd = 1
  - and enters slow-start mode

Some TCP congestion control versions:

- **TCP Tahoe** (1988)
  - Slow Start
  - Congestion Avoidance
  - Fast Retransmit
- **TCP Reno** (1990)
  - Fast Recovery

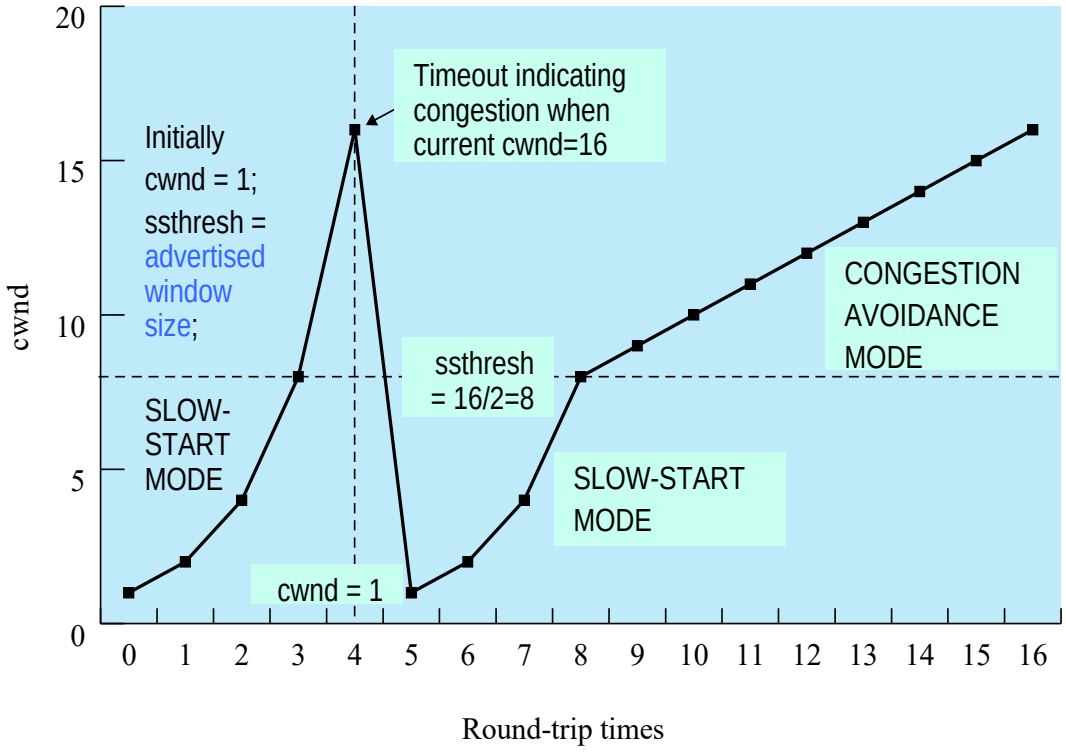


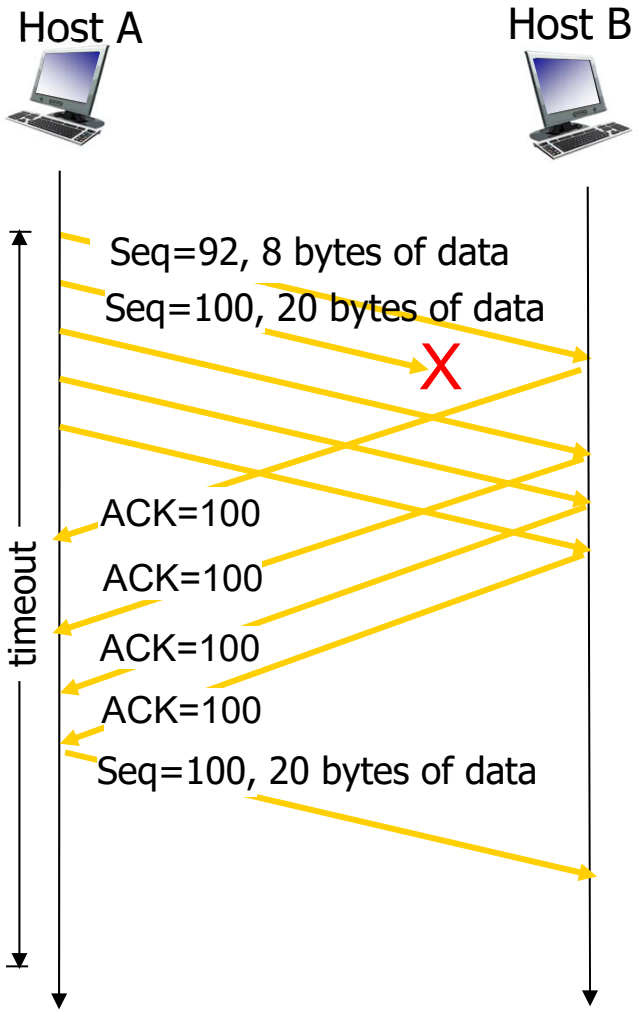
Figure 20.10 Illustration of Slow Start and Congestion Avoidance

