King Fahd University of Petroleum & Minerals Computer Engineering Dept

COE 540 – Computer Networks

Term 112

Dr. Ashraf S. Hasan Mahmoud

Rm 22-420

Ext. 1724

Email: ashraf@kfupm.edu.sa

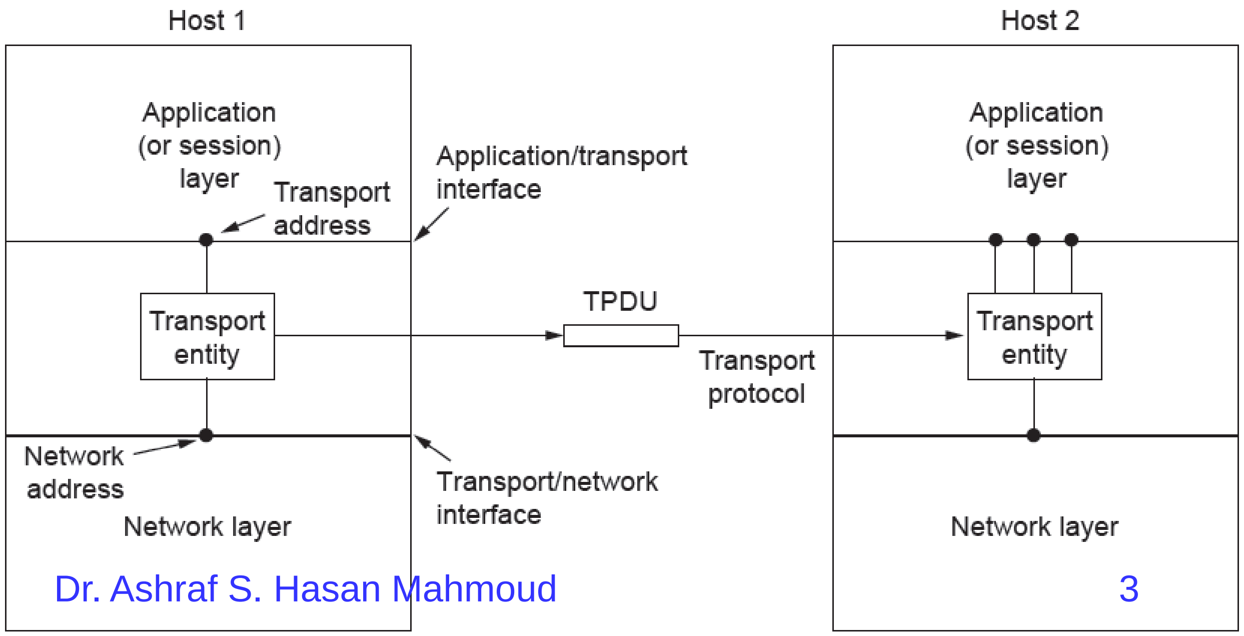
Lecture Contents

1. Transport Layer Services
2. Elements of Transport Protocols
3. Congestion Control
4. The Internet Transport Protocols: UDP and TCP

These slides are based on the Tanenbaum's textbook and original author slide

Transport Layer Services

- Transport layer should provide efficient, reliable, and cost-effective data transmission service to its users (processes in the application layer)
- Transport entity – Refer to figure
  - Location – OS kernel, library package for network application, or on the NIC itself

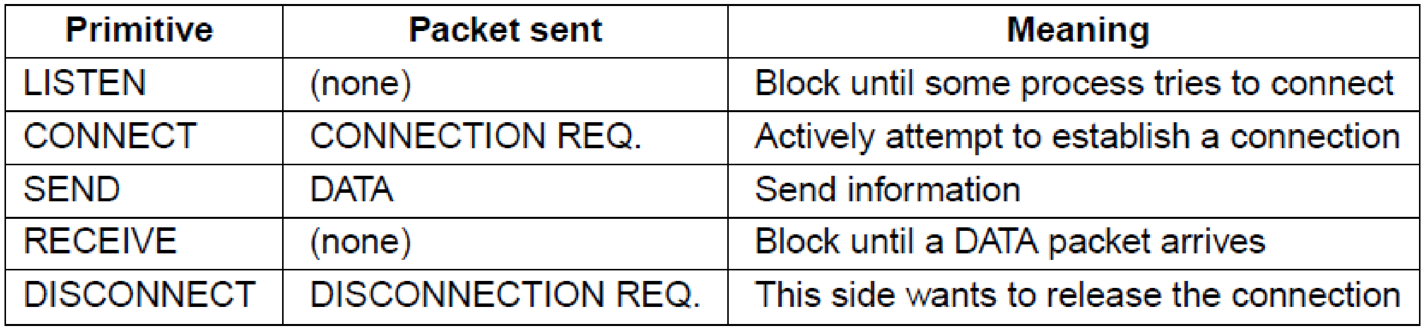


Transport Layer Services – cont'd

- Types: connection-oriented and connectionless
  - Similar to the services provided by the network layer
- Items to think about:
  - Would the transport layer provide a connectionless service over a connection-oriented network layer? Why?
  - If the transport layer is providing services similar to those provided by the network layer – then why do we have two layers?
- Transport layer may provide extra reliability (end-to-end) compared to the underlying network
- Transport layer defines standard set of primitive
  - Application layer protocols may be developed in isolation of the underlying network layer

Transport Layer Service Primitives

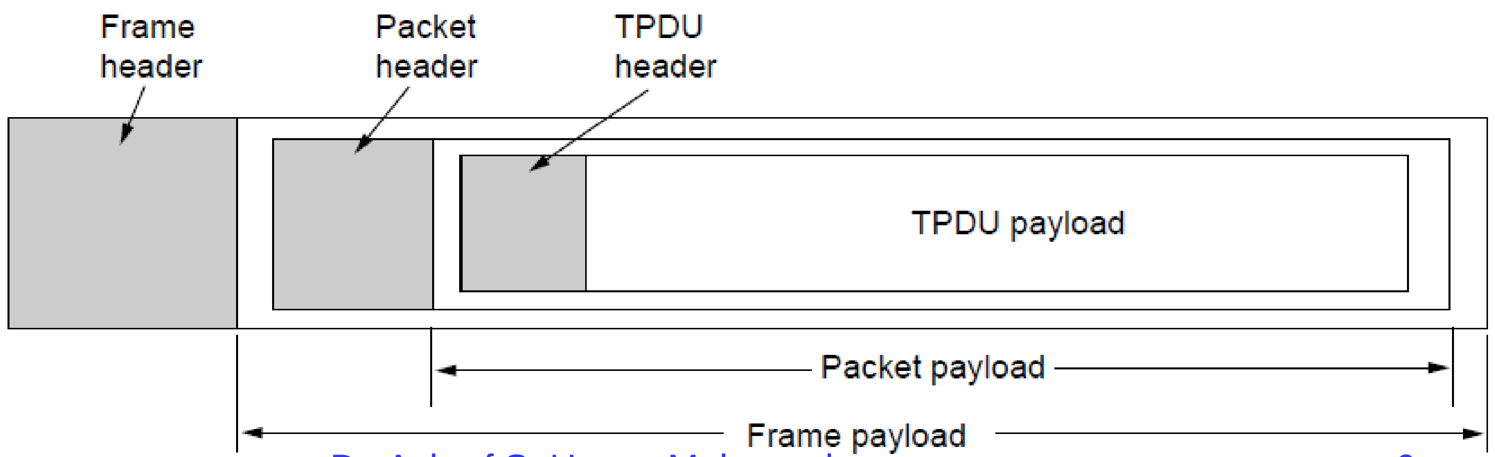
- Connection-oriented service – provides reliable transport over unreliable network
  - Hides all imperfections in terms of lost packets, acks, timers, congestion, etc.
- Connectionless service similar to datagram service provided by network layer
  - Some applications such as streaming-streaming
- Transport service must be clearly defined and easy to use
  - Example – 5 primitives defined below
  - Allow an application to establish (listen and connect), use (send & receiver), and release (disconnect) connections



Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

Transport Layer Service Primitives – cont'd

- Nesting of segments, packets, and frames
- Transport Protocol Data Unit (TPDU) – used by older protocols
- Segment – used by TCP and UDP



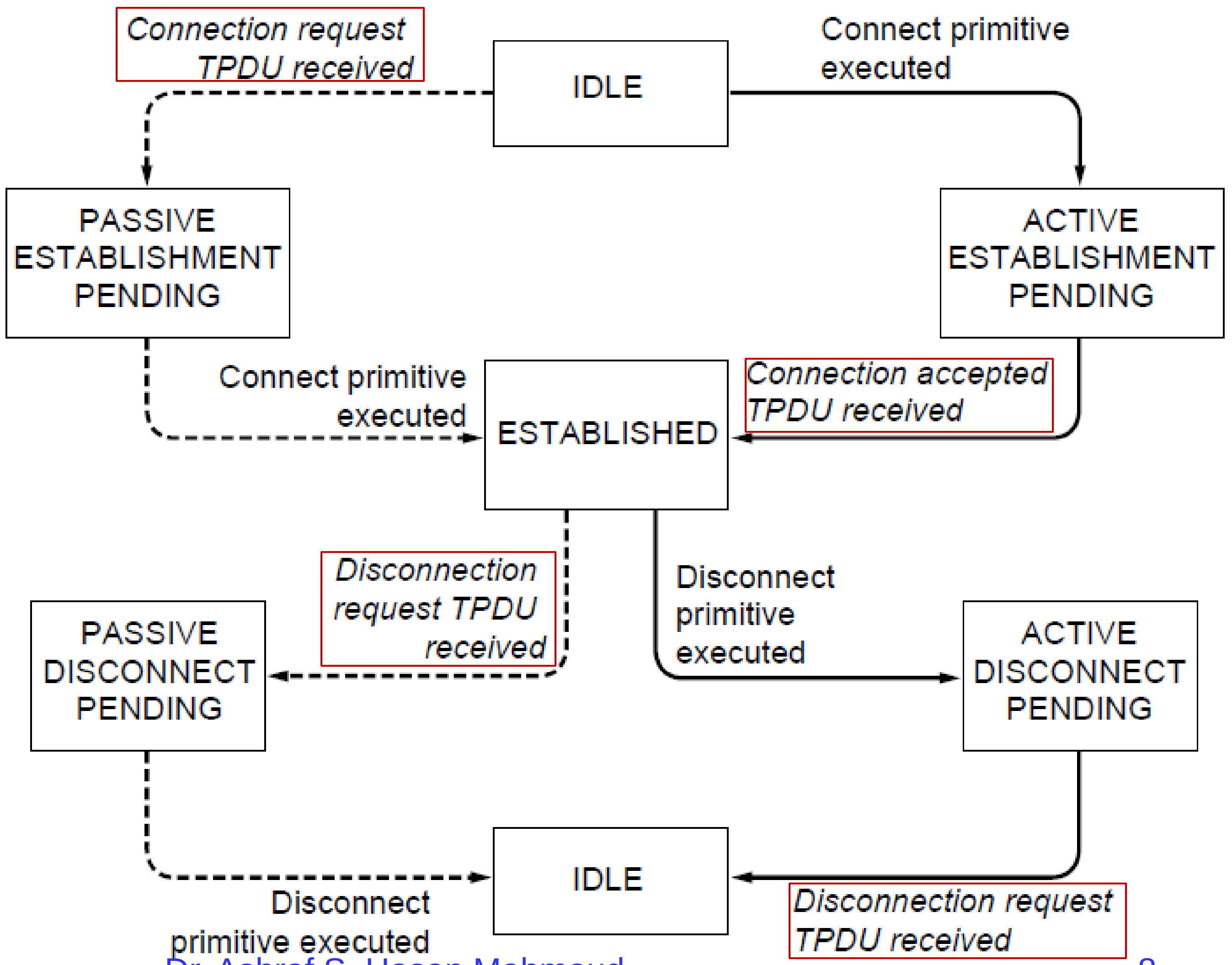
Transport Layer Service Primitives – Client-Server Example

- (1) Client does CONNECT call → CONNECTION REQUEST segment is sent to server
- (2) Server transport entity blocked on a LISTEN call – receives segment and processes it
- Server sends CONNECTION ACCEPTED segment back to client
- (3) Data can now be exchanged between client and server
  - SEND and RECEIVE calls
- Hidden tools (from transport users):
  - ACKs for control segments and data segments – managed by transport entities
  - Timers and retransmission
- (4) When connection is no longer needed – DISCONNECT primitive
  - Asymmetric – one side sends the DISCONNECT segment; when received by the other side, the entire connection is released
  - Symmetric – each direction is closed separately

Transport Layer Service – Connection States

- State diagram for a simple connection management scheme
- Transitions labeled in italics are caused by packet arrivals
- The solid lines show the client's state sequence
- The dashed lines show the server's state sequence

▭ Packets received


IDLE
Connection request TPDU received
Connect primitive executed
PASSIVE ESTABLISHMENT PENDING
ACTIVE ESTABLISHMENT PENDING
Connect primitive executed
ESTABLISHED
Connection accepted TPDU received
Disconnection request TPDU received
Disconnect primitive executed
PASSIVE DISCONNECT PENDING
ACTIVE DISCONNECT PENDING
IDLE
Disconnect primitive executed
Disconnection request TPDU received

Elements of Transport Protocols

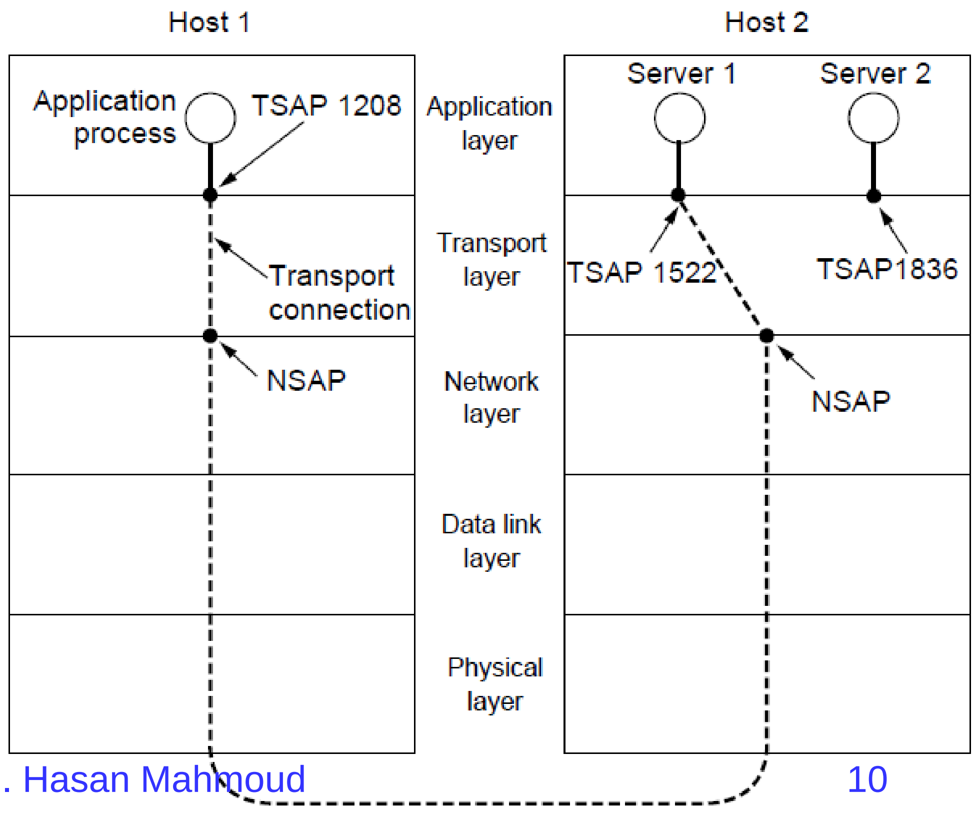
- Transport service implemented by “transport protocol” between two transport entities
  - Similar to data link protocols of chapter 3
- Functions: error control, sequencing, flow control, etc.
- What are the core differences between data link protocols and transport protocols?
  - Point-to-point link versus entire network
  - Simple (no) addressing versus explicit addressing of destinations
  - Trivial connection establishment versus involved
  - Storage capacity of the network – delayed/out-of-order/lost segments
  - Buffering and flow control – different/more robust mechanisms required at transport level to handle bandwidth fluctuations, etc.
- Elements: Addressing, Connection Establishment, Connection Release, Error Control and Flow Control, Multiplexing, Crash Recovery

Elements of Transport Protocols - Addressing

- Transport Service Access Point (TSAP)
  - Required for both connection-oriented or connectionless services
- Application processes can attach themselves to a local TSAP to establish a connection to a remote TSAP

- **Example – Mail Server**

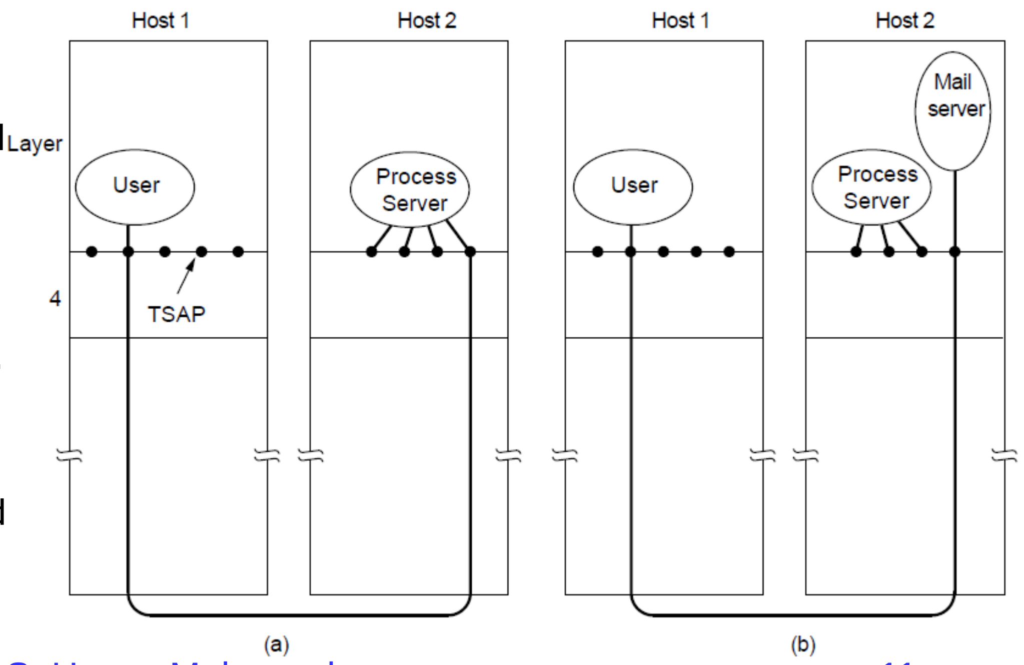
1. Mail server on Host 2 listens to TSAP 1522
2. Application process on Host 1 uses TSAP 1208 to establish connection to mail server (dst TSAP is 1522)
3. Application process on Host 1 sends mail message to mail server on Host 2
4. Mail server on Host 2 accepts mail message
5. Transport connection released



Elements of Transport Protocols – Addressing – cont'd

- How does a user process (e.g. the mail client) know the TSAP for the mail server?

