Transport Layer

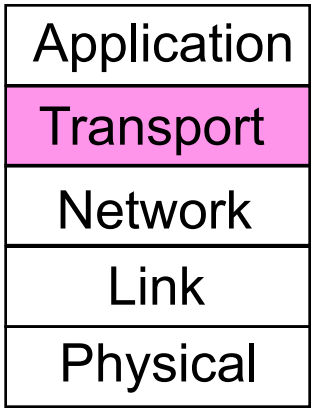
COMP90007

Internet Technologies

Chien Aun Chan

Outline

- Transport Layer Services
- Transport Layer Primitives
- Elements of Transport Protocols
- Sockets



The Transport Service

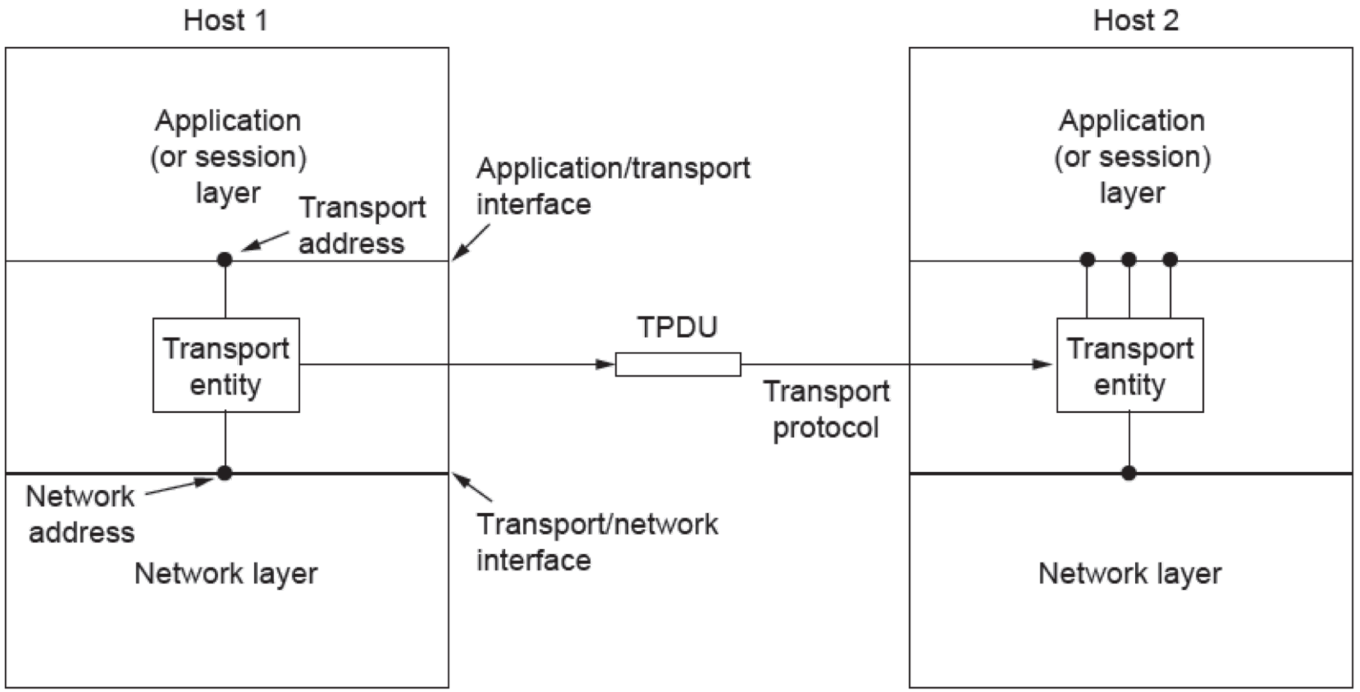
- Primary function
  - provide efficient, reliable & cost-effective data transmission service to the processes in the application layer, independent of physical or data networks
- To Achieve this
  - It uses service provided by the network layer
- *The Transport entity*: the software or hardware within the transport layer that does the work.

Transport Layer Services

- Transport Layer Services provide interfaces between the Application Layer and the Network Layer
- Transport Entities (the hardware or software which actually does the work) can exist in multiple locations:
- Common
  - OS kernel
  - System library (library package bound into network applications)
- Not so common
  - User process
  - NIC

Services

- Transport layer adds *reliability* to the network layer
  - Offers connectionless (e.g., UDP) and connection-oriented (e.g, TCP) services to applications
- Relationship between network, transport and application layers:



Transport Layer and Network Layer Services Compared

- If transport and network layers are so similar, why are there two layers?
- Transport layer code runs entirely on hosts
- Network layer code runs almost entirely on routers
- Transport layer can fix reliability problems caused by the Network layer (e.g., delayed, lost or duplicated packets)
- Users have no real control over the network layer – Transport layer: improve QoS

Role of the Transport Layer

- The Transport Layer occupies a key position in the layer hierarchy because it clearly delineates
  - providers of (reliable) data transmissions services
    - at the network, data link, and physical layers
  - users of reliable data transmission services
    - at the application layer
- In particular, connection-oriented transport services provide a reliable service on top of an unreliable network

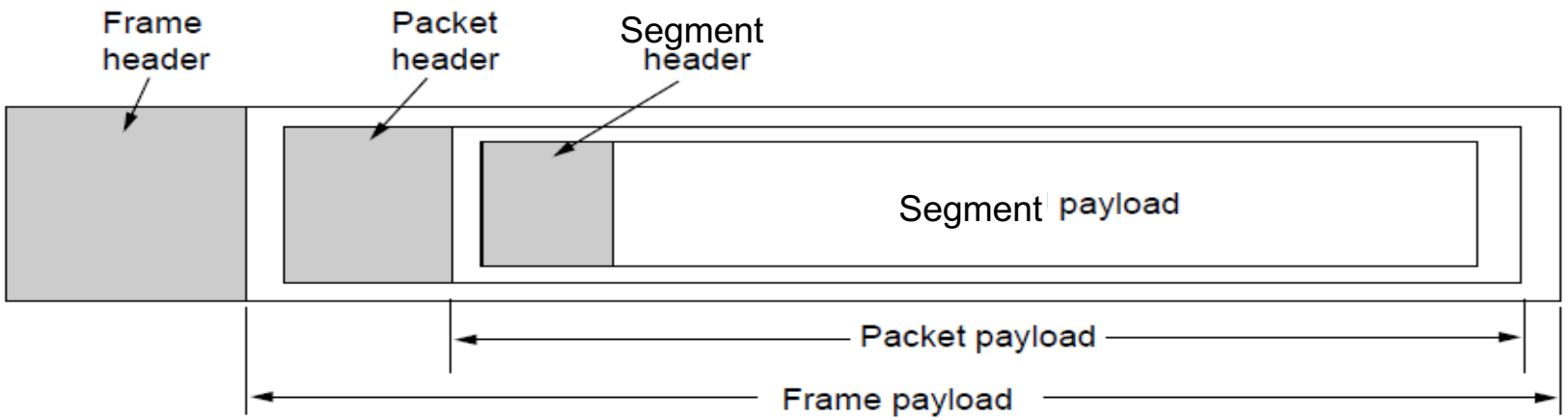
Features of Transport Layer

- Mechanism for improved QoS for users
  - Reliability at application level through interface with network layer
- Abstraction and primitives provide a simpler API for application developers independent of network layer