TCP fast retransmit

- time-out period often relatively long:
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs.

TCP fast retransmit

- if sender receives 3 (or more) ACKs for same data ("triple duplicate ACKs"), the TCP sender
  - believes that a segment has been lost, and
  - resends unacked segment with smallest seq # *without waiting for a timeout to happen* (i.e., likely that unacked segment lost, so don't wait for timeout), then
  - enters slow start mode with
    - ssthresh = cwnd/2
    - cwnd = 1



fast retransmit after sender receipt of triple duplicate ACK

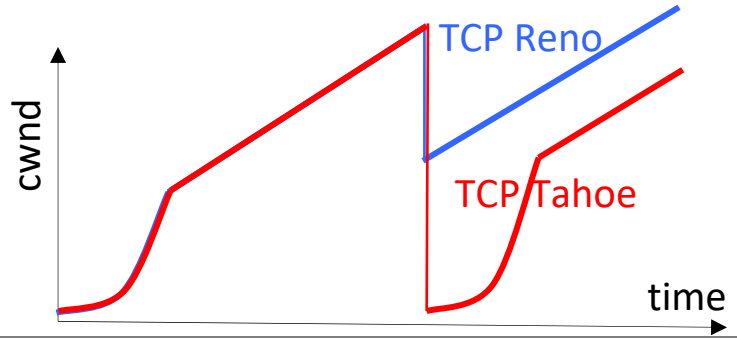
Fast Recovery (in TCP Reno)

- Fast recovery avoids slow start after a fast retransmit as an improvement that allows high throughput under *moderate* congestion, especially for large windows.
- **Reason:** the receipt of the duplicate ACKs tells the TCP sender: Because the receiver can only generate the duplicate ACK, when another segment is received that segment has left the network and is in the receiver's buffer, i.e., there is still data flowing between the two ends, and TCP does not want to reduce the flow abruptly by going into slow start.
- The fast retransmit and fast recovery algorithms are usually implemented together as follows:
  - After reception of 3 duplicate ACKs, retransmit the "lost segment", and set:
    - ssthresh = cwnd/2
    - cwnd = ssthresh+3. (3 segments left the network and cached by the other end has)

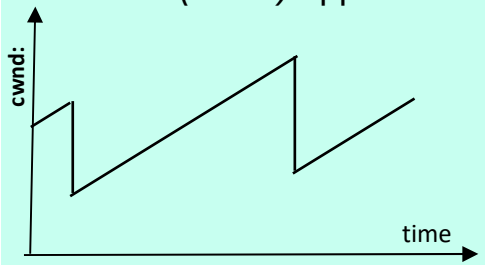
    Enter congestion avoidance mode
  - Increment cwnd by one for each received additional duplicate ACK.