- Stored at */etc/services* of UNIXX and/or use well know ports (TCP port 25 for mail)
- What about the case of thousands of newly developed services?
  - **portmapper** - users sets up connection to portmapper service (listens to well-known TSAP) and obtains the target TSAP for the desired service
  - New service must register with portmapper
  - Similar to directory assistance
- Initial connection protocol – **process server**
  - On UNIX – called *inetd*.
  - Gets initial request, spawns the requested server (e.g. mail server) and allows it to inherit the connection
  - Valid when servers can be created on demand!

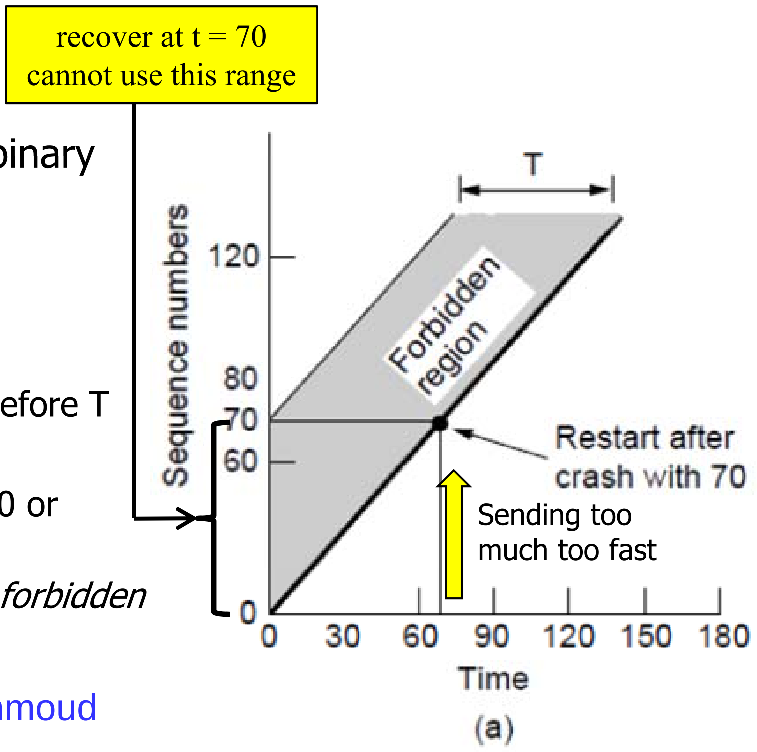


Elements of Transport Protocols – Connection Establishment

- Core problem in transport – packets get duplicated and delayed
  - Should not be thought of as new packets
- Solutions:
  - Throw away addresses – makes it very hard to connect to services in the first place!
  - Connection identifiers table used by the peers to distinguish valid from obsolete ones – what to do in case of machine crash? Where to start from?
  - Packet lifetime: e.g. restricted network design, hop counter, time stamp, etc.
- Need to guarantee that delay duplicate data packet and their corresponding ACKs are dead after some time
  - Maximum packet life time – 120 sec for the Internet
  - Select T = small **multiple** of 120 sec – multiple is protocol dependent; makes T larger!
- If we wait T sec after sending a packet → we are certain that all traces of it are gone

Elements of Transport Protocols – Connection Establishment – cont'd

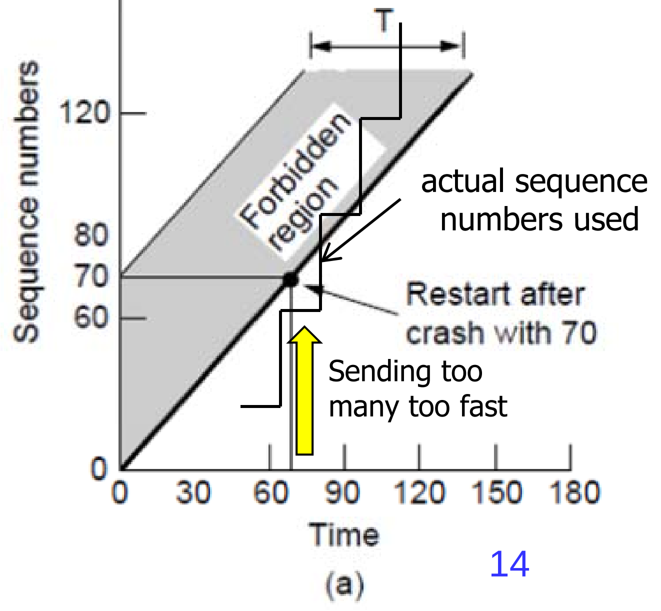
- Solution used for TCP
- T is defined for the network
- (1) Use sequence #s for sent segments
  - Reused **only** after T seconds
  - Only one packet with a given sequence number may be outstanding at any given time
- To handle machine losing memory after a crash – force machine to wait T second
  - Sender can then start fresh from any sequence number
  - What if T is large?
- (2) Equip hosts with time-of-the-day clock (form of binary counter incremented in uniform intervals)
  - Number of bits in clock >= or bits in sequence number
  - Clock continues to run even if host is down
  - Use first k-bits of clock as initial sequence number
  - Sequence space should be *large* to prevent wrapping before T
- Example (a) if host crashes and restarts at t = 70 seconds
  - It is will use initial sequence number based on clock (70 or higher)
  - Host cannot start with lower sequence numbers in the *forbidden region*



Elements of Transport Protocols – Connection Establishment – cont'd

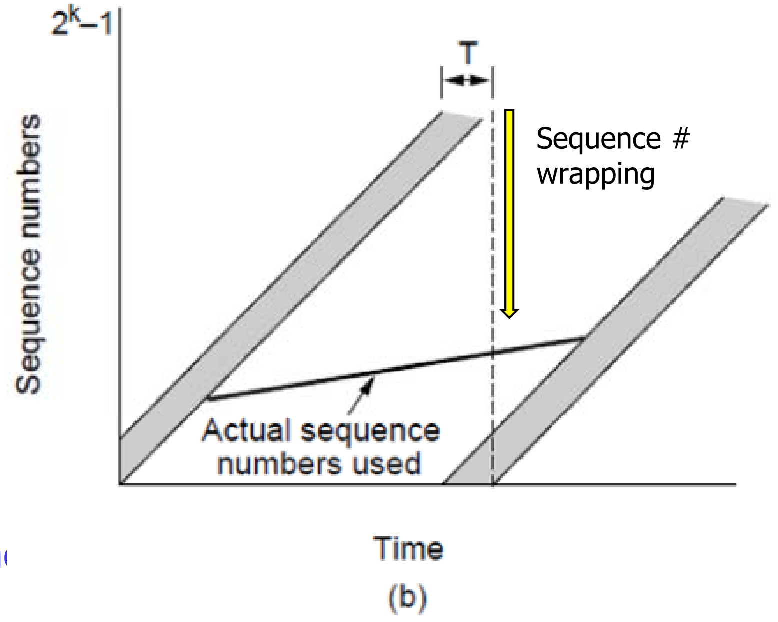
- How can a host enter the forbidden region of sequence numbers?
- (1) Host sends too much data too fast on a newly opened connection
  - Sequence # versus time curve rises too quickly – enters the forbidden region from below
  - Solution – maximum rate = one segment per clock tick!
    - Host also waits for at least one clock tick if recovering from a crash
- Use fast clock – there is a limit to how fast the clock can tick!

- Let C – clock rate, S – sequence number space size, T – lifetime
- Then from geometry S/C must be greater than T or C <= T x S
  - Otherwise, sequence space will run out very quickly!


T
Sequence numbers
120
80
70
60
0
Forbidden region
actual sequence numbers used
Restart after crash with 70
Sending too many too fast
0
30
60
90
120
150
180
Time
(a)

Elements of Transport Protocols – Connection Establishment – cont'd

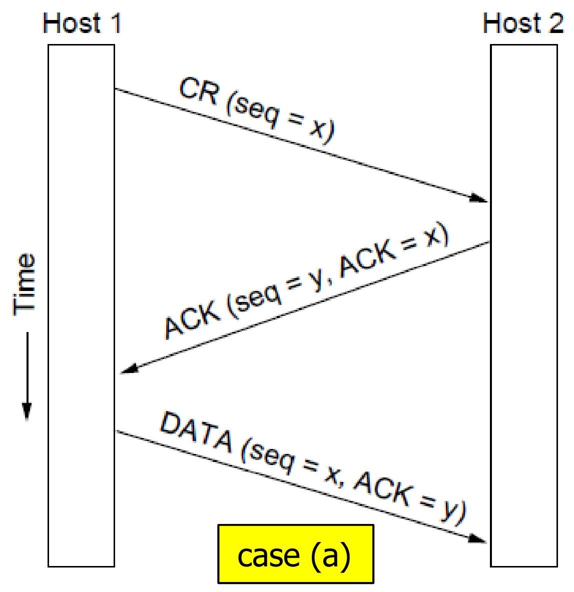
- How can a host enter the forbidden region of sequence numbers?
- (2) Host sends data slowly on a newly opened connection (slower than one segment per clock tick!)
  - Actual sequence numbers used curve (versus time) will enter the forbidden region from the left.
  - This limits the minimum send rate on the connection!!
    - Or how long the connection may last



Elements of Transport Protocols – Connection Establishment – Three-Way Handshake

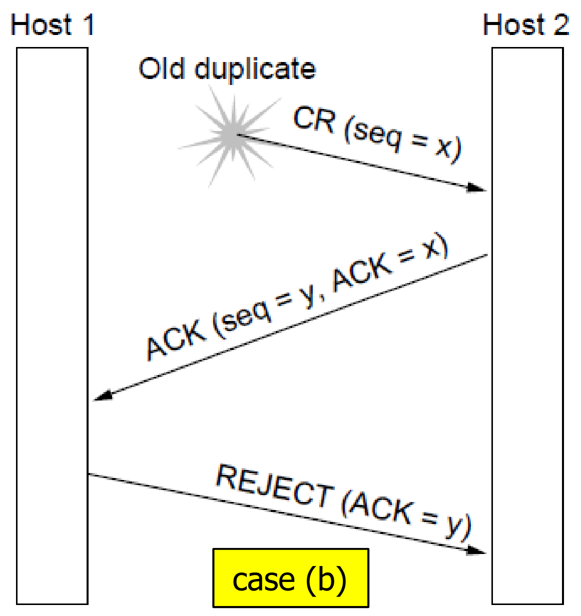
- Clock-based method solves the problem of not being able to distinguish delayed duplicate segments from new ones
- **Problem:** Sequence numbers are not remembered at the destination – How would a receiver know that this CONNECTION REQUEST segment of an initial sequence number is not a duplicate?
- **Solution:** *Three-way handshake* protocol

- Why doesn't this problem occur during the connection?
  - Note that during the connection the sliding window protocol will remember the current (used) sequence numbers
- The above issue is for the initial CONNECTION REQUEST segment
  - **Assumption** – Destination does not remember sequence numbers between connections



Elements of Transport Protocols – Connection Establishment – Three-Way Handshake – cont'd

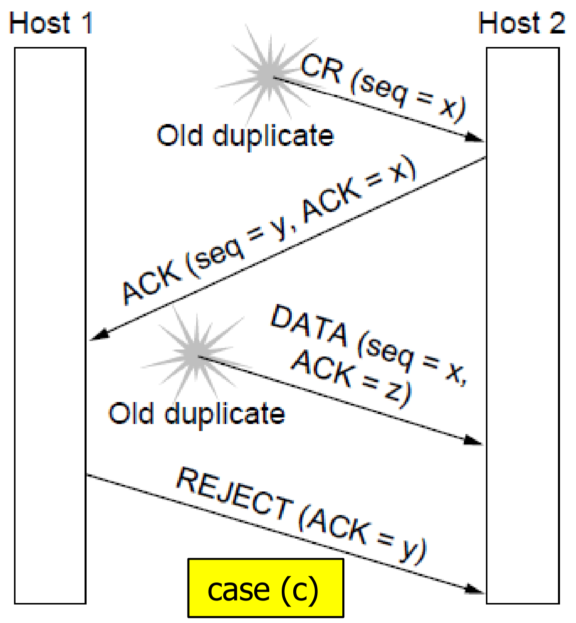
- Three-way handshake in the presence of delayed duplicate control segments
- Case (b)
- Case (c) – worst case scenario
- No combination of old segments that can cause the protocol to fail and have a connection set up by accident when no host wants it.



Delayed duplicate of CR appears at Host 2

Host 2 checks with Host 1 to see if indeed a new connection is required

Host 1 rejects and Host 2 does not open new connection



Delayed duplicate of CR appears at Host 2

Host 2 checks with Host 1 to see if indeed a new connection is required – suggests y as a sequence #

Host 2 receives ACK z ➔ Host 2 knows this is delayed duplicate

Host 1 rejects new connection as before

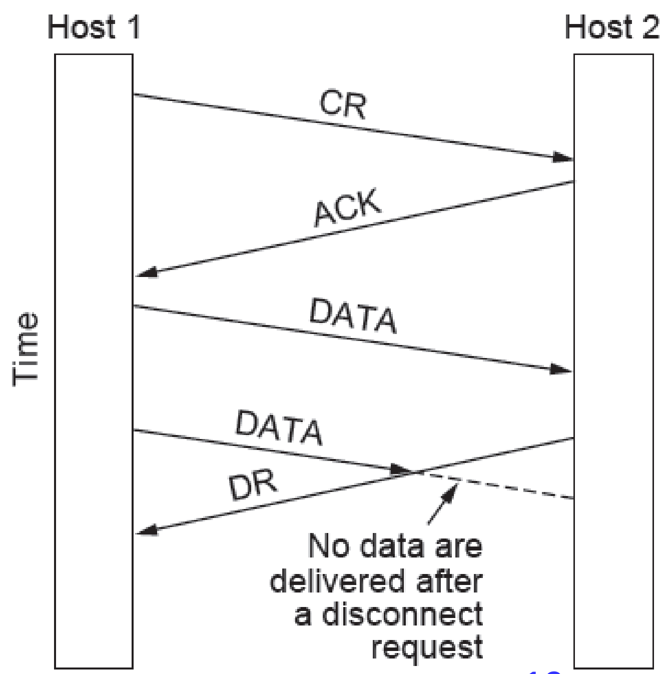
Elements of Transport Protocols – Connection Establishment – TCP Three-Way Handshake

- TCP uses **three-way handshake** protocol to establish connections
- Time-stamp to extend the 32-bit sequence number
  - Will not wrap within the maximum packet lifetime even for gigabit/sec connections
- RFC 1323 describes **Protection Against Wrapped Sequence Numbers (PAWS)**
- TCP used the clock-based sequence number initially
  - Attacker can easily predict the next initial sequence number and send packets to trick the three-way handshake procedure
- Currently, *pseudorandom* initial sequence numbers are used for connections in practice.
  - Forbidden region problem still applicable.