Primitive	Meaning
LISTEN	Block waiting for an incoming connection
CONNECT	Establish a connection with a waiting peer
RECEIVE	Block waiting for an incoming message
SEND	Send a message to the peer
DISCONNECT	Terminate a connection

Transport Layer Encapsulation

- Abstract representation of messages sent to and from transport entities
  - Transport Protocol Data Unit (TPDU)
  - segment
- Encapsulation of TPDUs (transport layer units) in packets (network layer units) in frames (data layer units)



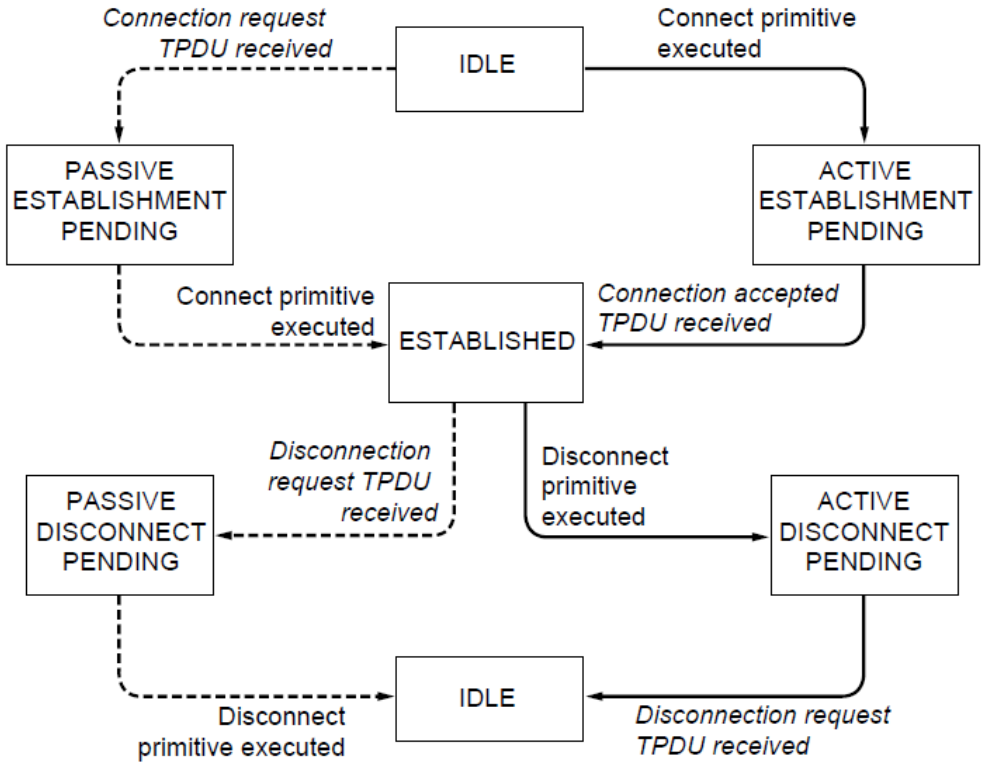
Transport Service Primitives

- Primitives that applications might call to transport data for a simple connection-oriented service:
  - Server executes LISTEN
  - Client executes CONNECT
    - Sends CONNECTION REQUEST TPDU to Server
    - Receives CONNECTION ACCEPTED TPDU to Client
  - Data exchanged using SEND and RECEIVE
  - Either party executes DISCONNECT

Primitive	Segment 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

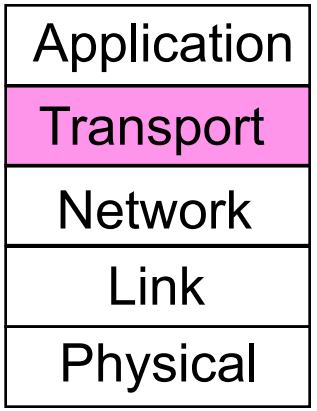
Simple Connections Illustrated

- Solid lines (right) show client state sequence
- Dashed lines (left) show server state sequence
- Transitions in italics are due to segment arrivals



Elements of Transport Protocols

- Connection establishment
- Connection release
- Addressing



Connection Establishment in the Real World

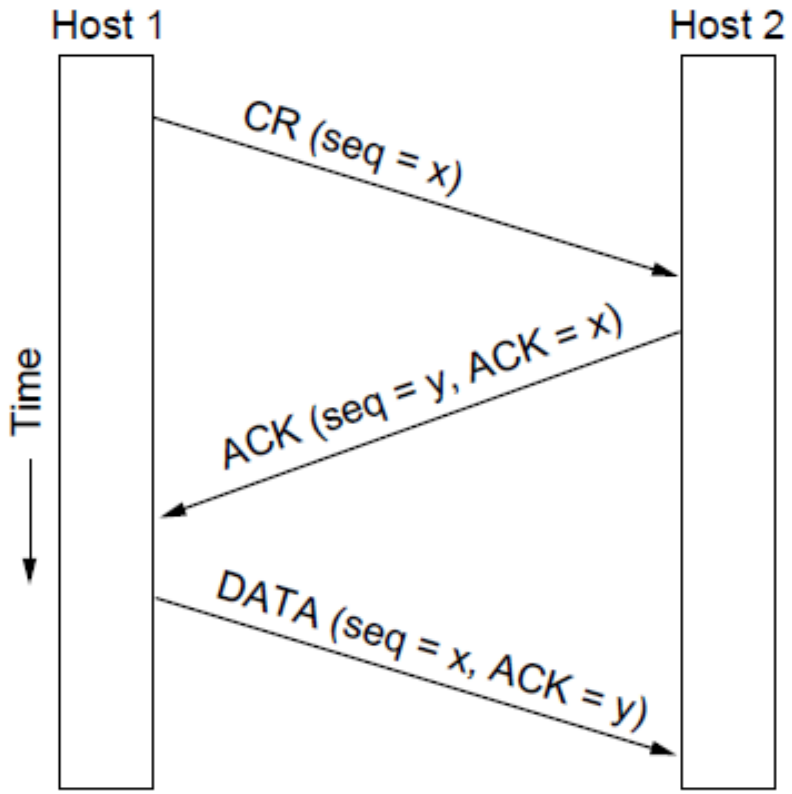
- When networks can lose, store and duplicate packets, connection establishment can be complicated
  - congested networks may delay acknowledgements
  - incurring repeated multiple transmissions
  - any of which may not arrive at all or out of sequence – delayed duplicates
  - Applications degenerate with such congestion (eg. Imagine duplication of bank withdrawals)