    Transmit a segment, if allowed by the new value of cwnd
  - When the next ACK arrives that acknowledges "new data", set: cwnd=ssthresh

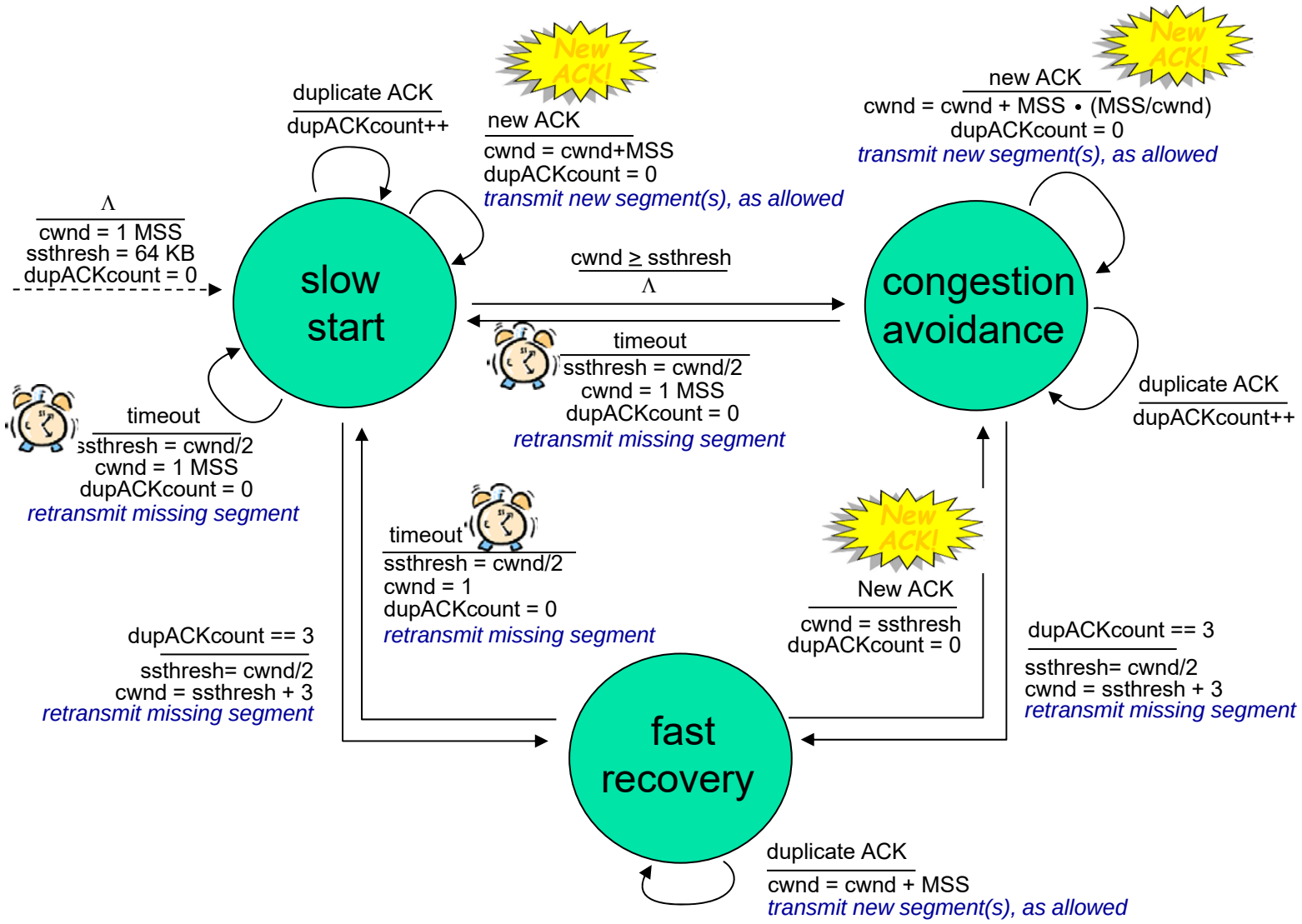
    Enter congestion avoidance mode
- TCP Reno improves upon TCP Tahoe when a single packet is dropped in a RTT



TCP congestion control: *Additive Increase Multiplicative Decrease (AIMD)* approach: sender *keeps* additively increasing transmission rate (window size), probing for usable bandwidth, until loss occurs then cut window in half (i.e., *multiplicative decrease*)