Elements of Transport Protocols – Connection Release

- Connection release – symmetric versus asymmetric
- Asymmetric release is abrupt and may lead to data loss (as shown in Figure)
  - Host 2 issues DISCONNECT before the second data segment arrives

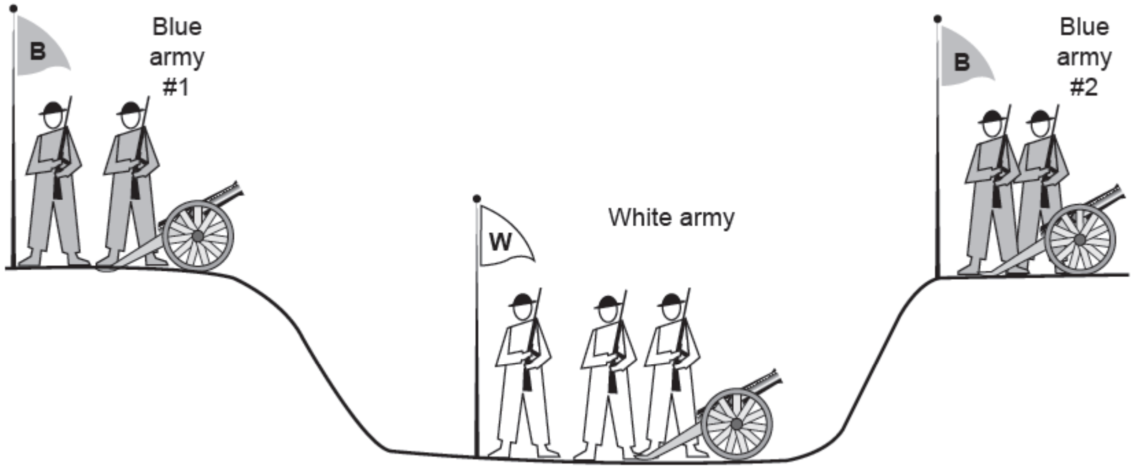
- Solution – use symmetric release
  - Each direction is released independently – A host can continue to receive data after it has sent a DISCONNECT segment.
  - Works well ONLY if each side has FIXED amount of data and CLEARLY KNOWS when it has sent it!
- What to do?



Elements of Transport Protocols – Connection Release – The Two-Army Problem

- Consider the two-army-problem
  - White army can defeat each of the blue armies individually. The two blue armies can defeat the white army if they join forces
  - The blue armies want to synchronize attacks and their only communication medium is to send messengers on foot into the valley where they might be captured or lost
  - Is there a protocol that will allow the blue armies to win?

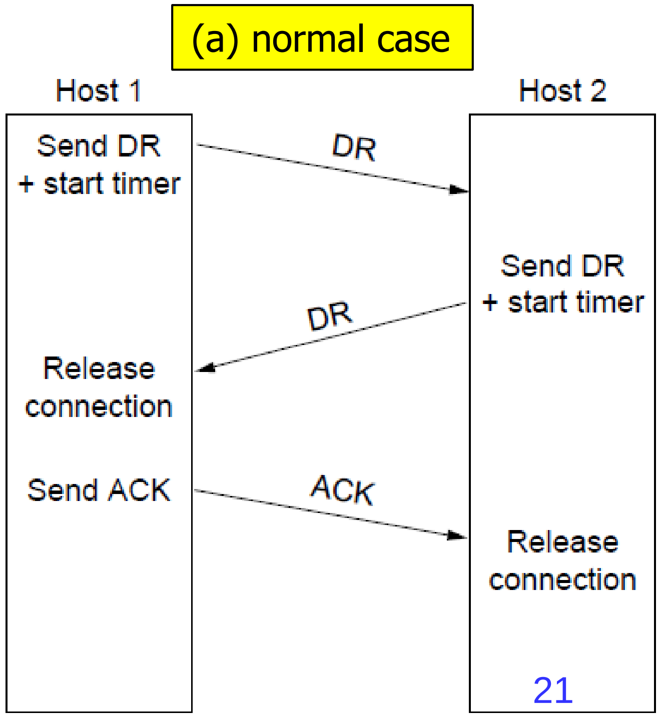
- Run 1: B1 (Time) → B2; B2 agrees and sends ACK; B2 (ACK) → B1
  - Will B2 charge? Probably not – B2 has no way of knowing that ACK has been received at B1!
- Run 2 (using three-way handshake) – The initiator of the original proposal must acknowledge the response
  - B2 will get his ACK – but B1 is now sure that his ACK has been received?
- Run 3 (using four-way handshake)
  - ….



It can be proven that now protocol exists that would solve the two-army problem - Proof in textbook – important!!

Elements of Transport Protocols – Connection Release – cont'd

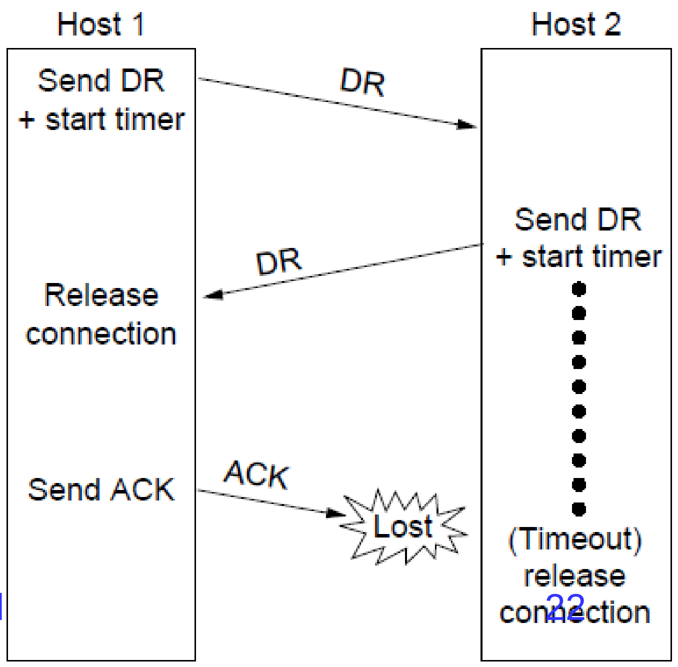
- Normal case – three-way handshake
- Timers associated with DISCONNECT REQUEST
- Connection released when Host 2 receives ACK



Elements of Transport Protocols – Connection Release – cont’d

- Final ACK sent from Host 1 to Host 2 is lost
- Timer at Host 2 expires ➔ connection released anyway

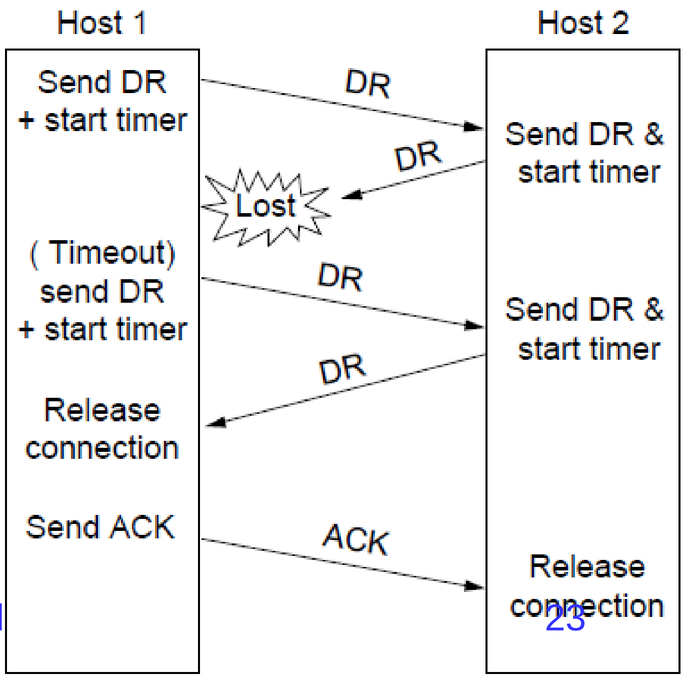
(b) final ACK lost case



Elements of Transport Protocols – Connection Release – cont'd

- Second DISCONNECT REQUEST lost
- Host 1 will NOT receive the expected response!
- Timer at Host 1 expires ➔ Restarts all over
  - Re-send DISCONNECT REQUEST

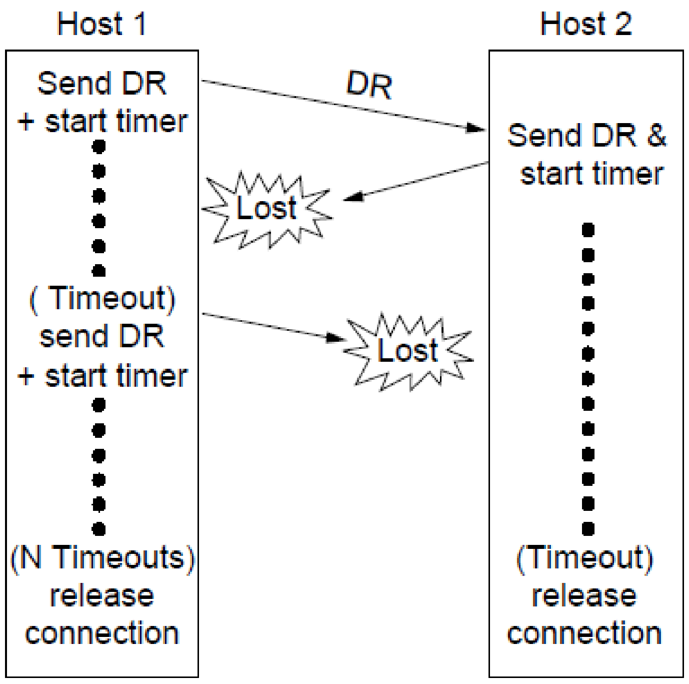
(c) 2nd DR lost case



Elements of Transport Protocols – Connection Release – cont'd

- Response DISCONNECT REQUEST from Host 2 and subsequent DISCONNECT REQUESTs from Host 1 are lost
- Host 2 times out and releases connection
- Host 1 will try N times and after N timeouts it will release the connection

(d) Response lost and subsequent DRs lost case


Host 1
Host 2
Send DR + start timer
DR
Send DR & start timer
Lost
(Timeout) send DR + start timer
Lost
(N Timeouts) release connection
(Timeout) release connection

- Host 2 times out well before Host 1 (diagram not to scale!!)
- **What if** one DISCONNECT REQUEST from Host 1 makes it to Host 2 after Host 2 has timed out?

Elements of Transport Protocols – Connection Release – cont'd

- Is it possible for the previous protocol to fail?
- Yes – If initial DISCONNECT REQUEST and N retransmissions from Host 1 all fail
  - Host 1 will release connection while Host 2 still thinks connection is active!
  - Half-open connection
- Solution (to half-open connection)
  - Force Host 1 to get a response (i.e. N → infinity)
  - But if Host 2 is allowed to time-out, then indeed Host 1 will keep sending DRs for ever!
  - If Host 2 is NOT allowed to time-out, the protocol hangs (in case (d))
- Modified solution (for half-open connection) – if no segment arrives within X seconds, close connection
  - If one side disconnects → no segments sent → receiver closes connection
  - Also if connection is broken (without either end disconnecting) → situation resolved by both sides closing the connection
  - Each host must make sure that inter-segment time is less than the chosen threshold
    - Send dummy segments when no data segment are available to keep the connection alive
- What if automatic disconnect is employed and too many dummy segments get lost?
  - First one side will disconnect – then the other

Elements of Transport Protocols – Connection Release – cont'd

- Conclusion – Releasing connection without data loss is NOT a simple problem!!
- The transport user must determine when to disconnect
  - Any solution based on the transport entities themselves is not perfect
- Example 1: TCP (normal case)
  - Symmetric close – each side sends a FIN segment to close its half of the connection
- Example 2: Web server
  - Web server sends RTS (warning) packet to client; causes abrupt close of connection
    - Server knows the traffic pattern and knows that all what need to be sent on this connection has already been sent
  - Server closes connections
  - If client receives the warning it closes the connection; if it does not, it times out and closes the connection eventually!

Elements of Transport Protocols – Error Control and Flow Control

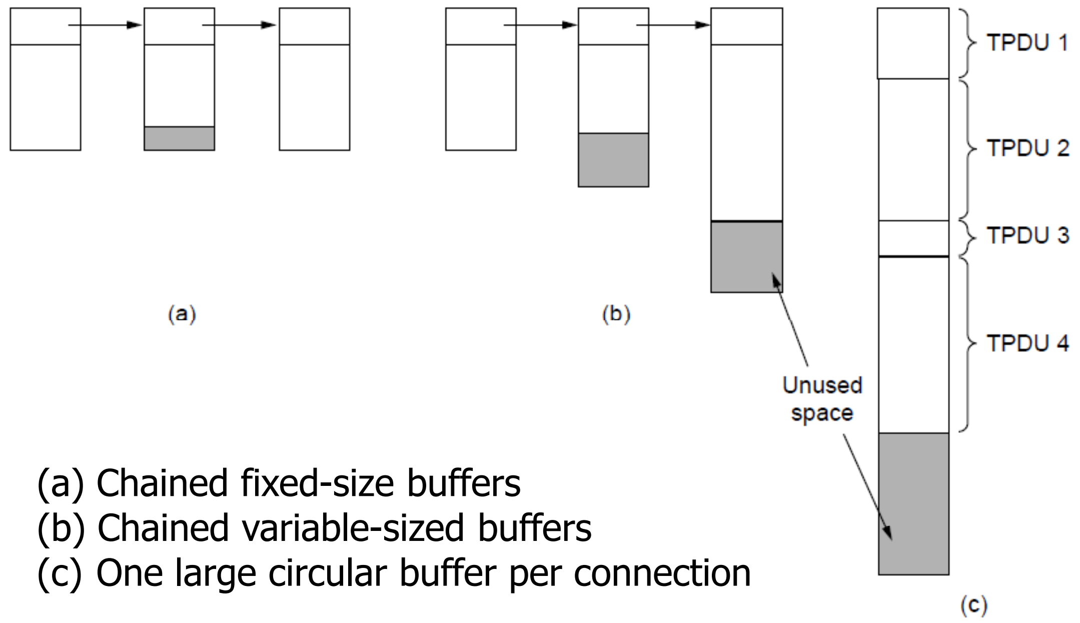
- Managing connections while they are in use
- Error control (EC) – ensuring that all data is delivered reliably
- Flow control (FC) – preventing sender from overwhelming receiver
- EC and FC are also functions at the data link layer
  - EC – CRC plus ARQ
  - FC – stop-and-wait and sliding window protocols
- Why have the same function again on transport layer?
  - Differences in *function* and *degree* – refer to textbook discussion
  - The end-to-end argument and bandwidth-delay product perspective

Elements of Transport Protocols – Error Control and Flow Control – Buffering of Data

- Transport layer generally may deploy a sliding window protocol – needs to buffer data
  - For each on going connection
- Sender must buffer sent but not yet acked segments
- Receiver may or may not buffer
  - Receiver may drop a segment if buffer is not available since sender is prepared to retransmit!
- What is the best trade-off between buffering at sender and buffering at receiver? Depends on type of traffic
  - Low bandwidth bursty traffic (e.g. terminal) – dynamic buffer (not pre-reserved) – rely on buffering at sender
  - High bandwidth traffic (e.g. file transfer) – dedicated buffers at receiver – allow data to flow at maximum speed

Elements of Transport Protocols – Error Control and Flow Control – Buffering of Data – cont'd

- How to organize the buffer pool?
- Three possible strategies are shown below
- (a) – Simple – but will have problems if segment sizes differ widely; may lead to low memory utilization
- (b) – better memory utilization – complex buffer management
- (c) – good memory utilization especially when connections are heavily loaded



(a) Chained fixed-size buffers
(b) Chained variable-sized buffers
(c) One large circular buffer per connection

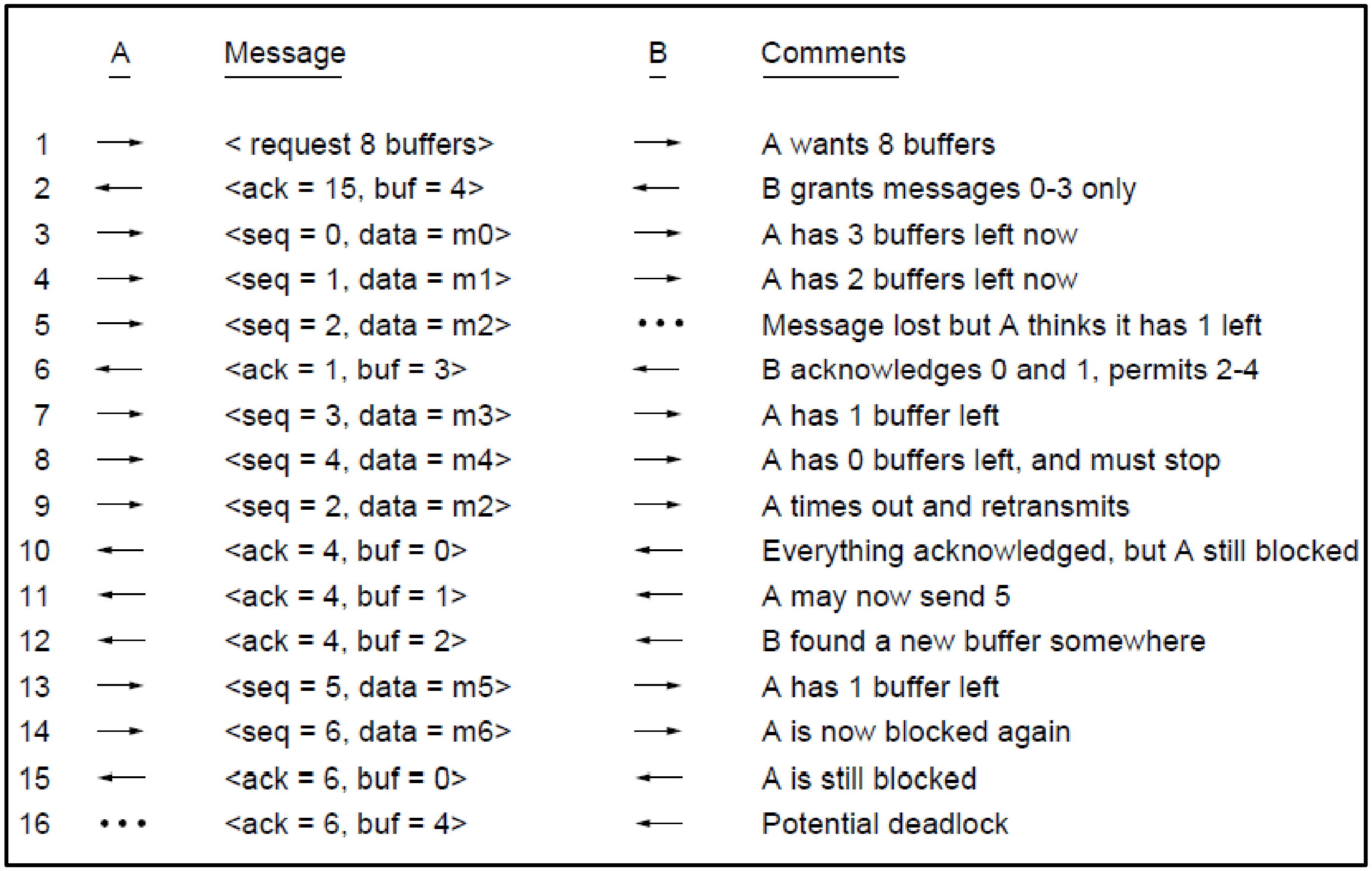
Elements of Transport Protocols – Error Control and Flow Control – Buffering of Data – cont’d

- Transport protocol should allow a sending host to request buffer space from receiver (i.e. dynamic buffer allocation)
  - Buffers may be allocated: collectively or per connection
- General steps:
  - Sender request certain number of buffer (based on expectation)
  - Receiver grants as many as it can afford
  - Every time sender sends a segment, it decrements from allocated buffer; stops when it reaches zero
  - Receiver piggybacks acks and buffer allocations onto reverse traffic
- The above policy is used by TCP
  - Allocated buffer space – WindowSize field in TCP segment header

Elements of Transport Protocols – Error Control and Flow Control – Buffering of Data – cont'd

- Flow of data segments from Host A to Host B
- Flow of ACKs and buffer allocations from B to A
- Line 10 – B acks all segments but freezes A
- Line 16 – if buffer allocation segment is lost → **deadlock**
  - To prevent – control segments may be sent periodically!