Reliable Connection Establishment

- Key challenge is to ensure reliability even though packets may be lost, corrupted, delayed, and duplicated
  - Don't treat an old or duplicate packet as new
  - (Use ARQ and checksums for loss/corruption)
- Approach:
  - Don't reuse Maximum Segment Lifetime sequence numbers within twice the MSL (2min)
  - Three-way handshake for establishing connection
  - Use a sequence number space large enough that it will not wrap, even when sending at full rate

Three Way Handshake

- Three-way handshake used for initial packet
  - Since no state from previous connection
  - Both hosts contribute fresh seq. numbers
  - CR = Connect Request



Three Way Handshake (Working)

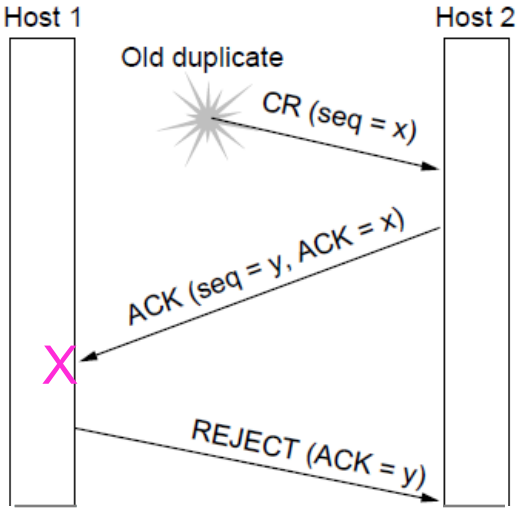
- Three-way handshake protects against odd cases:

a) Duplicate CR. Spurious ACK does not connect

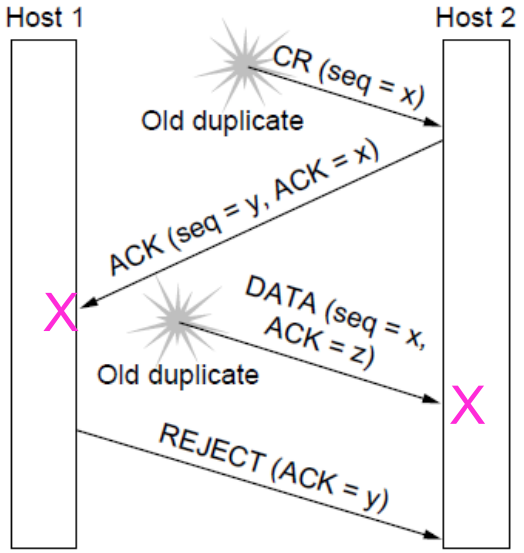
b) Duplicate CR and DATA. Same plus DATA will be rejected (wrong ACK).

- Within a connection, a timestamp is used to extend the 32-bit sequence number so that it will not wrap within the maximum packet lifetime, even for gigabit-per-second connections

a)



b)

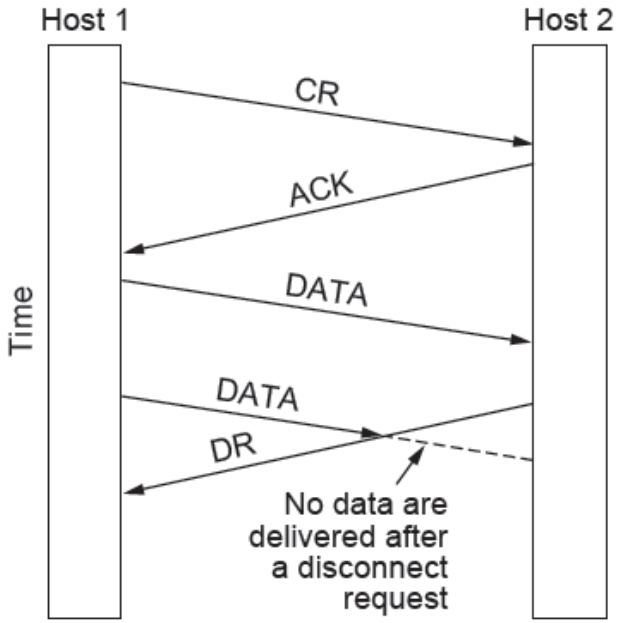


Connection Release

- Asymmetric Disconnection
  - Either party can issue a DISCONNECT, which results in DISCONNECT TPDU and transmission ends in both directions
- Symmetric Disconnection
  - Both parties issue DISCONNECT, closing only *one direction at a time* -allows flexibility to remain in receive mode