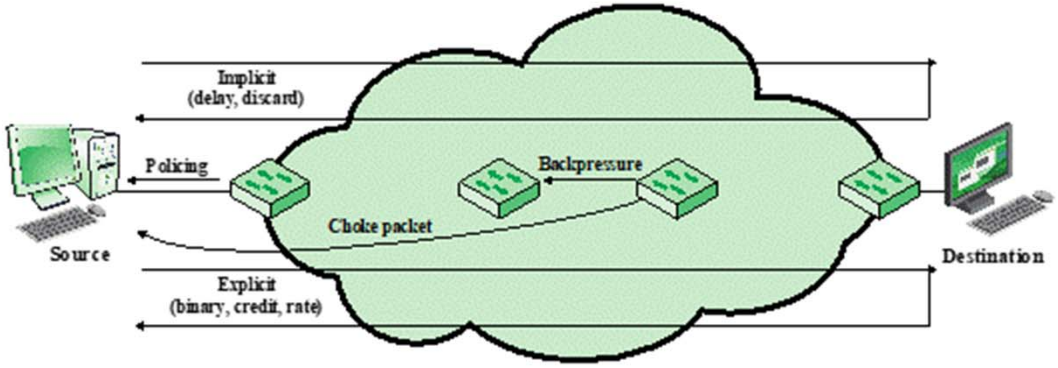
Summary: TCP Congestion Control



Congestion Control Mechanism: Backpressure & Choke Packet

Backpressure:

- If node becomes congested it can slow down or stop flow of packets from other nodes
- Can be exerted on the basis of links or logical connections
- Flow restriction propagates backward to sources, which are restricted in the flow of new packets into the network
- Can be selectively applied to logical connections so that the flow from one node to the next is only restricted or halted on some connections



Choke Packet:

- A control packet
  - Generated at congested node
  - Sent back to source node
- ➢ An example is the Internet Control Message Protocol (ICMP) Source Quench packet
  - From router or destination end system
  - Source cuts back until it no longer receives quench messages
  - Message is issued for every discarded packet
  - Message may also be issued for anticipated congestion
- Is a crude technique for controlling congestion

Congestion Control Mechanism: Implicit & Explicit Congestion Signaling

Implicit Congestion Signaling:

- With network congestion: Transmission delay increases & Packets may be discarded
- Source can detect congestion and reduce flow: Responsibility of end systems
- Effective on *connectionless* (datagram) networks
- Also used in connection-oriented networks: LAPF control is capable of detecting lost frames

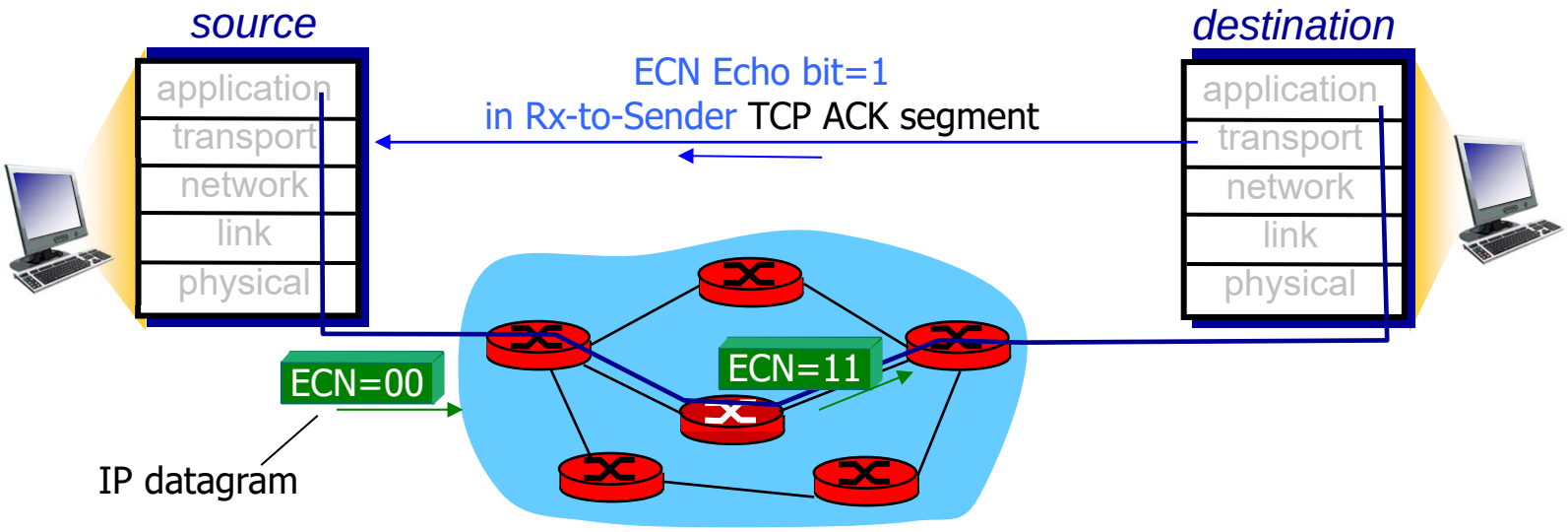
Explicit Congestion Signaling:

- Network alerts end systems of increasing congestion
- End systems take steps to reduce offered load
- Backward: Congestion avoidance notification in opposite direction to packet required
- Forward: Congestion avoidance notification in same direction as packet required
- Explicit Signaling Categories:
  - Binary: A bit set in a packet indicates congestion
  - Credit based: Indicates how many packets source may send (Common for end-to-end flow control)
  - Rate based: Supply explicit data rate limit (Nodes along path may request rate reduction

Explicit Congestion Notification (ECN)

network-assisted congestion control: provided in IP and TCP and defined in RFC 3168

- two bits in IP header (ToS field) marked *by network router* to indicate end nodes packets that are experiencing congestion, without the necessity of immediately dropping such packets
  - 00 indicates a packet that is not using ECN
  - 01 or 10 is set by the data sender to indicate that the end-points of the transport protocol are ECN-capable
  - 11 is set by a router to indicate congestion has been encountered
- congestion indication carried to receiving host
- receiver (seeing congestion indication in IP datagram)) sets ECE bit on receiver-to-sender ACK segment to notify sender of congestion
- TCP header includes two one-bit flags:
  - ECN-Echo flag
  - CWR (congestion window reduced) flag



Datagram Congestion Control Protocol (DCCP)

- Defined in RFC 4340, Connection-oriented
- Runs on top of IP and serves as an alternative transport protocol for applications that would otherwise use UDP

DCCP Packet Types:

- DCCP-Request: Sent by the client to initiate a connection (the first part of the three-way initiation handshake)
- DCCP-Response: Sent by the server in response to a DCCP-Request (the second part of the three-way initiation handshake)
- DCCP-Ack: Used to transmit pure acknowledgments
- DCCP-Data: Used to transmit application data
- DCCP-DataAck: Used to transmit application data with piggybacked acknowledgment information
- DCCP-CloseReq: Sent by the server to request that the client close the connection
- DCCP-Close: Used by the client or the server to close the connection; elicits a DCCP-Reset in response
- DCCP-Reset: Used to terminate the connection, either normally or abnormally

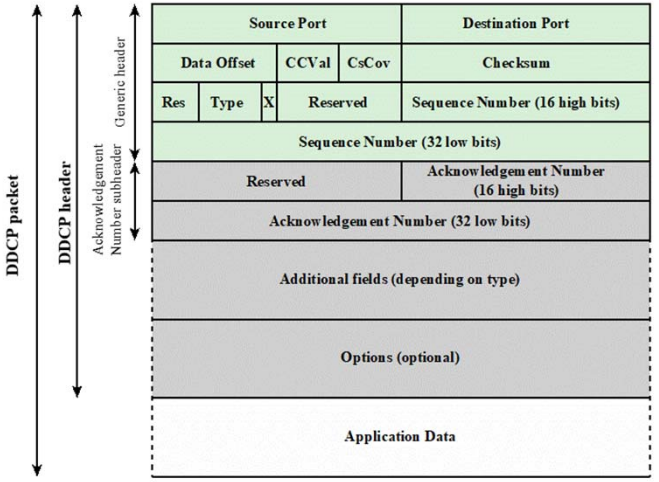


Figure 20.12 DCCP Packet Format (X = 1)