Example – Datagram network with 4-bit sequence numbers



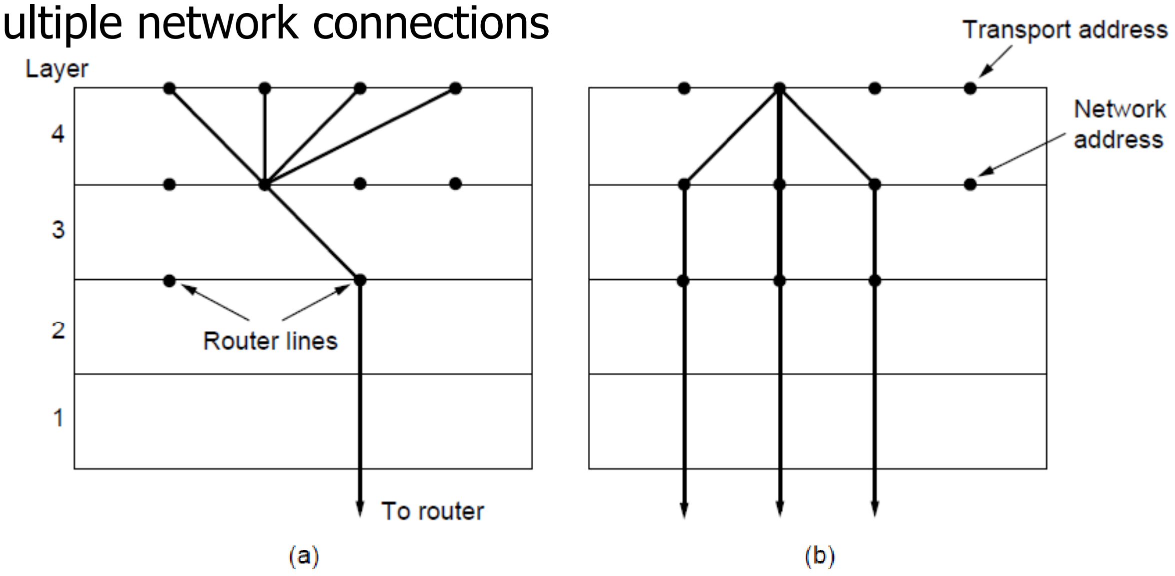
	A	Message	B	Comments
1	→	< request 8 buffers>	→	A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	A may now send 5
12	←	<ack = 4, buf = 2>	←	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	A is now blocked again
15	←	<ack = 6, buf = 0>	←	A is still blocked
16	• • •	<ack = 6, buf = 4>	←	Potential deadlock

Elements of Transport Protocols – Error Control and Flow Control – Buffering of Data – cont'd

- The previous material assumes memory space is premium and thus buffer space limits the maximum flow
  - Nowadays, memory space is not that expensive – can afford to allocated large (if not huge) buffer spaces
- Another bottleneck – The network bandwidth
  - Sender should not send more than the carrying capacity of the network → congestion
- We need a mechanism to limit the sender rate based on the network bandwidth (not the receiver buffer only)
  - Dynamic sliding window at sender can implement both: flow control and congestion control
- Network capacity – c segments/sec; round trip time = r seconds ➔ Sender window should c x r - segments

Elements of Transport Protocols – Multiplexing

- Multiplexing – several transport connections are carried over one network connection
  - TCP uses single network connection
- Inverse multiplexing – several network connections exist
  - May be used for load balancing or bandwidth expansion
  - Stream Control Transmission Protocol (SCTP) is a transport protocol that is able to operate over multiple network connections



(a) Multiplexing. (b) Inverse multiplexing.

Elements of Transport Protocols – Crash Recovery

- Recovery from network and router crashes is trivial IF transport entity resides entirely at the end host system
  - Transport protocol expects lost segments
- What is the case when hosts crash?
  - Would a client be able to continue to work if the server crashes and quickly reboots?
- Consider the client-server scenario shown in Figure.
  - Assuming no *simultaneous* (i.e. A and W) events

- Assuming stop-and-wait
- Six different combinations
  - (A) – Acknowledge
  - (W) – write segment to application
  - (C) – crash
- Client states: S0 – no segment outstanding, S1 – segment outstanding
- Four different strategies for client
- For any strategy – there exist a sequence of events at server that makes the protocol fail!

Strategy used by receiving host

Strategy used by sending host	First ACK, then write			First write, then ACK		
	AC(W)	AWC	C(AW)	C(WA)	W AC	WC(A)
Always retransmit	OK	DUP	OK	OK	DUP	DUP
Never retransmit	LOST	OK	LOST	LOST	OK	OK
Retransmit in S0	OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1	LOST	OK	OK	OK	OK	DUP

OK	= Protocol functions correctly
DUP	= Protocol generates a duplicate message
LOST	= Protocol loses a message

Different combinations of client and server strategy

Elements of Transport Protocols – Crash Recovery – cont’d

- Recovery from a layer N crash can only be done by layer N+1
  - Only if higher layer retains enough information to reconstruct
- In light of the previous example – what does receiving an end-to-end ack mean?

Congestion Control

- Focus – congestion control mechanism at the transport layer
- Goals of the congestion control
- How can hosts regulate the rate at which they send packets into the network
- The Internet relies ***heavily*** on the transport layer congestion control and the specific algorithms built into the TCP

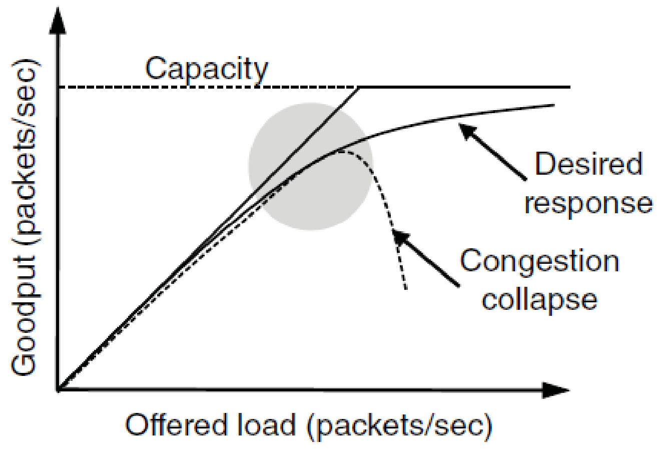
Desirable Bandwidth Allocation

- Find a good allocation of network bandwidth
  - Deliver good performance; uses all available bandwidth
  - Avoid congestion
  - Fair amongst competing transport entities
- Refer to Goodput/load versus load curves
  - Onset of congestion occurs well before maximum capacity is reached
- Practical range – just before the delay starts to climb rapidly

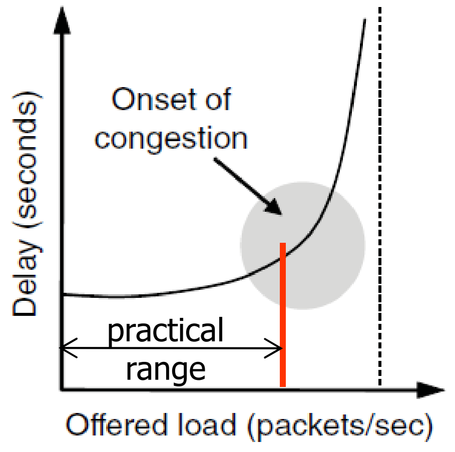
- A new performance metric – power

power = load / delay

- Power will initially rise with offered load, will reach a maximum and falls as delay grows rapidly
- Efficient load ←→ max power



(a)



(b)

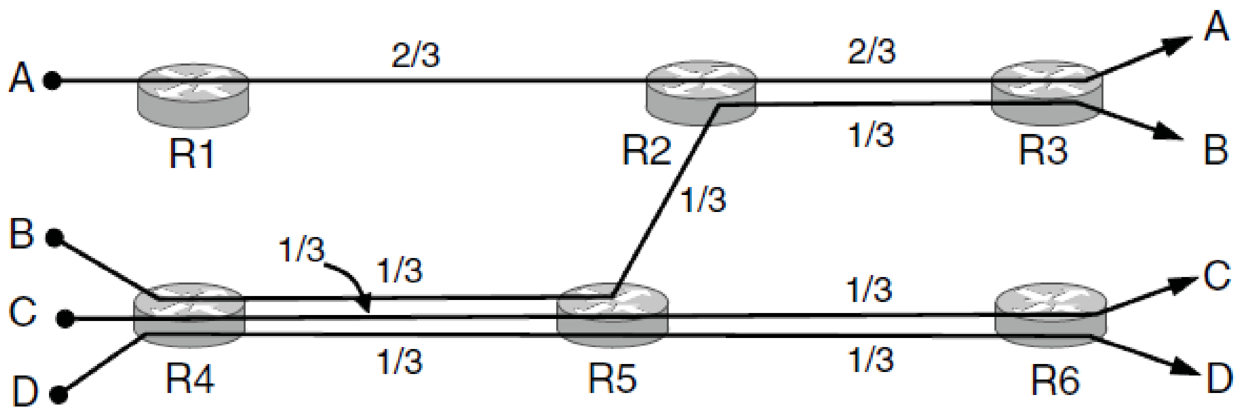
Max-Min Fairness

- Three considerations:
  1. How is allocated portions of BW related to congestion control – reservations versus getting available BW
  2. What fair portion to allocate for flows?
  3. Level over which to consider fairness?
     - Per host, per connection
     - WFQ
- The next two slides elaborate on fairness issue

Max-Min Fairness – cont'd

- How to divide bandwidth (BW) amongst different transport senders?
- Max-Min Fairness – an allocation is max-min fair if the BW given to one flow can not be increased without decreasing the BW of another flow with an allocation that is no larger
  - i.e. increasing the BW of one flow will make the situation worse for flows that are already at a disadvantage
- Example – Max-Min BW allocation for four flows
  - Flows – A, B, C, and D
  - All links have same capacity – 1 unit
  - Three flows (B, C, and D) compete for link R4-R5 – 1/3 is allocated for each flow
  - On link R2-R3, flow A gets remaining 2/3 capacity

- Links R1-R2 and R5-R6 have spare capacity
  - Cannot be used to increase the allocation of any flow without decreasing the allocation of another lower flow
- The allocation is max-min



Max-Min Fairness – cont'd

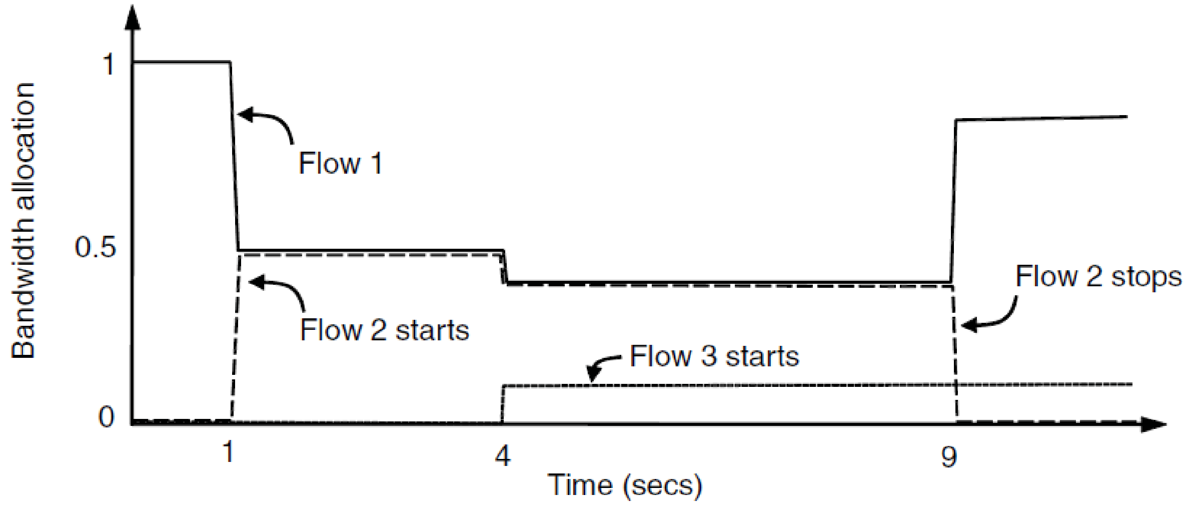
- How to compute max-min allocations?

- Any suggestions?
    - Refer to textbook pages 551 and 552
- At what level to consider fairness – no clear answers
    - Connection
    - Connections between a pair of hosts
    - All connections per host

Congestion Control - Convergence

- Criterion – congestion control algorithm should converge quickly to a fair and efficient allocation of BW
- Example – change of BW allocations with time

- Bottom line:
  - Total allocated BW is approximately 100%
  - Competing flows get equal treatment – but not more than what they need!

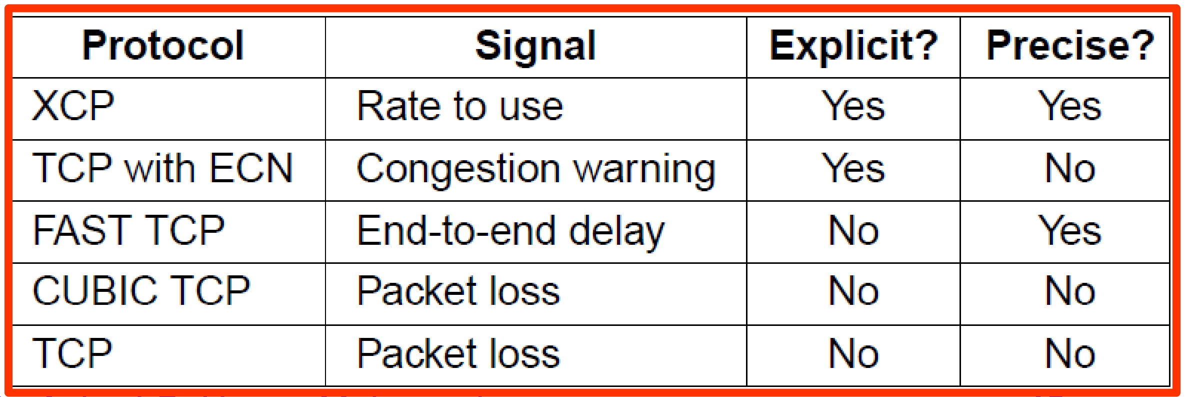


Changing bandwidth allocation over time

Regulating the Sending Rate

- Sending rate limited by
  - Flow control – variable-sized window at receiver
  - Congestion control
- How to infer the problem occurring on the connection
  - Network layer provides feedback – why type?
  - Explicit versus implicit?
  - Precise versus imprecise?
- Refer to example congestion control protocol shown in Table
- XCP – router informs source the rate at which it may send
- TCP with ECN – routers set bits to indicate congestion – reduction by how much?
- Fast TCP – depends on round-trip time (RTT) as an indication of congestion
- TCP with drop-tail or RED (e.g. CUBIC TCP) – loss of packets indicate congestion