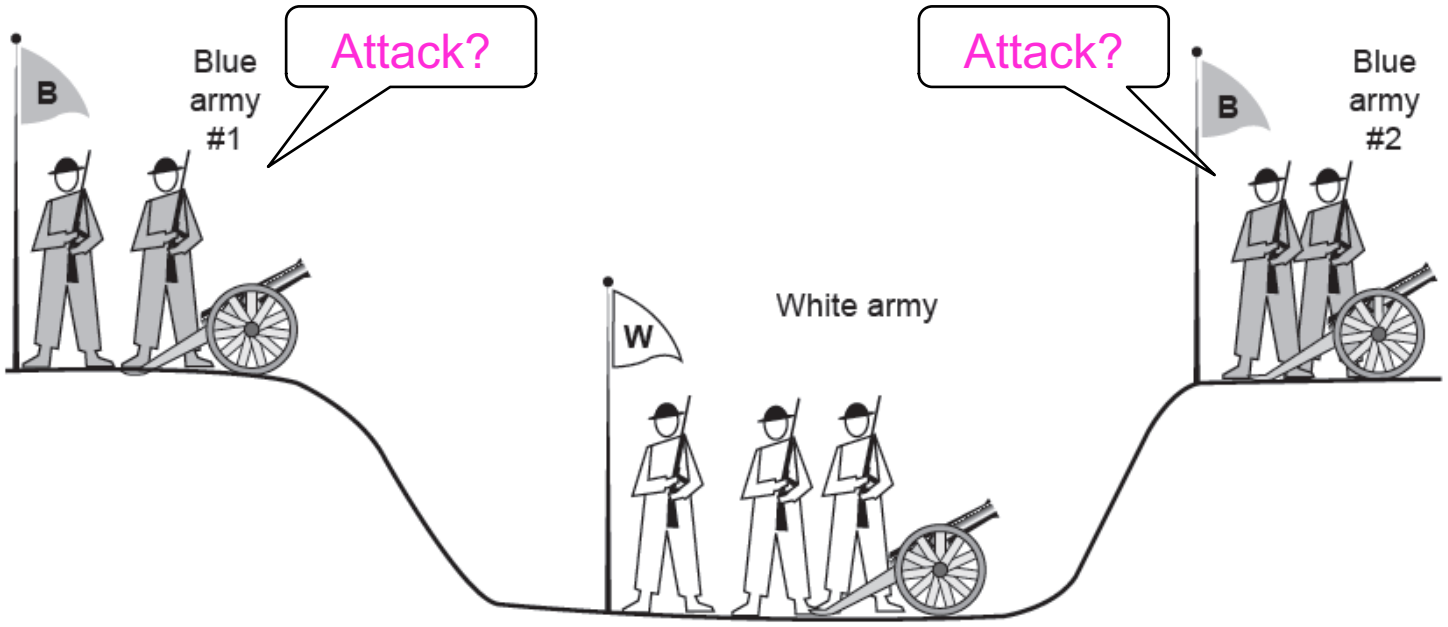
Connection Release (Cont.)

- Asymmetric vs Symmetric connection release types
- Asymmetric release may result in data loss hence symmetric release is more attractive
- Symmetric release works well where each process has a set amount of data to transmit and knows when it has been sent
- What happens in other cases?



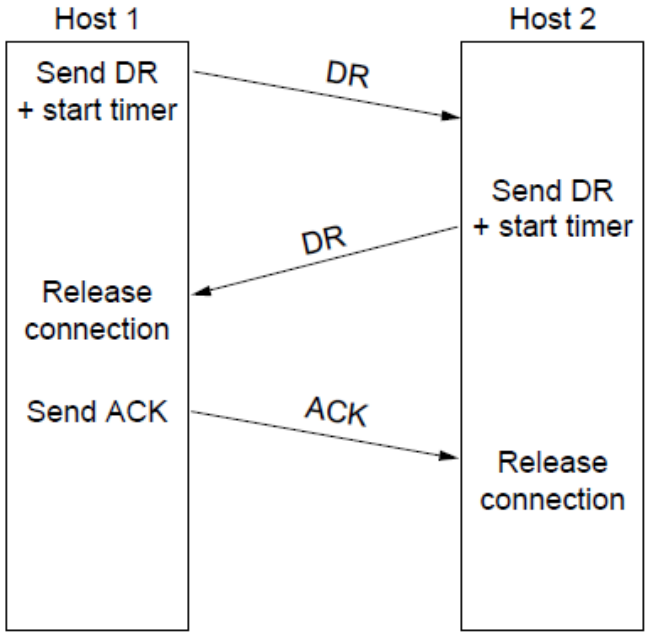
Resolving the Connection Release Problem

- How to we decide the importance of the last message? Is it essential or not?
- No protocol exists which can resolve this ambiguity- Two-army problem shows pitfall of agreement



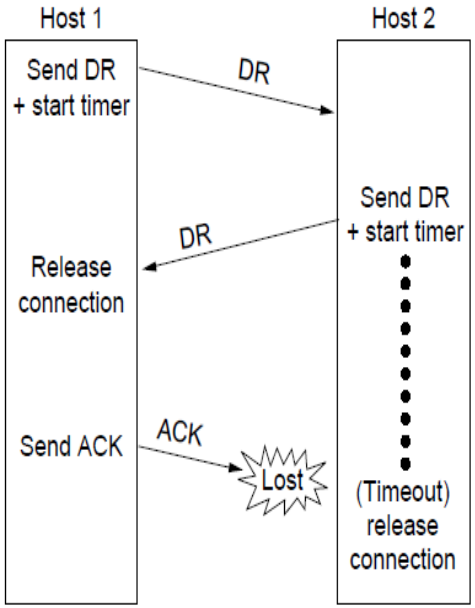
Strategies to Allow Connection Release

- 3 way handshake
- Finite retry
- *Timeouts*
- Normal release sequence, initiated by transport user on Host 1
  - DR=Disconnect Request
  - Both DRs are ACKed by the other side

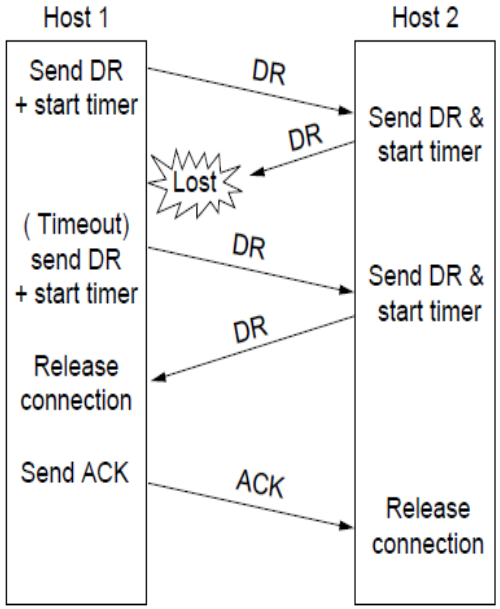


Connection Release (Error Cases)

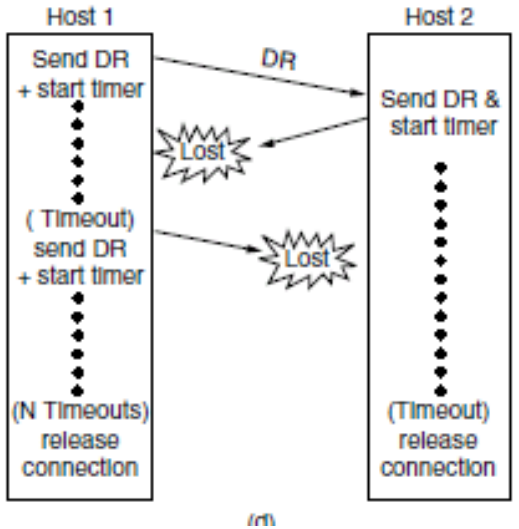
- Error cases are handled with timer and retransmission