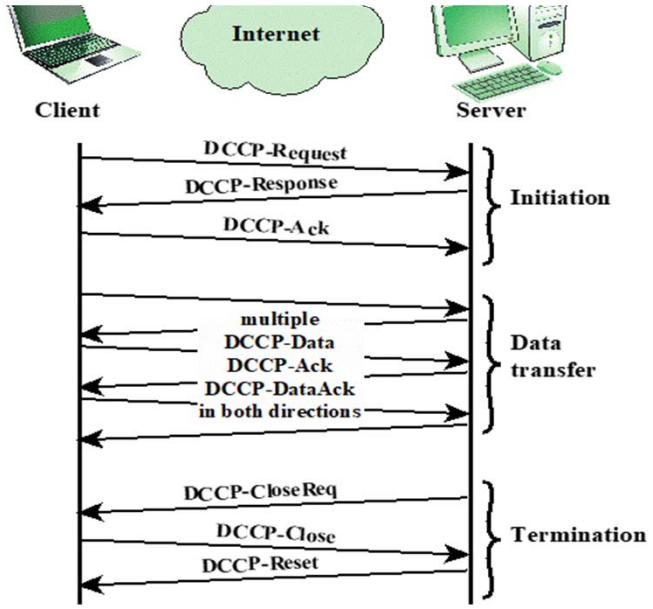


Figure 20.11 DCCP Packet Exchange

Traffic Management

Fairness

- Provide equal treatment of various flows

Quality of service

- Different treatment for different connections

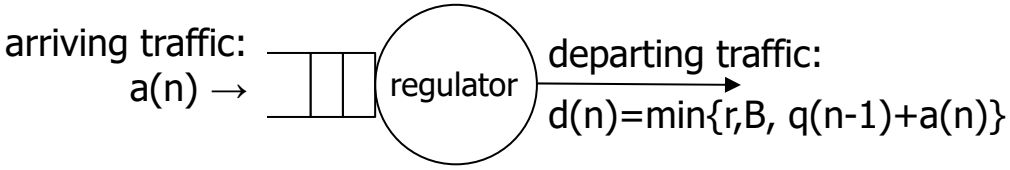
Reservations

- Traffic contract between user and network
- Excess traffic discarded or handled on a best-effort basis

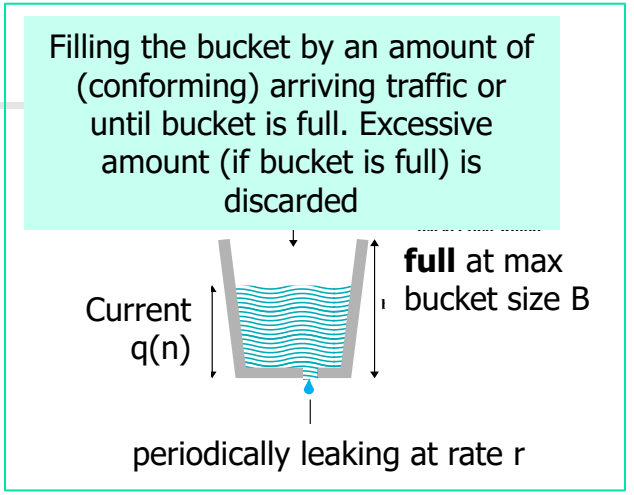
Traffic Shaping/Traffic Policing

- Two important tools in network management:
  - **Traffic shaping**
    - Concerned with traffic leaving the switch
    - Reduces packet clumping
    - Produces an output packet stream that is less burst and with a more regular flow of packets
  - **Traffic policing**
    - Concerned with traffic entering the switch
    - Packets that don't conform may be treated in one of the following ways:
      - Give the packet lower priority compared to packets in other output queues
      - Label the packet as nonconforming by setting the appropriate bits in a header
      - Discard the packet
- Leaky Bucket & Token Bucket:
  - Widely used traffic management tool
  - Advantages:
    - Many traffic sources can be defined easily and accurately
    - Provides a concise description of the load to be imposed by a flow, enabling the service to determine easily the resource requirement
    - Provides the input parameters to a policing function