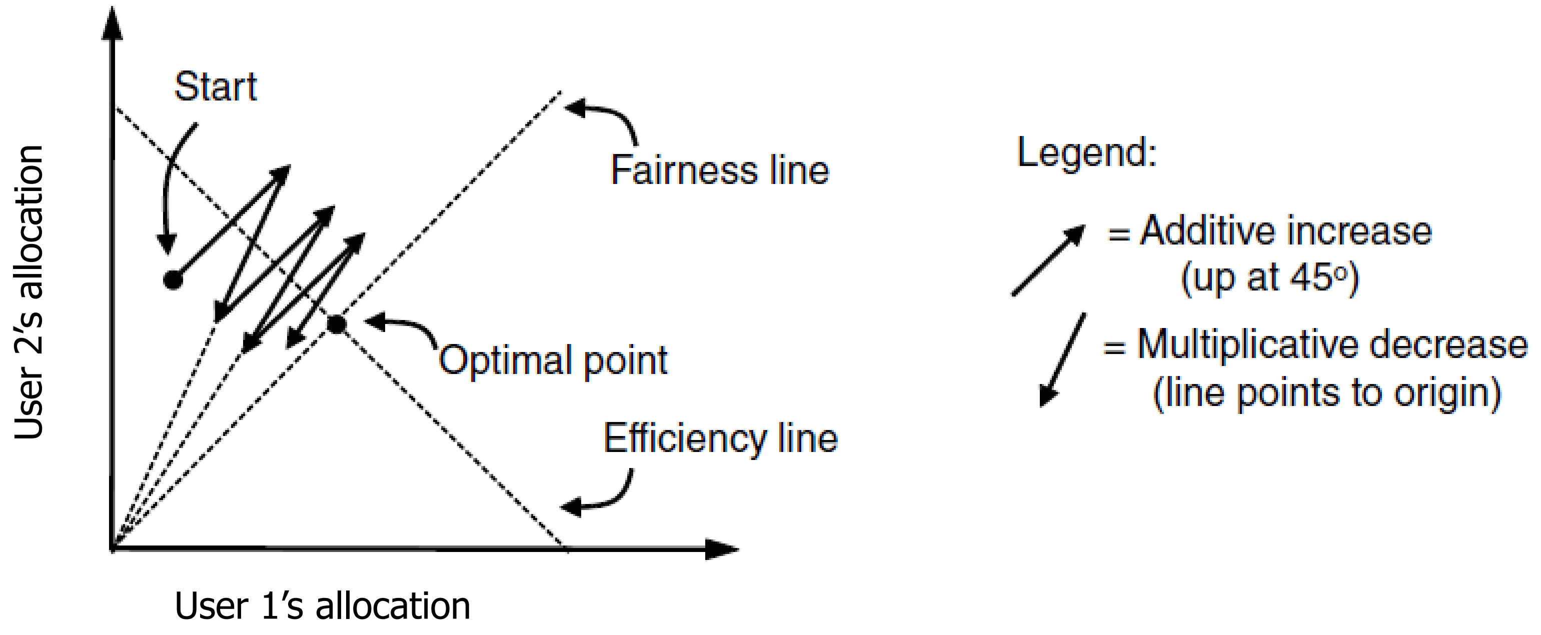
- Compound TCP – uses both packet loss and delay as feedback signals
- Based on signals given ➔ **<u>Control Law</u>** used to adjust how much is sent by senders



Protocol	Signal	Explicit?	Precise?
XCP	Rate to use	Yes	Yes
TCP with ECN	Congestion warning	Yes	No
FAST TCP	End-to-end delay	No	Yes
CUBIC TCP	Packet loss	No	No
TCP	Packet loss	No	No

Additive Increase Multiplicative Decrease (AIMD)

- AIMD is the control law used by TCP
- Fairness line – Efficiency line
- Additive increase additive decrease
- Multiplicative increase multiplicative decrease
- Optimal operating point
- AIMD ➔ slowly increase BW allocation and aggressive decrease policy



Additive Increase Multiplicative Decrease (AIMD) – cont'd

- TCP connections adjust their window sizes every RTT?
  - Favors connections with small RTT
  - TCP adjust the window size (W) - during RTT → sender rate = W/RTT
- This is combined with flow control
  - Therefore, can slow down in one RTT or stop is it stops getting ACKs from receiver
- TCP-friendly congestion control – TCP and non-TCP transport protocols can be used on same network

Wireless Issues

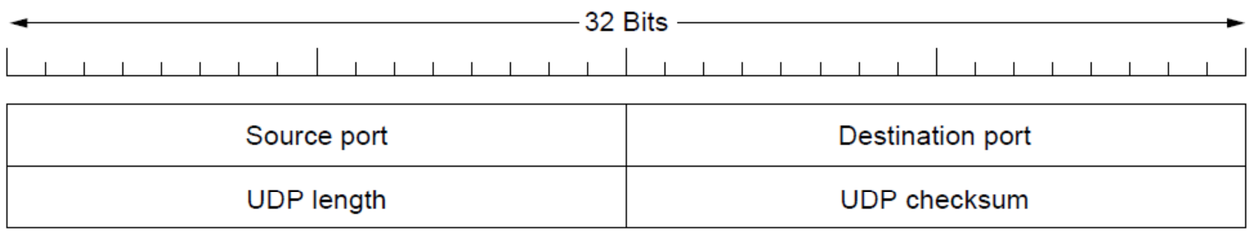
- Transport protocols (such as TCP) should be independent of the underlying network and link layer technologies
- Two problems:
  - (1) Interpretation of lost packets:
    - TCP – interprets packet loss as an indication of congestion, while on wireless links packet loss is often due to signal fades
  - (2) variable capacity of wireless links (variable SNR)!

Wireless Issues – cont’d

- Possible solution to problem (1): Mask losses on the wireless link using retransmission before reporting to upper layer
  - Time scale – retransmissions scale of milliseconds; Congestion control scale of seconds
  - Does this work for links with long round-trip times?
    - Use FEC
- Possible solution to problem (2): ignore it!
  - Congestion control algorithm handles the case of new connection coming into the network or existing users changing their sending rates
- A third solution is to design a specific transport protocol for wireless media – pros and cons?

The Internet Transport Protocol: UDP

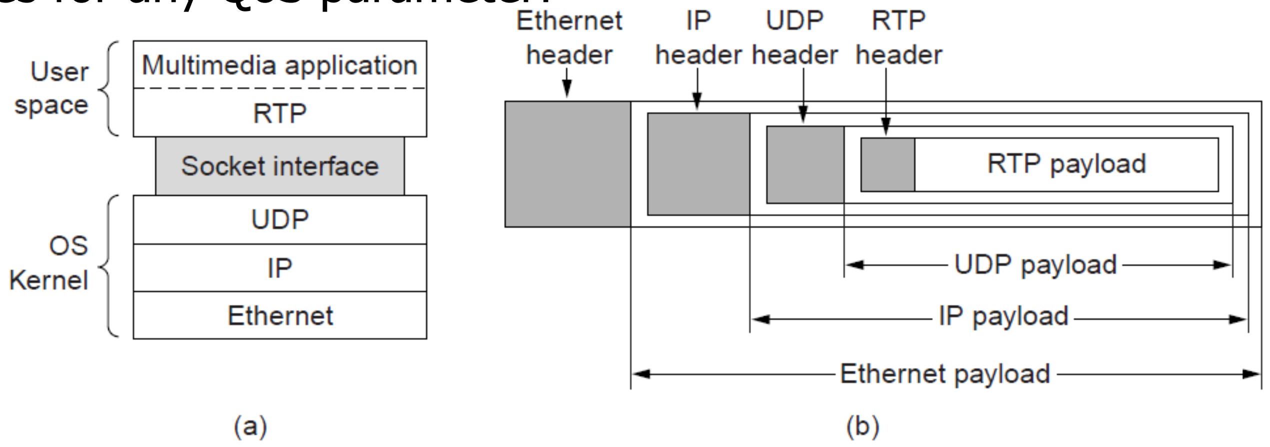
- User datagram protocol (UDP) – connectionless transport protocol
  - Applications send IP datagrams without establishing a connection
- Functions not included in UDP: flow control, congestion control, and retransmissions
- Functions in UDP: interface to IP layer, feature of demultiplexing multiple process using the ports, and optional error-detection
- Provides application with precise (all) control over the packet flow!
  - Simplifies development for client-server programming
- Used by application protocols such DNS and client-server RPC, real-time multimedia



The UDP header

Real-Time Transport Protocol (RTP)

- Real-Time Transport Protocol (RTP) – for transporting audio and video data packets
- Runs in space user over UDP – a generic transport protocol that happens to be implemented in the application layer
- Basic function – to multiplex several real-time data streams on to a single stream of UDP packets
  - Unicast – to single destination
  - Multicast – to multiple destinations
- No special guarantees for any QoS parameter!

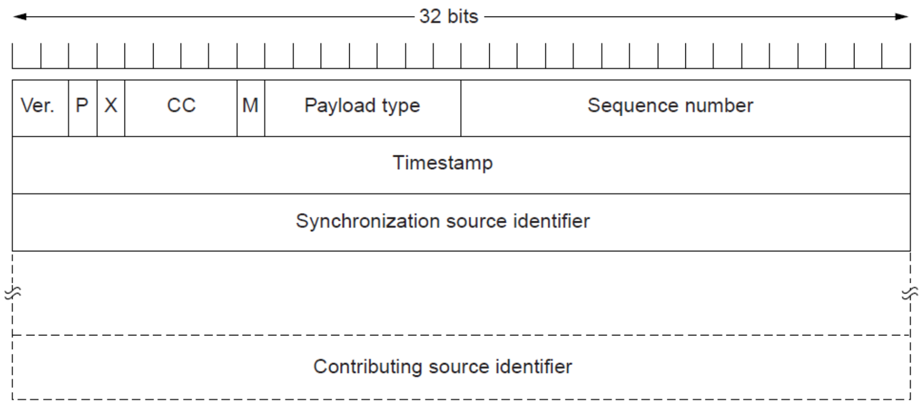


(a) The position of RTP in the protocol stack. (b) Packet nesting.

RTP – Features and Header Format

- **Sequence no** incremented on a stream-basis; dst can identify lost packets – application decides best course of action
  - Retransmission, replay, interpolate, etc.
- **Defines profiles** (e.g. single audio stream) – each profile allows multiple encoding schemes.
  - Payload may contain multiple samples – coded (e.g. 8-bit PCM, GSM encoding, MP3 encoding, etc.)
- **Timestamping** – relative (to first sample) effect:
  - Utilized in buffering and jitter reduction
  - May be used to synchronize amongst different streams

- 3 32-bit words header with potential extensions
- Fields: Ver, P (padding), X (header extended), CC (contributing sources), M (application specific mark bit)
- Payload Type: coding algorithm
- Sequence number
- Synch source identifier – used to identify the stream the packet belongs to



32 bits
Ver.
P
X
CC
M
Payload type
Sequence number
Timestamp
Synchronization source identifier
Contributing source identifier

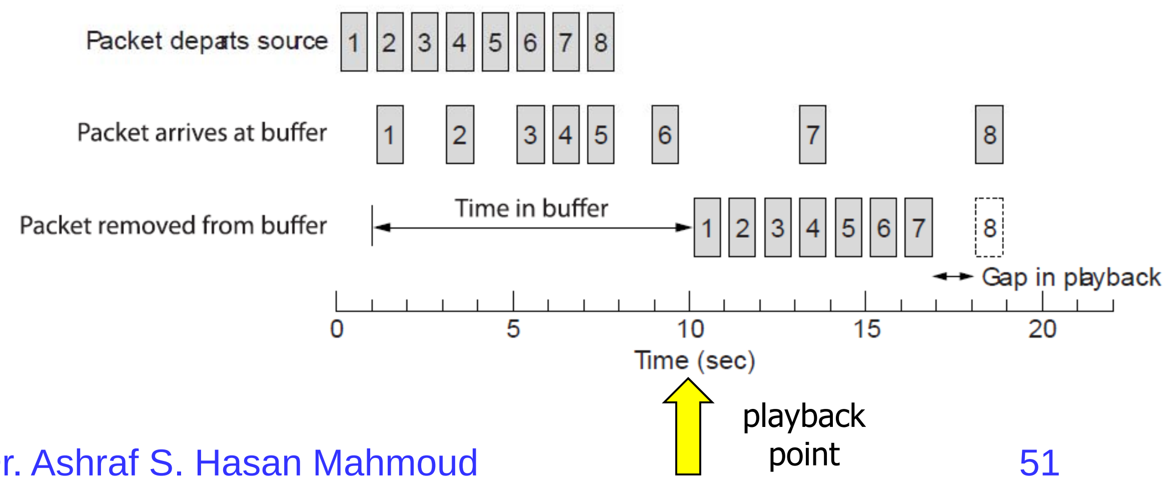
The RTP header format.

The Real-Time Transport Control Protocol: RTCP

- Handles feedback, synchronization, and user interface
- Not used for transporting any media samples
- Feedback may be used by src to adapt encoding algorithm to network status
    - PayloadType – is used always to tell encoding format for the sample
- RTCP also handles interstream synchronization
- Provides a method for naming sources in ASCII text

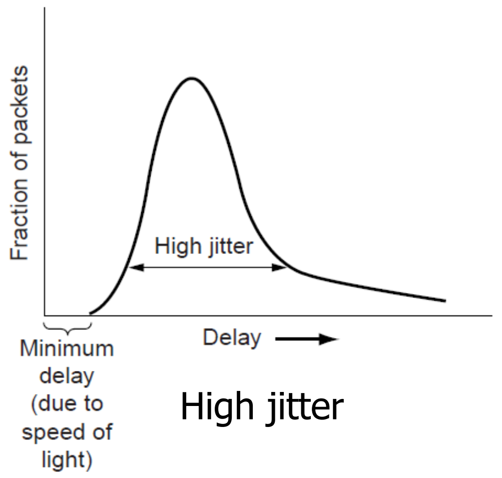
Playback with Buffering and Jitter Control

- Jitter – variation in delay
- Causes distracting media artifacts
- Solution – buffer packets at dst before playback – Refer to example below
  - Packets buffered for t = 10 sec
  - Packet 8 arrives late!
- Deciding on the **playback point**
  - Depends on the severity of the jitter
  - The M bit may be used by the application to compute playback points between talkspurts

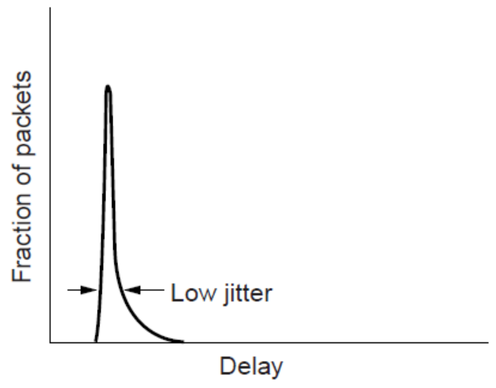


Playback with Buffering and Jitter Control – cont’d

- If playback point is too far to live applications, then
  - Application may have to accept larger fraction of packets being delayed (not available when playback is occurring)
  - Provide a better network capable of providing better QoS – e.g. expedited forwarding diffServ



High jitter

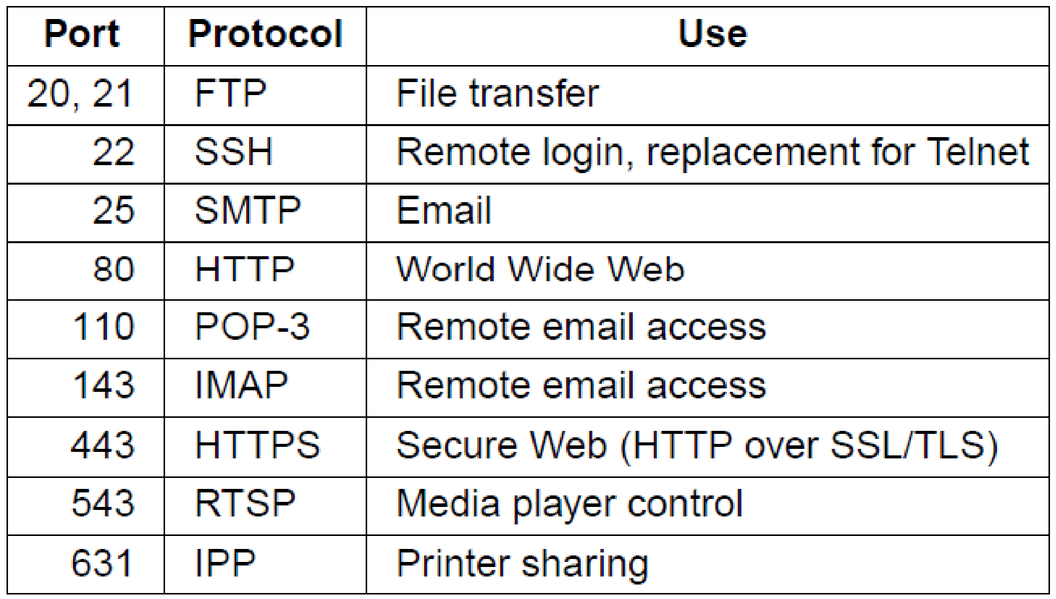


Low jitter

Transport Control Protocol (TCP) - Introduction

- Defined in 1981 – RFC 793; extended and refined several times (RFC 1122, RFC 1323, RFC 2018, RFC 2581, RFC 2988, RFC 3168, and RFC 4614)
- TCP transport entity
  - Manages TCP streams and interfaces to the IP layer
  - Accepts user data from application, breaks them to pieces not exceeding 64KB, and sends each piece as a separate IP packet
- TCP Service Model
  - Sockets = IP address + 16-bit port number (or TSAP)