Final ACK lost, Host 2 times out



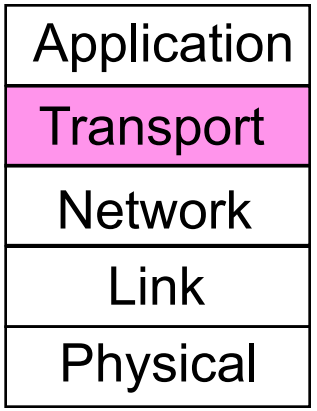
Lost DR causes retransmissions



Extreme: Many lost DRs cause both hosts to timeout

Elements of Transport Protocols

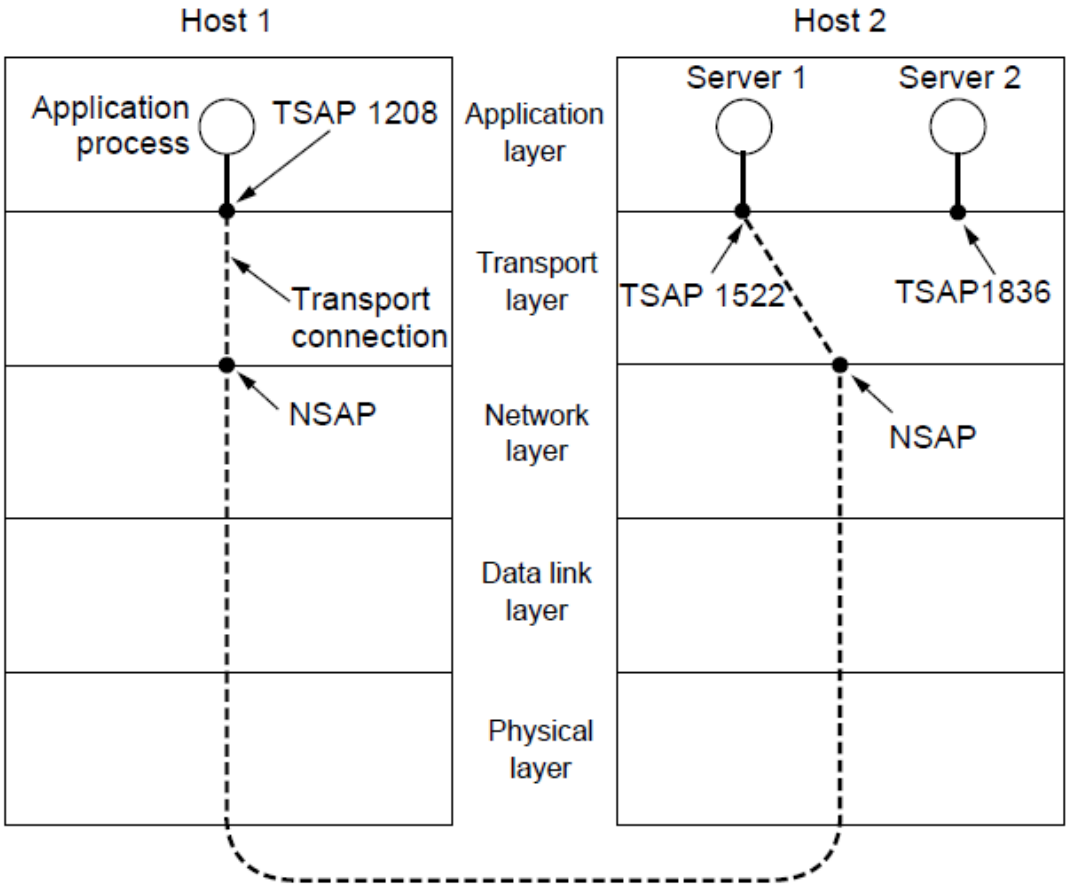
- Connection establishment
- Connection release
- Addressing



Addressing

- Specification of remote process to connect to is required at application and transport layers
- Addressing in transport layer is typically done using Transport Service Access Points (TSAPs)
  - on the internet, a TSAP is commonly referred to as a port (eg port 80)
- Addressing in the network later is typically done using Network Service Access Points (NSAPs)
  - on the internet, the concept of an NSAP is commonly interpreted as simply an IP address