Leaky bucket



departing traffic rate **cannot** exceed leaking rate r



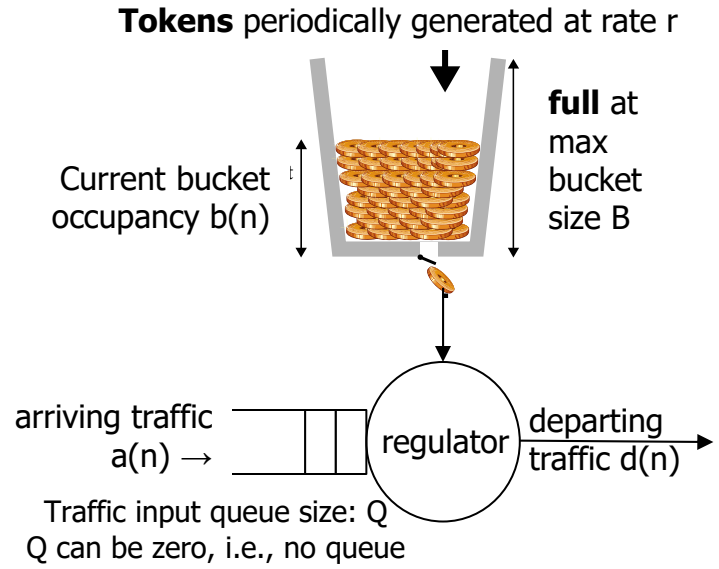
- max queue (bucket) size B
  - controls traffic burstiness.
- Leaking rate r regulates the traffic average rate
- **Previous** queue (bucket) occupancy: q(n-1)≤ B, (initially empty q(0)=0)
- **excessive** traffic=max{0, [q(n-1)+a(n)-B]} is **discarded (lost)**
- **Current** queue limit**:** q'(n)=min{B, [q(n-1)+a(n)]}
- **Current** departing traffic: d(n)=min{r,q'(n)}
- **Current** queue (bucket) occupancy: q(n)=max{0, [q'(n)-d(n)]}

Example: bucket initially empty with max size B=5 packets, leaking rate is r=3 packets/interval. (Notes: $(x)^+=\min(B, \max(x,0))$)

Time (n):	0	1	2	3	4	5
Arriving traffic: a(n)		2	4	5	7	0
queue limit**:** q'(n)	0	$2=(0+2)^+$	$4=(0+4)^+$	$5=(1+5)^+$	$5=(2+7)^+$	$2=(2+0)^+$
Departing traffic: d(n)	0	2	3	3	3	2
Current Bucket: q(n)	0	0	1	2	2	0

token bucket

- Token rate r control avg traffic rate
- bucket size B control traffic bustiness



departing traffic d(n) **cannot** exceed B+r

- previous **queue** q(n-1)≤Q, initially q(0)=0
- previous bucket occupancy b(n-1), b(0)=B
- Current **input** traffic: i(n)= a(n)+q(n-1),
- **departing** traffic: d(n)= min {i(n), b(n-1)+r}
- **excessive** traffic: e(n)= i(n)-d(n),
  (is **marked** and kept in the input queue if available space, otherwise **discarded**)
- current **queue** q(n)=min{Q, e(n)}
- current **bucket** b(n)=min{B, [b(n-1)+r-d(n)]}

Ex: bucket initially **full** at max size B=5 packets, token rate is r=3 packets/interval.

Time (n):	0	1	2	3	4	5
Arriving traffic: a(n)		2	4	5	7	0

Case: Q=5 Notes: $(x)^+=\min(B, \max(x,0))$

Current Bucket: b(n)	5	$5=(5+3-2)^+$	$4=(5+3-4)^+$	$2=(4+3-5)^+$	$0=(2+3-7)^+$	$1=(0+3-2-0)^+$
Current Queue: q(n)	0	0	0	0	**2**	0
Departing traffic: d(n)	0	2	4	5	**5**	**2**

Case: Q=0

Current Bucket: b(n)	5	$5=(5+3-2)^+$	$4=(5+3-4)^+$	$2=(4+3-5)^+$	$0=(2+3-7)^+$	$3=(0+3-0)^+$
Departing traffic: d(n)	0	2	4	5	**5**	**0**

summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- Internet: UDP, TCP
- Connectionless (UDP) and connection-oriented (TCP) transport protocol mechanisms
- Effects of congestion
  - Ideal performance
  - Practical performance
- Congestion control
  - TCP congestion control
- Traffic management
  - Fairness
  - Quality of service
  - Reservations
  - Traffic shaping and policing