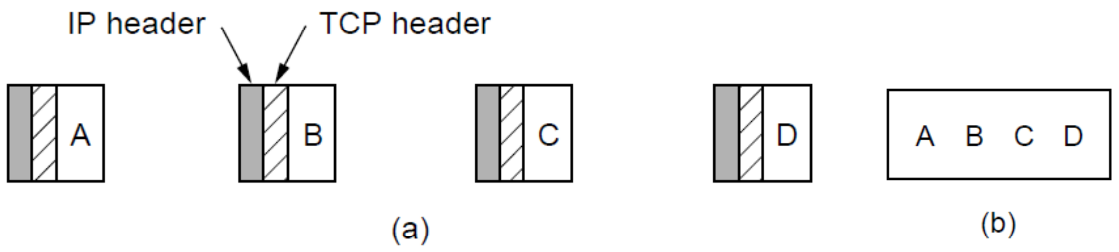
- More than one connection may terminate at the same socket
  - Connections are identified by the socket identifiers at both ends: (socket1, socket2)
- Well-known ports – ports below 1024 reserved for standard services



Port	Protocol	Use
20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

Transport Control Protocol (TCP) – Introduction – cont’d

- ALL TCP connections are **FULL-DUPLEX** and **point-to-point**
  - TCP DOES NOT support multicasting or broadcasting
- A TCP connection is a **byte stream** – not message stream!
  - Message boundaries are NOT PRESERVED end-to-end
  - Example – four 512 Bytes chunks are written by the application to a TCP stream – they may be delivered to receiving process as four 512 Bytes chunks, two 1024 Bytes chunks, or one 2048 Bytes chunk.
  - “There is no way for the receiver to detect the unit(s) in which the data were written”
- TCP may send data immediately or buffer it
  - Uses of PUSH flag and “urgent data”



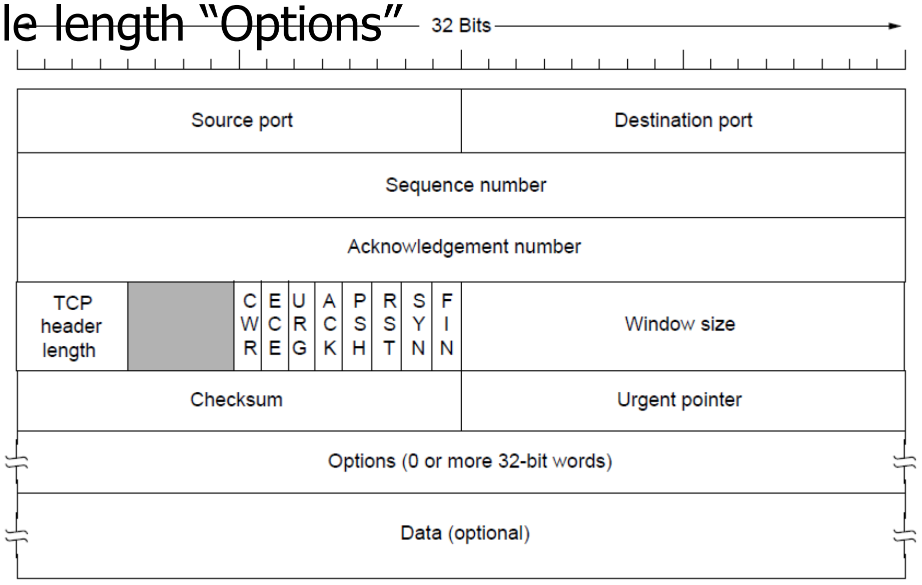
(a) Four 512-byte segments sent as separate IP diagrams
(b) The 2048 bytes of data delivered to the application in a single READ call

TCP Segment and Segment Header

- Byte sequence numbers carried for:
  - Sliding window position in one directions, and
  - Acknowledgments in the reverse direction
- Sending and receiving TCP entities exchange "segments"
  - 20-Bytes header (plus optional part), and
  - Zero or more data bytes
- Two limits:
  - TCP header plus data must fit in 65,515 byte IP payload
  - Maximum Transfer Unit (MTU) for link – to avoid fragmented packets
    - E.g. 1500 Bytes for Ethernet
- Path MTU discovery – to avoid fragmenting IP packets; utilizes ICMP to find smallest MTU on the path
- Basic protocol – sliding window with a dynamic (i.e. *variable*) window size
  1. Start a timer when segment is sent
  2. Dst sends back ACK
  3. Src resends segment is timer goes off before ACK is received
- Potential problems:
  1. Segments may arrive out of order – can not immediately send ACK!
  2. Segments may be delayed leading to retransmissions from src; retransmission may include different byte ranges → required careful administration

TCP Segment and Segment Header – cont'd

- Fixed 20-bytes header
- Maximum of 65,515 – 20 = 65,495 bytes data
- Source port and destination port – define the two ends of the connection
- Connection identifier – 5 tuple: protocol (TCP), src IP and port, dst IP and port.
- Sequence no and ACK no – flow and error control
  - TCP uses cumulative acknowledgments
- TCP header length – to support variable length "Options"
- Eight 1-bit flags
  - CWR and ECE – used to signal congestion when ECN is used
  - URG – urgent pointer from current sequence # to urgent data
  - ACK – ACK no is valid – default case for nearly all packets
  - PSH – pushed data
  - RST – reset connection, reject invalid segment, or refuse connection open
  - SYN – to establish connection
  - FIN – to terminate connection



TCP Segment and Segment Header – cont'd

- Window size – sliding window size used for flow control
- Options –
  1. Maximum Segment Size (MSS) – All Internet hosts are required to accept TCP segments of 536 + 20 = 556 Bytes
  2. Window scale option – shift window size field by up to 14 bits to the left ➔ window up to $2^{(16+14)} = 2^{30}$ Bytes!
  3. Timestamp option –
     - Enables sender to estimate RTT
     - Used to extend the 32-bit sequence numbers – PAWS scheme discards arriving packets with old timestamps to prevent confusing between old and new data due to sequence no wrapping.
  4. Selective Acknowledgment (SACK) – Allows receiver to inform sender of ranges of sequence numbers it has received
     - Used after a packet is lost but subsequent (or duplicate) data has arrived

TCP Connection Establishment

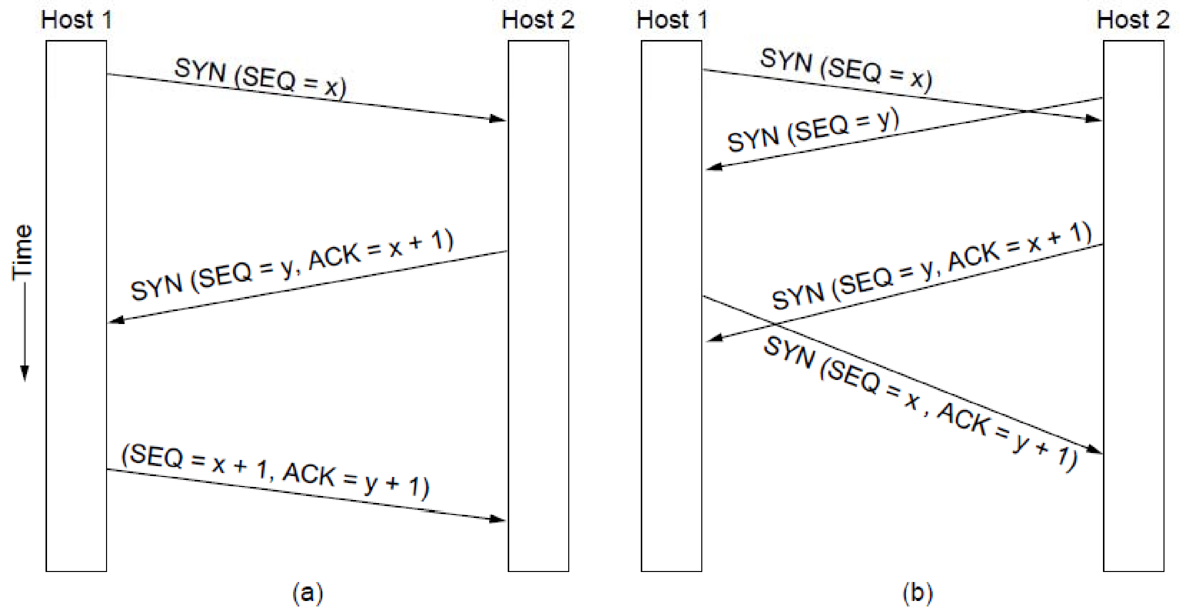
- Three-way handshake protocol – as explained in previous slides
- Server waits for incoming call: LISTEN and ACCEPT primitives
- Clients issue connection request
  - Specifies dst IP-port, MSS, some optional data (e.g. password)
  - CONNECT primitive sends a TCP segment with SYN bit on and ACK bit off and waits for response

(a) TCP connection establishment in the normal case.

- Note that SYN segment consumes one byte of sequence space!

(b) Simultaneous connection establishment on both sides.

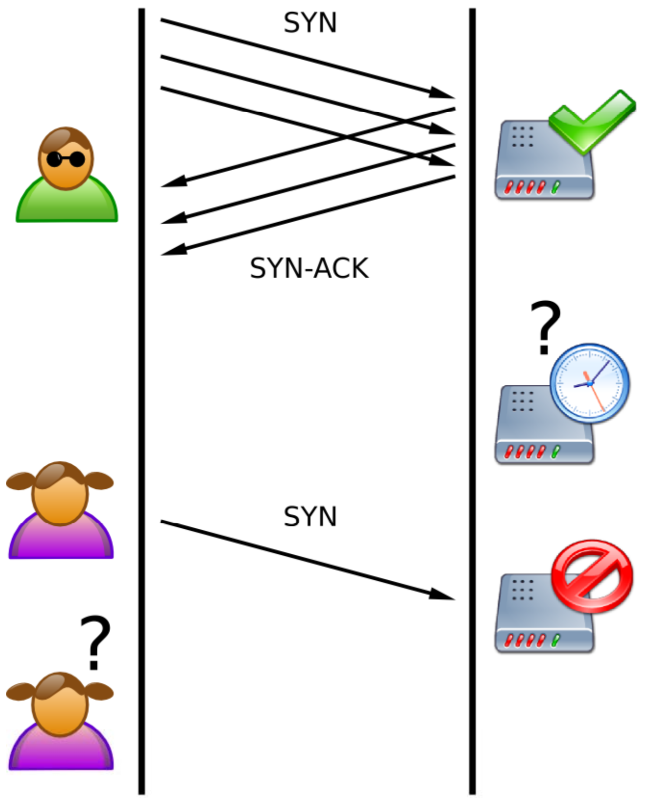
- The result is ONE connection opened since it is defined by the two end points.



Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y, ACK = x + 1)
(SEQ = x + 1, ACK = y + 1)
Time
(a)
Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y)
SYN (SEQ = y, ACK = x + 1)
SYN (SEQ = x, ACK = y + 1)
(b)

TCP Connection Establishment – cont’d

- Initial sequence number cycles slowly
  - Clock-based scheme – clock tick = 4 microseconds
- **SYN flood attack** – half open connections!
  - Solution – **SYN cookies** – host chooses a cryptographically generated seq no and puts in the outgoing segment and forgets it
    - If it receives an ACK (seq + 1), it can regenerate the same seq no (same input: other host IP/port addresses, local secret, etc.)
  - Host is able to check whether an acknowledgement seq no is valid without having to remember it!
- More details available on: http://en.wikipedia.org/wiki/SYN_cookies



TCP Connection Release

- Full-duplex = pair of simplex connections
  - Each released independently
- TCP segment with FIN bit set
- Four segments are needed (as shown) to close connections
  - (2) and (3) may be combined
- Two-army problem
  - Timers are set when FIN is sent
  - If response does not come in twice maximum packet lifetime, the sender of the FIN releases connection
  - Other side will notice after some time and time out (and release connection too)

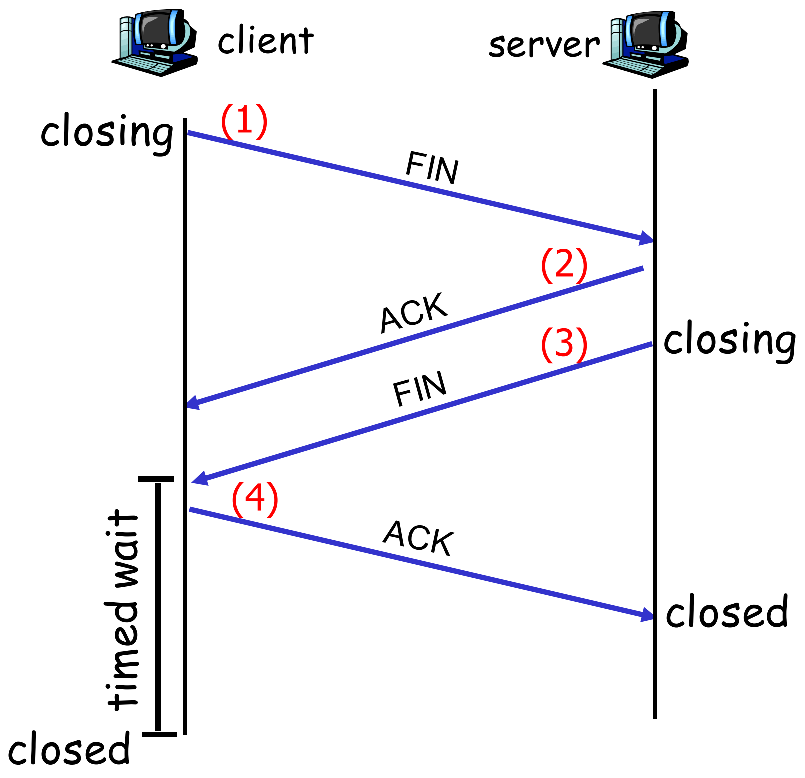
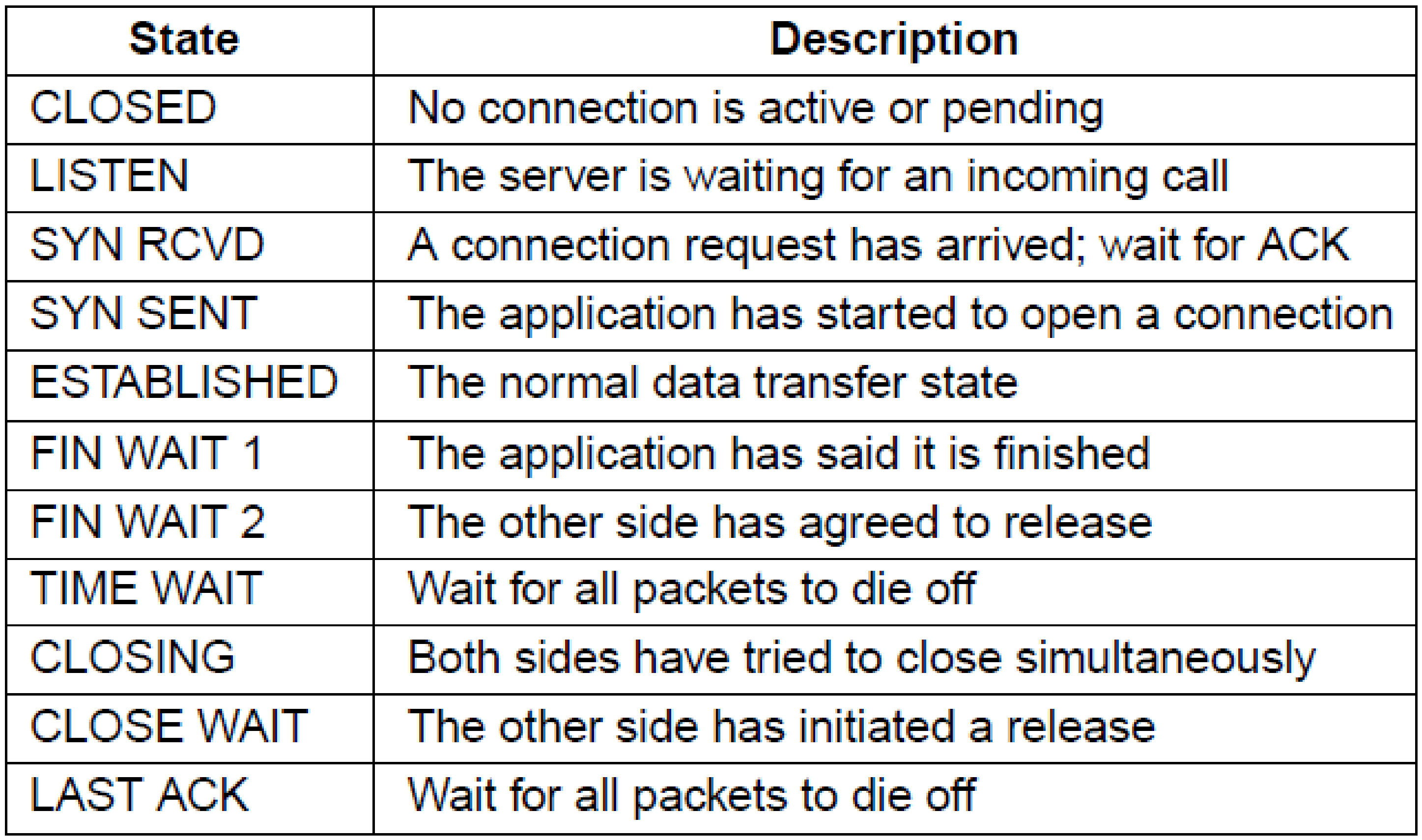


Figure from Kurose's textbook.