TSAPs, NSAPs and Transport Layer Connections Illustrated



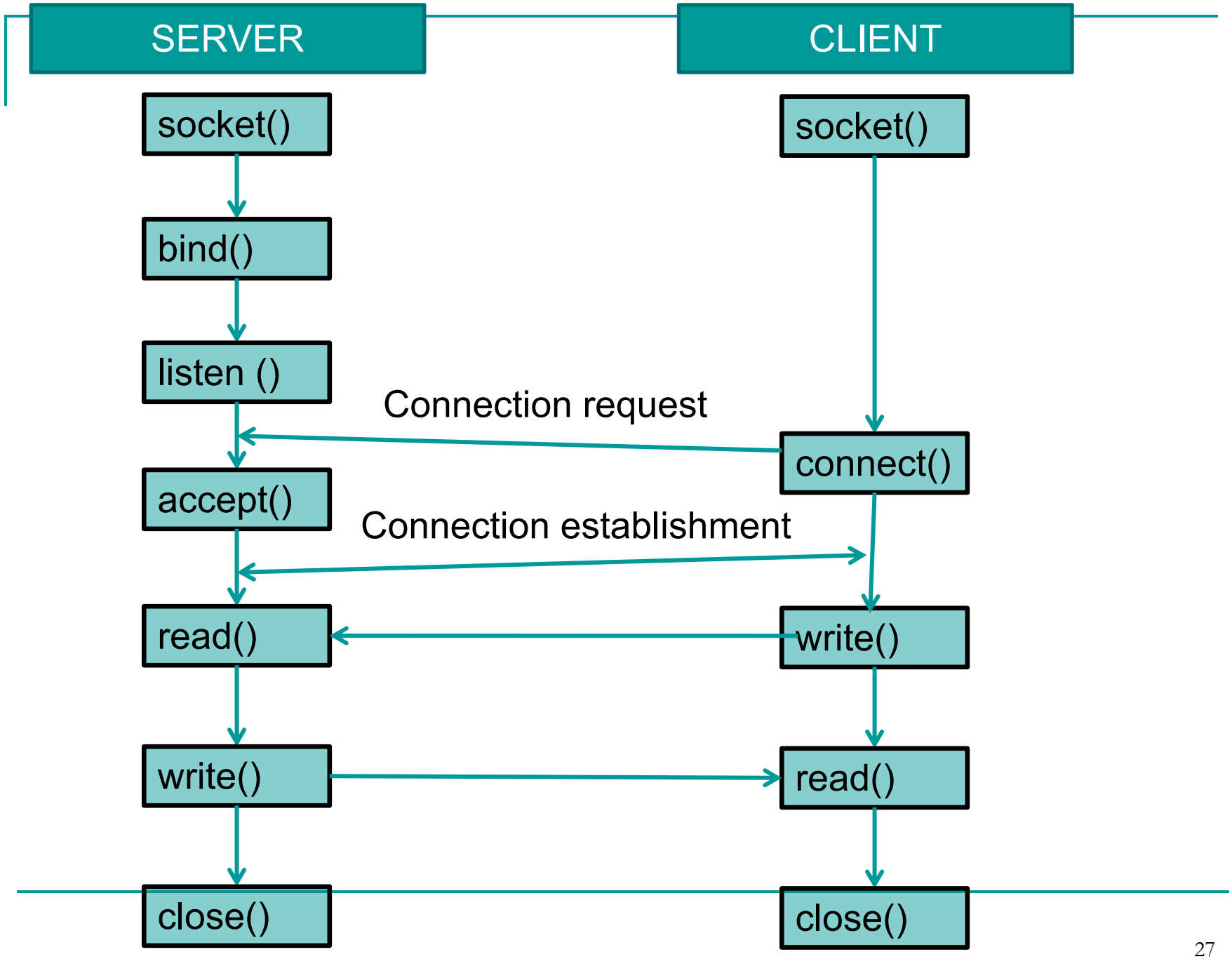
Types of TSAP Allocation

1. Static
   - Well known services have standard allocated TSAPs/ports, which are embedded in OS
   - cf. Unix /etc/services, www.iana.org
2. Directory Assistance – Port-mapper
   - A new service must register itself with the portmapper, giving both its service name and TSAP
3. Mediated
   - A process server intercepts inbound connections and spawns requested server and attaches inbound connection
   - cf. Unix /etc/(x)inetd

Sockets

- Sockets widely used for interconnections
  - “Berkeley" sockets are predominant in internet applications
  - Notion of “sockets” as transport endpoints
  - Like simple set plus SOCKET, BIND, and ACCEPT

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection


SERVER
CLIENT
socket()
socket()
bind()
listen ()
Connection request
connect()
accept()
Connection establishment
read()
write()
write()
read()
close()
close()

Socket Example – Internet File Server

Client code

```
. . .
if (argc != 3) fatal("Usage: client server-name file-name");
h = gethostbyname(argv[1]);
if (!h) fatal("gethostbyname failed");

s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s <0) fatal("socket");
memset(&channel, 0, sizeof(channel));
channel.sin_family= AF_INET;
memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
channel.sin_port= htons(SERVER_PORT);

c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
if (c < 0) fatal("connect failed");

. . .
```

Get server's IP address

Make a socket

Try to connect

socket() - parameters of the call specify the addressing format to be used, the type of service desired, and protocol

Socket Example – Internet File Server, Cont.

Client code (cont.)

. . .

```
write(s, argv[2], strlen(argv[2])+1);


while (1) {
    bytes = read(s, buf, BUF_SIZE);
    if (bytes <= 0) exit(0);
    write(1, buf, bytes);
}
}
```

Write data (equivalent to send)

Loop reading (equivalent to receive) until no more data; exit implicitly calls close

Socket Example – Internet File Server (3)

Server code

```
. . .

memset(&channel, 0, sizeof(channel));
channel.sin_family = AF_INET;
channel.sin_addr.s_addr = htonl(INADDR_ANY);
channel.sin_port = htons(SERVER_PORT);


s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s < 0) fatal("socket failed");
setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
if (b < 0) fatal("bind failed");

l = listen(s, QUEUE_SIZE);
if (l < 0) fatal("listen failed");

. . .
```

Make a socket

Assign address

Prepare for incoming connections

socket() - parameters of the call specify the addressing format to be used, the type of service desired, and protocol

Socket Example – Internet File Server Cont.

Server code

```
. . .
  while (1) {
      sa = accept(s, 0, 0);
      if (sa < 0) fatal("accept failed");

      read(sa, buf, BUF_SIZE);

      /* Get and return the file. */
      fd = open(buf, O_RDONLY);
      if (fd < 0) fatal("open failed");

      while (1) {
          bytes = read(fd, buf, BUF_SIZE);
          if (bytes <= 0) break;
          write(sa, buf, bytes);
      }
      close(fd);
      close(sa);
  }
}
```

Block waiting for the next connection

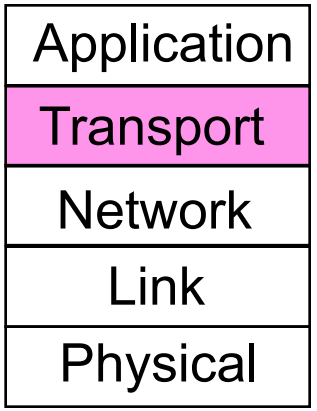
Read (receive) request and treat as file name

Write (send) all file data

Done, so close this connection

Outline

- Internet Transport Protocols
- UDP (connectionless)
- TCP (connection-oriented)

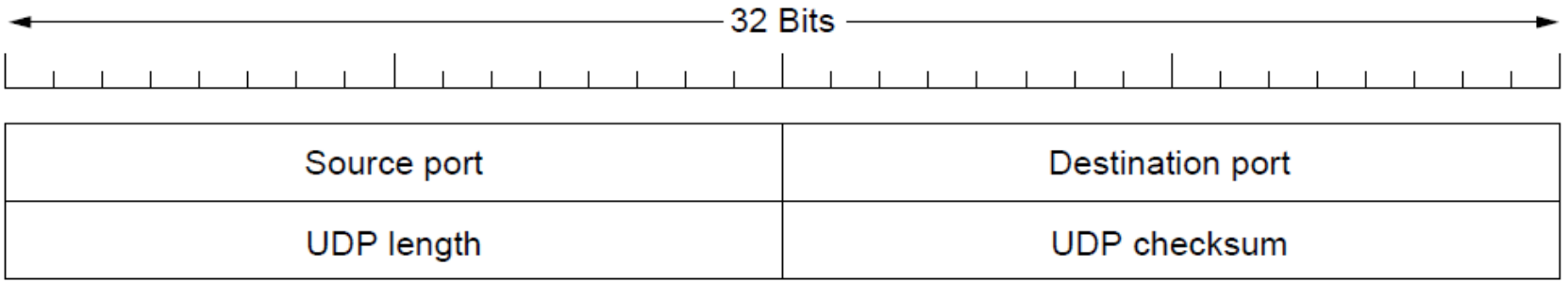


User Datagram Protocol (UDP)