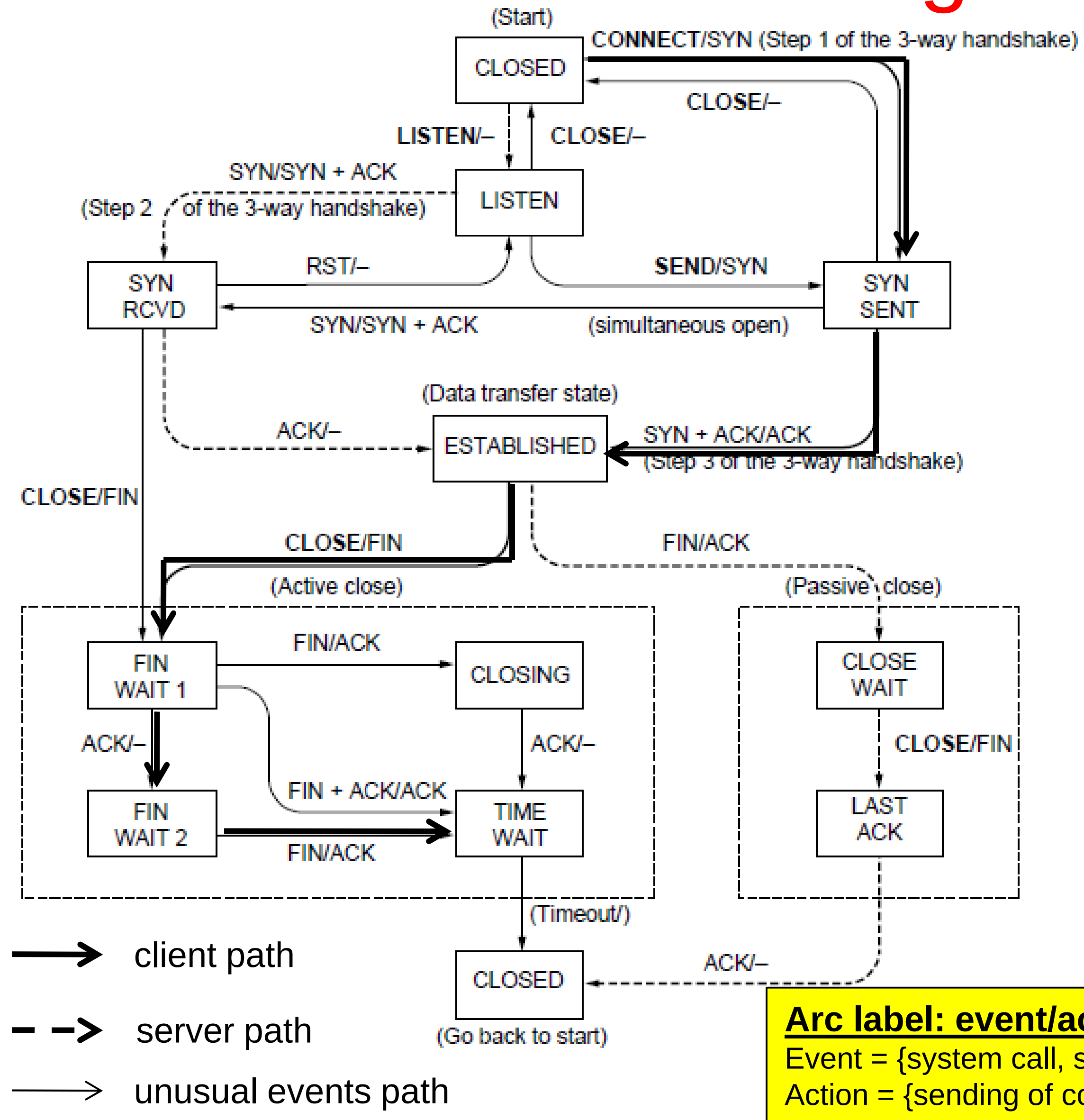
TCP Connection Management Modeling (1)



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 application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIME WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

The states used in the TCP connection management finite state machine.

TCP Connection Management Modeling (2)



TCP connection management finite state machine.

The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events. Each transition is labeled by the event causing it and the action resulting from it, separated by a slash.

Arc label: event/action

Event = {system call, segment arrival (SYN, FIN, ACK, RST), or timeout}

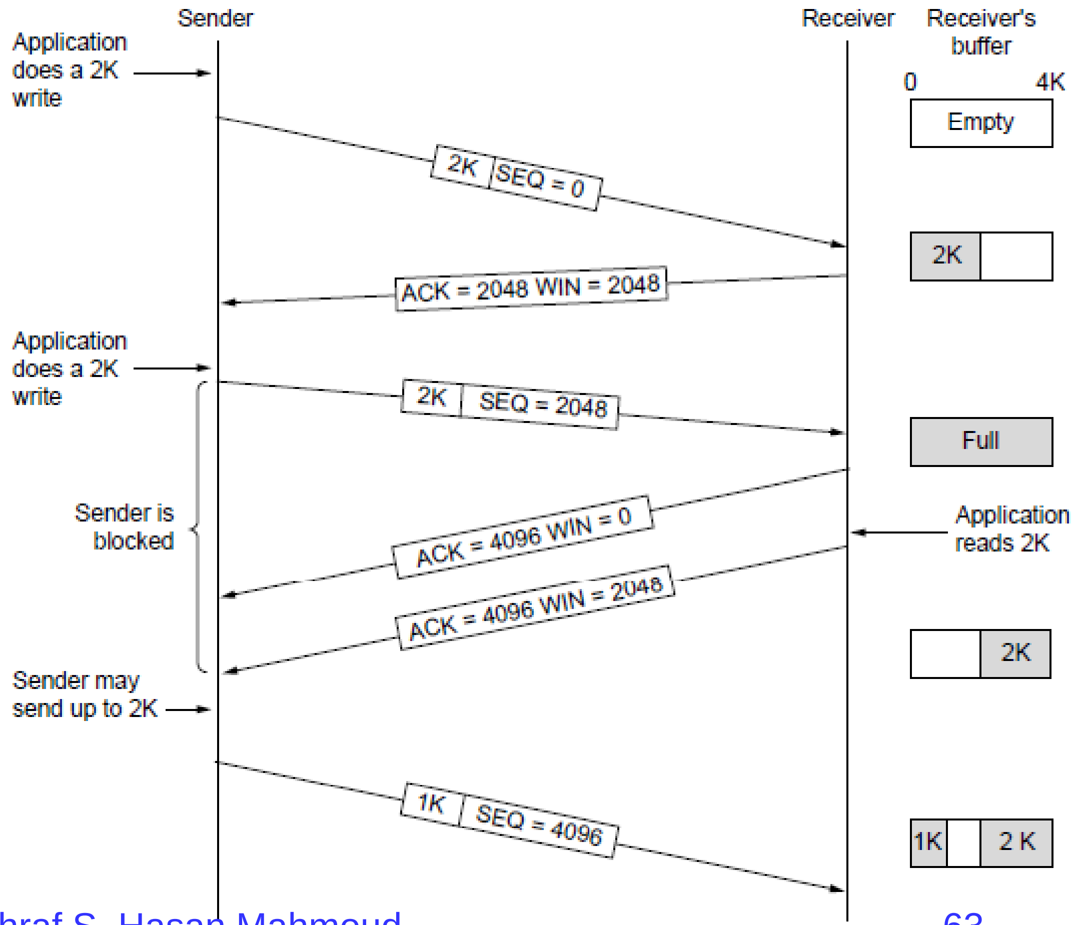
Action = {sending of control segment (SYN, FIN, or RST), or nothing "-"}

Comments are shown between ( )

TCP Sliding Window

- See example of window management in TCP
- WIN = 0 is used to block sender!

- Two exceptions:
  - Urgent data (e.g. to kill a process)
  - Window probe – one-byte segment to force receiver to announce the next byte expected and the window size – used to prevent deadlocks
- Senders are NOT required to send data as soon it becomes available
- Receivers are NOT required to send ACKs immediately after they receive data



TCP Sliding Window – Delayed Acknowledgement and Nagle's Algorithm

- How to improve efficiency
- At receiver side – **delayed acknowledgement**
  - Wait up to 500 msec before sending ACK – to *benefit* from accumulated ACK effect
- What if sender sends many short packets
  - Use **Nagle's algorithm** – only one short packet is outstanding at any time
  - Not good for interactive gaming
  - May create deadlocks – refer to textbook page 585
  - May be disabled – TCP_NODELAY option!

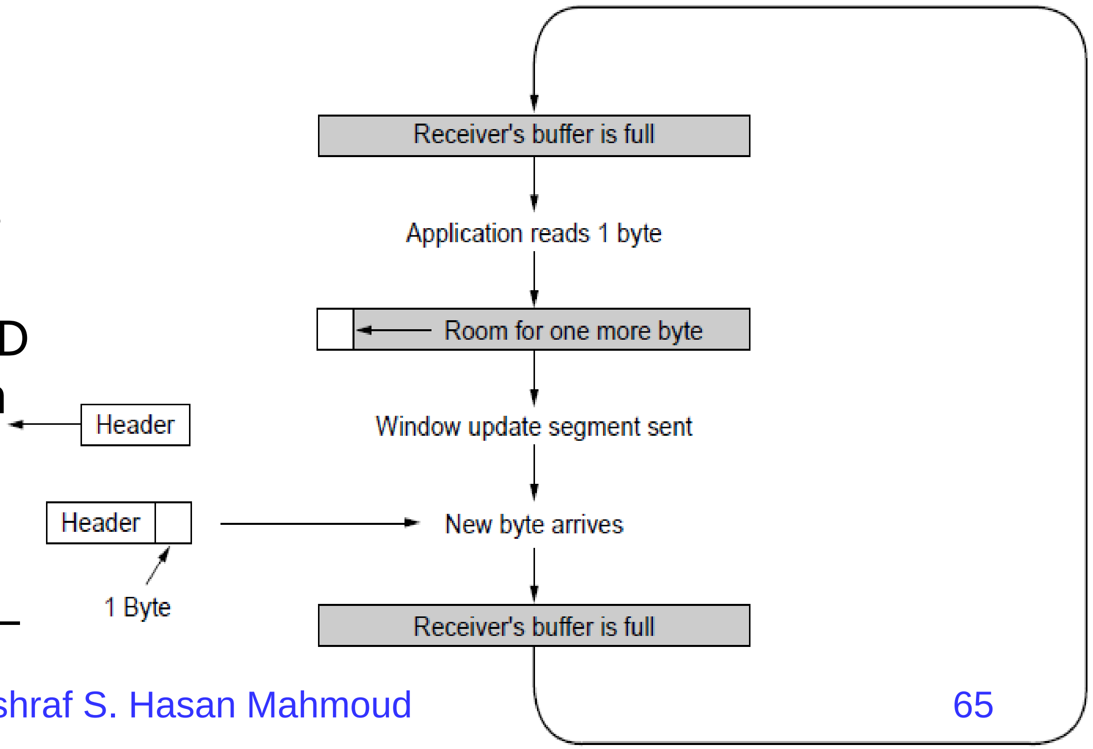
Nagle's Algorithm:

1. Send 1st piece
2. Buffer all the rest until 1st piece is ACKed
3. Send all buffered data in one TCP segment
4. Start buffering again until segment is ACKed
5. Etc.

(default) New segment is sent if MSS is reached.

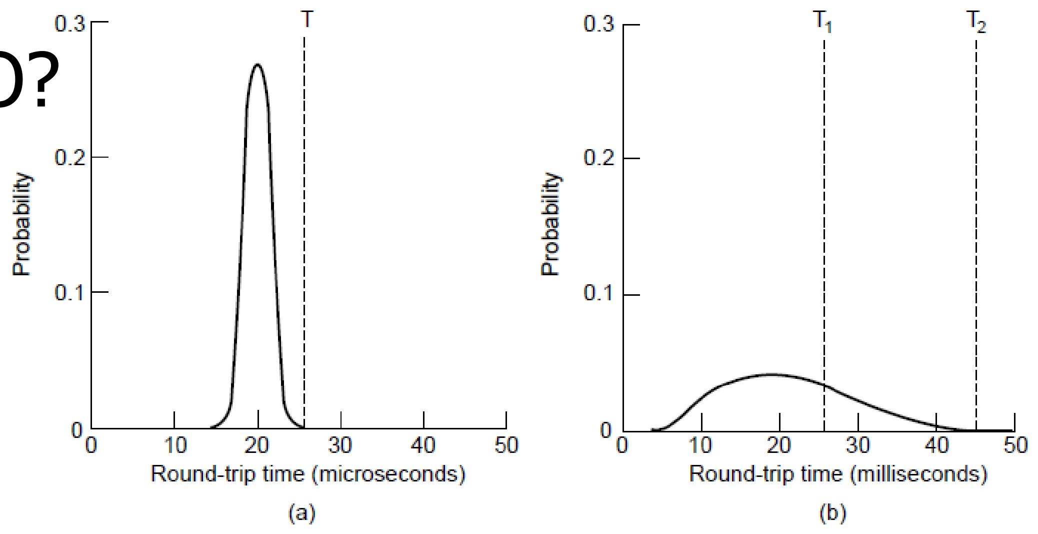
TCP Sliding Window – Silly Window Syndrome

- Occurs when data is passed to the sending entity in large blocks but the application on the receiver side is reading 1 byte at a time!
- Clark's solution
  - Receiver should advertise only reasonable window sizes
  - Size to advertise = min (MSS, or half original receiver buffer)
- Sender should also not send tiny segments – Nagle's algorithm
- Nagle's algorithm and Clark's solution are complementary
- Receiver may also block READ requests from the application till large chunks of data are available
  - Decreases no of calls to TCP and increases response time – OK for non-interactive applications



TCP Timer Management

- RTO (Retransmission TimeOut) – when segment is sent a timer is started
  - If RTO expires before ACK is received, the segment is retransmitted
- How to compute RTO?



(a) Probability density of acknowledgment arrival times in data link layer.

(b) .... for TCP

TCP Timer Management – Computation of RTO

$$RTO = SRTT + 4 \times RTTVAR \quad \text{--------} \; (1)$$

$$RTTVAR = \beta\, RTTVAR + (1-\beta)\, |SRTT - R| \quad \text{-----}(2)$$

$$SRTT = \alpha\, SRTT + (1-\alpha) R \quad \text{-------} \; (3)$$

- R is the time it took the ACK to get back
- SRTT – smoothed round trip time – best current estimate to the RTT to the destination in question
- α and β are smoothing factors < 1; typically $\alpha = 7/8$ and $\beta = 3/4$.
- RTTVAR – round trip time variation
- Karen's Algorithm – R is collected for un-retransmitted segments
- EWMA – Exponentially Weighted Moving Average

TCP Persistence Timer

- To prevent the following deadlock
  - Receiver sends an ACK with WIN = 0
  - Later, buffer becomes available
  - Receiver sends ACK with WIN = X (some buffer allocation) – This segment gets lost!
- When persistence timer goes off
  - Sender sends probe to receiver
  - Receiver responds to probe with WIN size
- If response WIN size = 0, persistence time is reset

TCP Keepalive and TIME-WAIT Timers

- Keepalive timer
  - Connection idle for long time
  - Keepalive timer goes off causes one side to check whether the other side is still there
  - If it fails, connection is terminated
- It may terminate otherwise healthy connections!
- TIME-WAIT timers – used in the TCP connection close state
  - It runs for twice the maximum packet life-time to make sure that when a connection is closed, all packets created by it have died off

TCP Congestion Control (1)

TCP uses AIMD with loss signal to control congestion

- Implemented as a congestion window (cwnd) for the number of segments that may be in the network
- Uses several mechanisms that work together

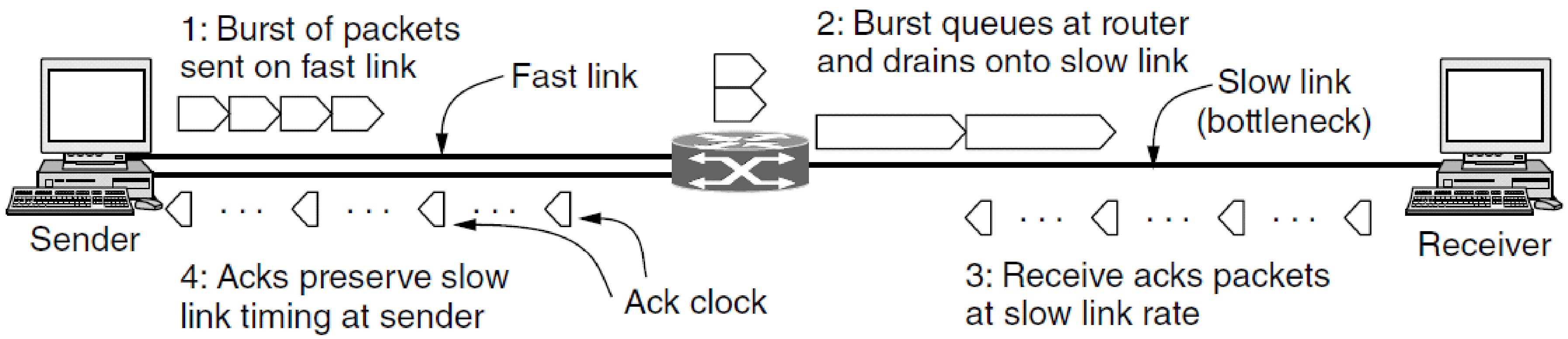
Name	Mechanism	Purpose
ACK clock	Congestion window (cwnd)	Smooth out packet bursts
Slow-start	Double cwnd each RTT	Rapidly increase send rate to reach roughly the right level
Additive Increase	Increase cwnd by 1 packet each RTT	Slowly increase send rate to probe at about the right level
Fast retransmit / recovery	Resend lost packet after 3 duplicate ACKs; send new packet for each new ACK	Recover from a lost packet without stopping ACK clock

Textbook author’s slides

TCP Congestion Control (2)