- Defined in RFC 768
- Provides a protocol whereby applications can transmit encapsulated IP datagrams without a connection establishment
- UDP transmits in segments consisting of an 8-byte header followed by the payload
- UDP headers contain source and destination ports, payload is handed to the process which is attached to the particular port at destination (using BIND primitive or similar)

User Datagram Protocol (UDP)

- Main advantage of using UDP over raw IP is the ability to specify ports for source and destination pairs
- Both source and destination ports are required - destination allows initial routing for incoming segments, source allows reply routing for outgoing segments



Structure of UDP header: It has ports (TSAPs), length and checksum

Strengths and Weaknesses of UDP

- Strengths: provides an IP interface with multiplexing/de-multiplexing capabilities and consequently, transmission efficiencies
- Weaknesses: UDP does not include support for flow control, error control or retransmission of bad segments
- Conclusion: where applications require a precise level of control over packet flow/error/timing, UDP is a good choice

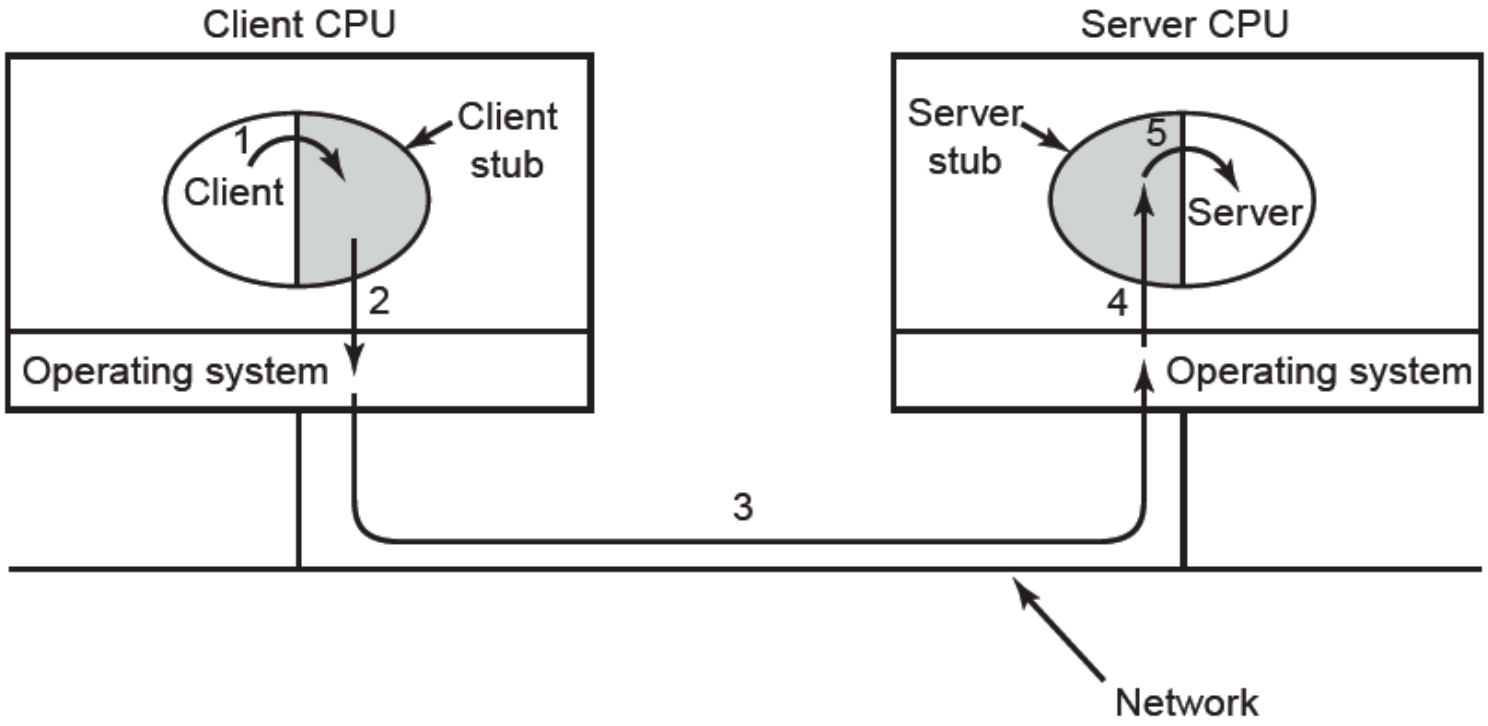
Using UDP: Remote Procedure Call (RPC)

- Sending a message and getting a reply back is analogous to making a function call in programming languages
- Birrell and Nelson (1984) modified this approach to allow local programs to call procedures on remote hosts using UDP as the transport protocol
  - Remote Procedure Call (RPC)

Using UDP: Remote Procedure Call (RPC)

- To call a remote procedure, the client is bound to a small library (the client stub) that represents the server procedure in the client‘s address space.
- Similarly the server is bound with a procedure called the server stub. These stubs hide the fact that the procedure itself is not local.
- UDP with retransmissions is a low-latency transport