Congestion window controls the sending rate

- Rate is cwnd / RTT; window can stop sender quickly
- **ACK clock** (regular receipt of ACKs) paces traffic and smoothes out sender bursts


1: Burst of packets sent on fast link
Fast link
2: Burst queues at router and drains onto slow link
Slow link (bottleneck)
Sender
Receiver
4: Acks preserve slow link timing at sender
Ack clock
3: Receive acks packets at slow link rate

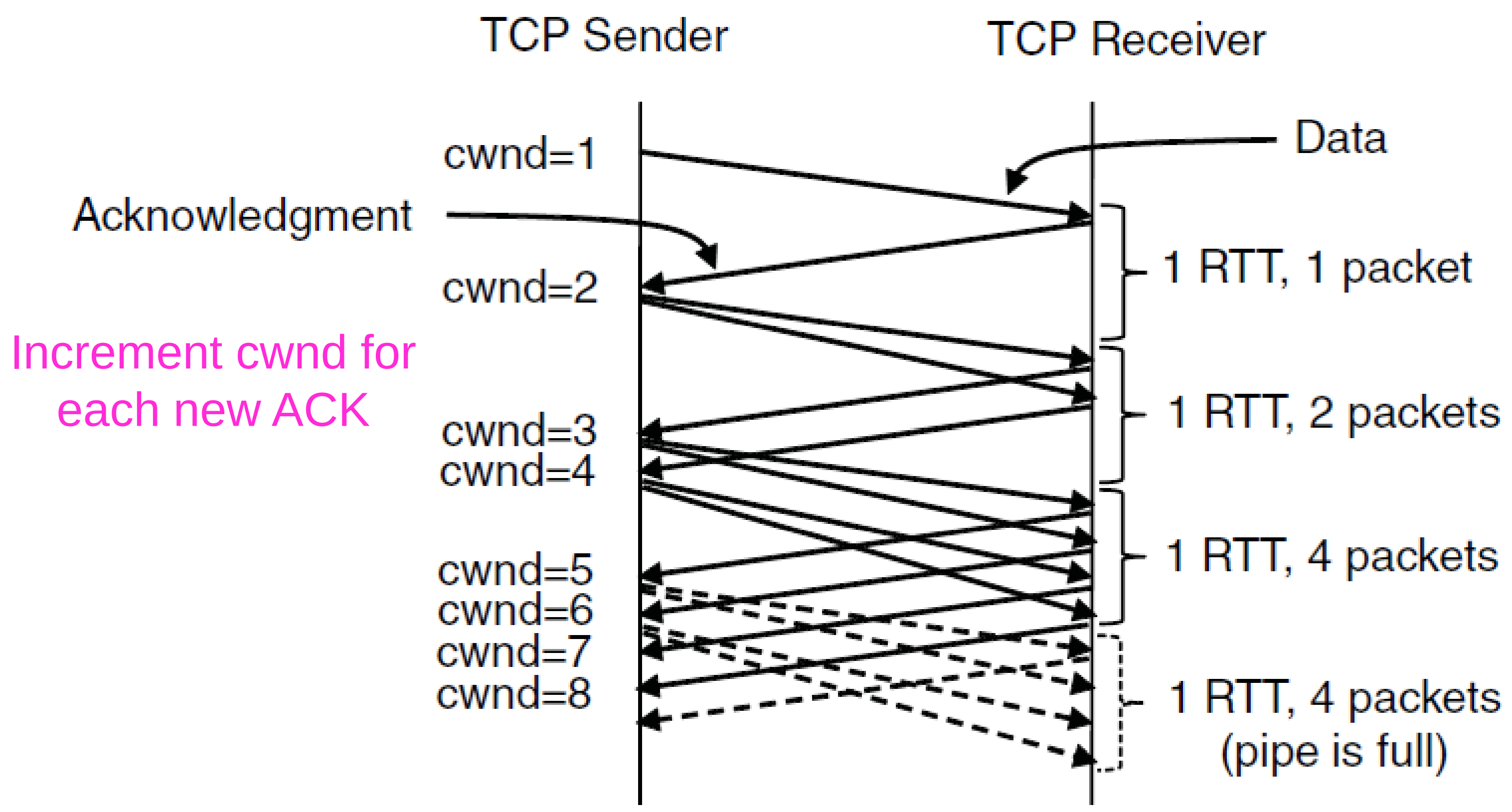
ACKs pace new segments into the network and smooth bursts

Textbook author's slides

TCP Congestion Control (3)

Slow start grows congestion window exponentially

- Doubles every RTT while keeping ACK clock going

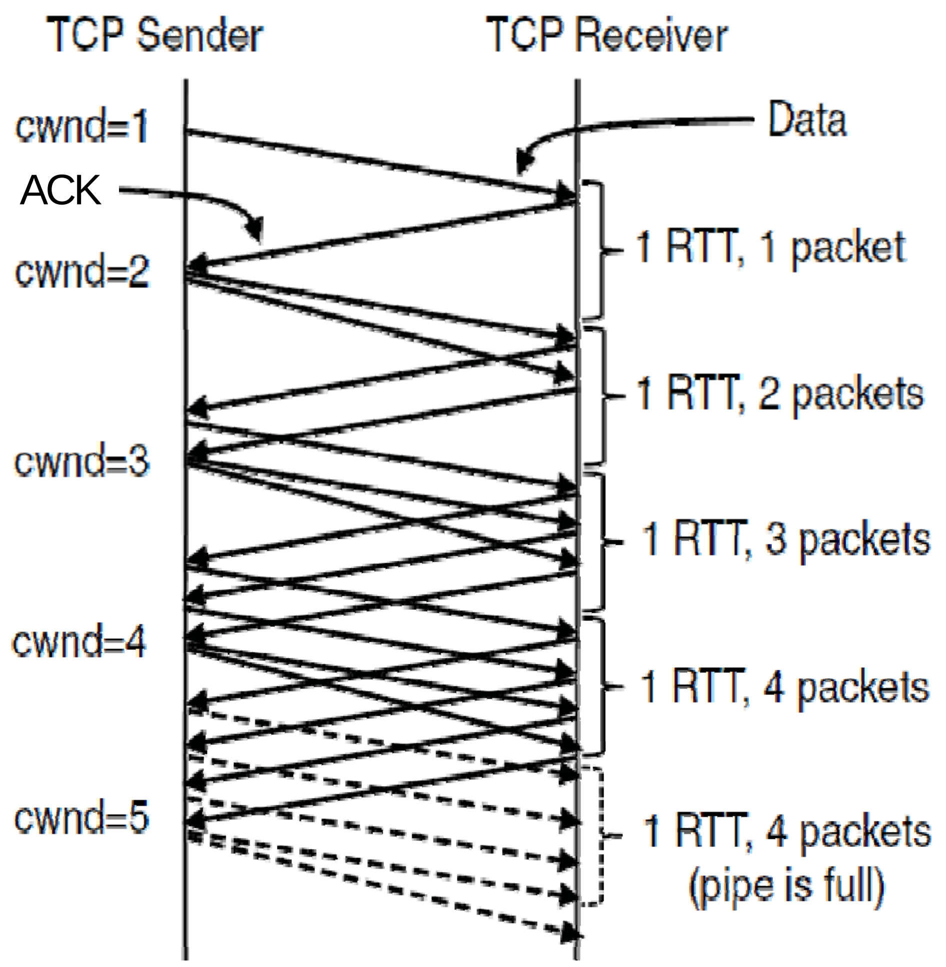


Textbook author's slides

TCP Congestion Control (4)

Additive increase grows cwnd slowly

- Adds 1 every RTT
- Keeps ACK clock

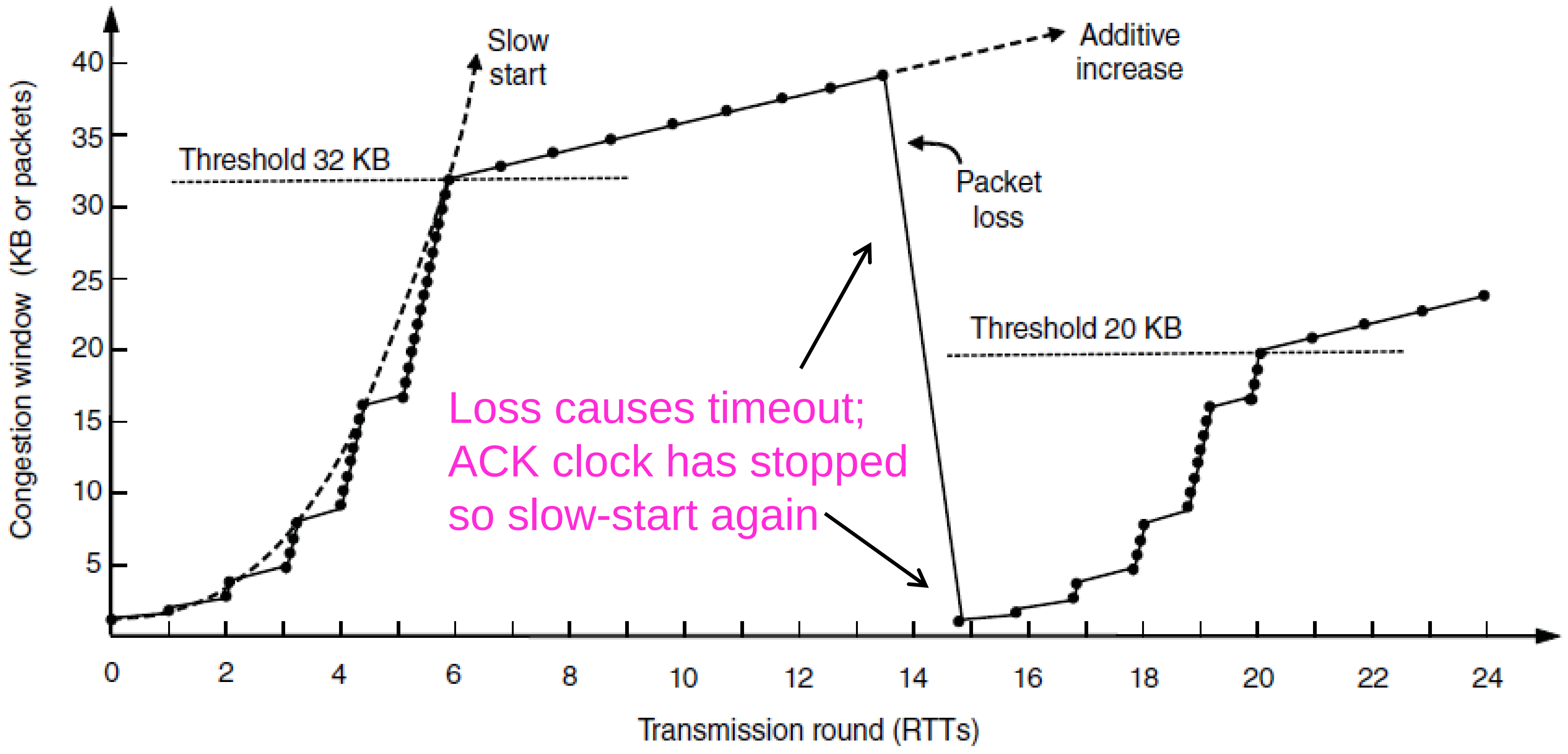


Textbook author's slides

TCP Congestion Control (5)

Slow start followed by additive increase (TCP Tahoe)

- **Threshold** is half of previous loss cwnd

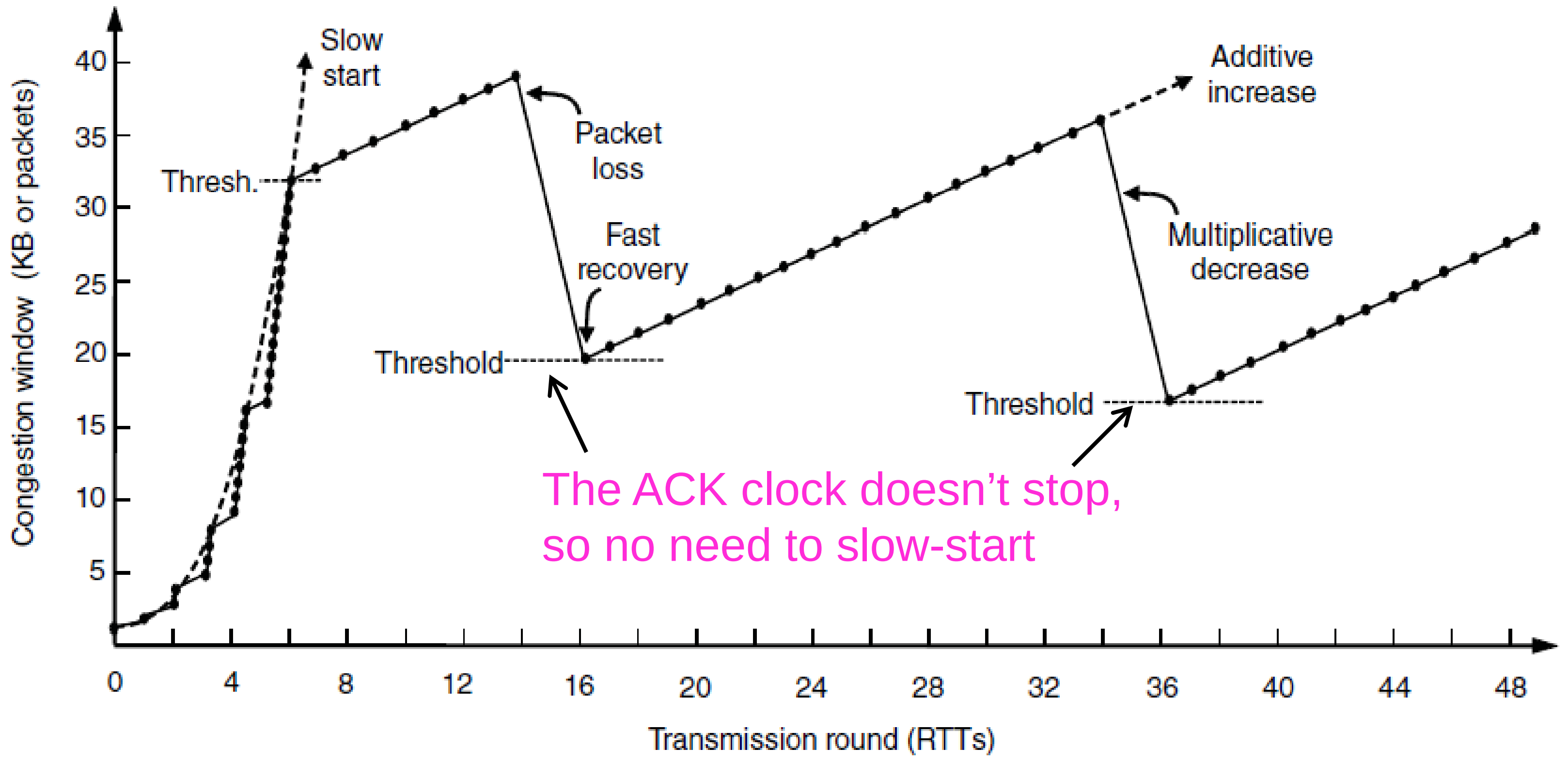


Textbook author's slides

TCP Congestion Control (6)

With **fast recovery**, we get the classic sawtooth (TCP Reno)

- Retransmit lost packet after 3 duplicate ACKs
- New packet for each dup. ACK until loss is repaired

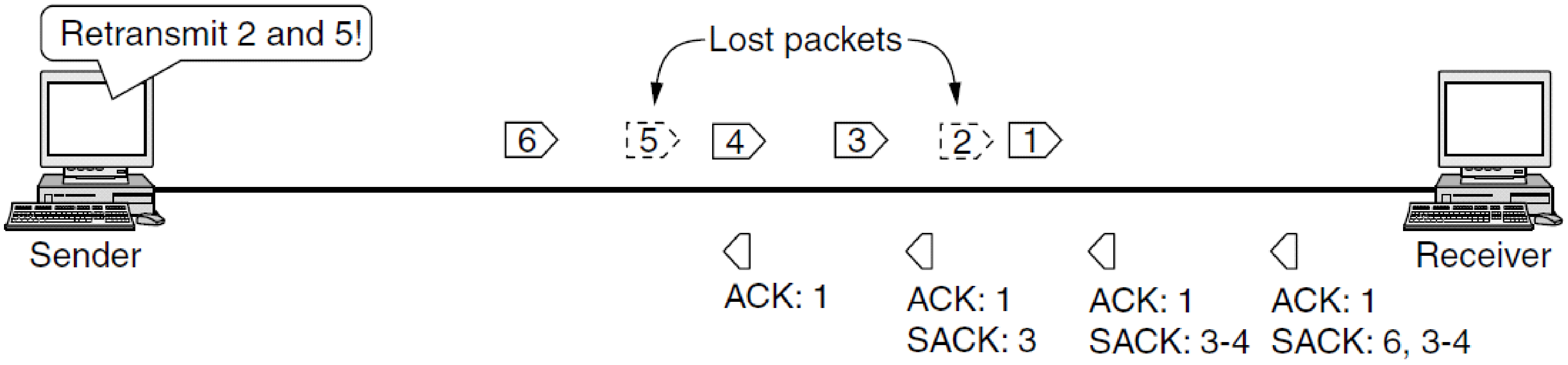


Textbook author's slides

TCP Congestion Control (7)

SACK (Selective ACKs) extend ACKs with a vector to describe received segments and hence losses

- Allows for more accurate retransmissions / recovery



No way for us to know that 2 and 5 were lost with only ACKs