RPC Illustrated



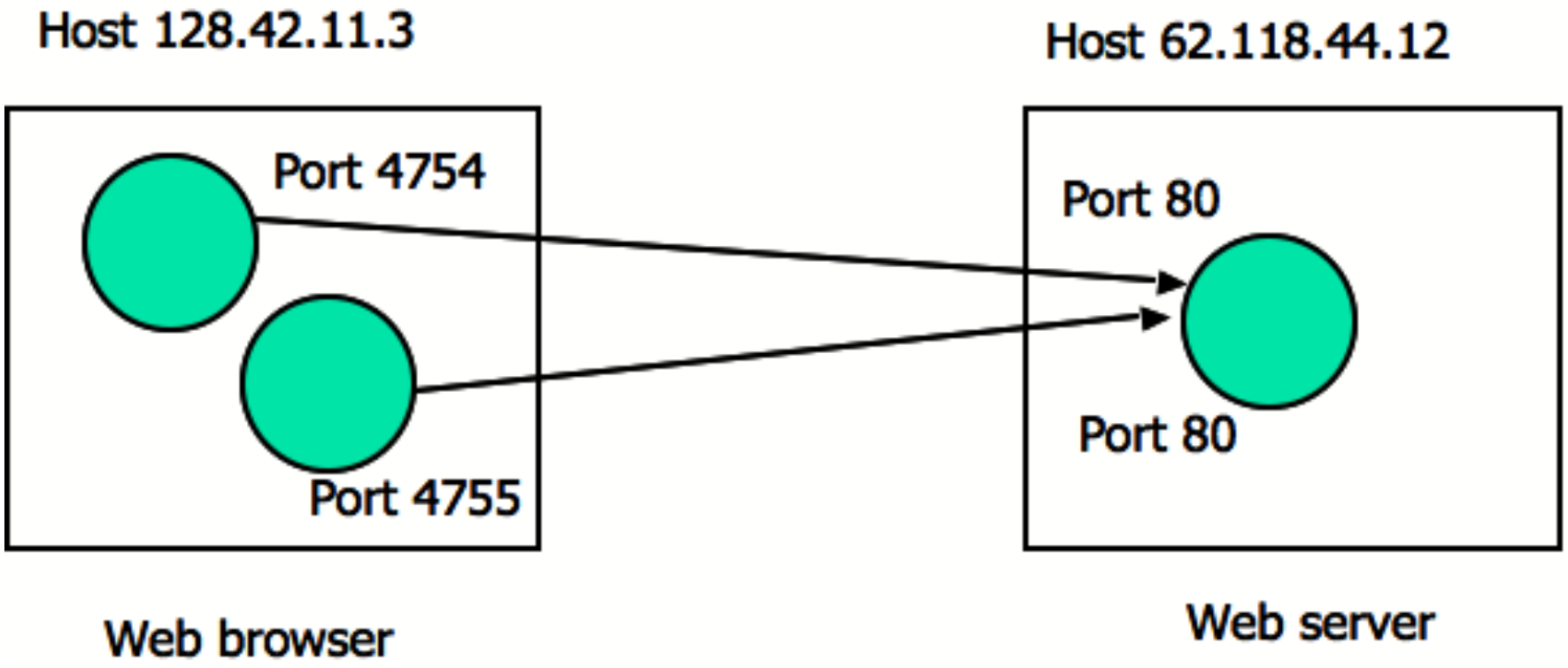
Transmission Control Protocol (TCP)

- RFC 793, 1122, 1323
- Provides a protocol by which applications can transmit IP datagrams within a connection-oriented framework, thus increasing reliability
- TCP transport entity manages TCP streams and interfaces to the IP layer - can exist in numerous locations (kernel, library, user process)
- TCP entity accepts user data streams, and segments them into pieces < 64KB (often 1460B in order so that the IP and TCP headers can fit into a single Ethernet frame), and sends each piece as a separate IP datagram
- Recipient TCP entities reconstruct the original byte streams from the encapsulation

The TCP Service Model

- Sender and receiver both create “sockets”, consisting of the IP address of the host and a port number
- For TCP Service to be activated, connections must be explicitly established between a socket at a sending host (src-host, src-port) and a socket at a receiving host (dest-host, dest-port)
- Special one-way server sockets may be used for multiple connections simultaneously

Example


Host 128.42.11.3
Host 62.118.44.12
Port 4754
Port 4755
Port 80
Port 80
Web browser
Web server

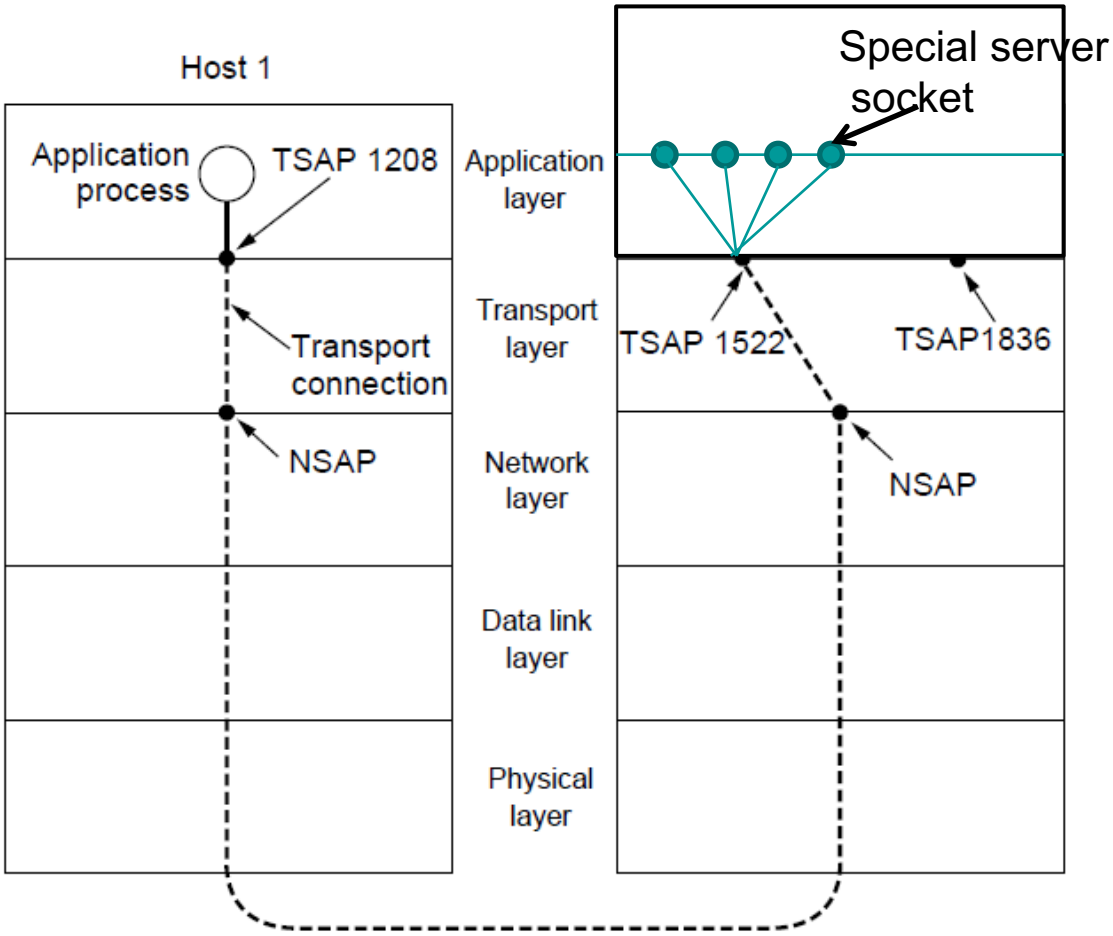
Port Allocations

- Recall TSAPs
- Port numbers can range from 0-65535
- Port numbers are regulated by IANA (http://www.iana.org/assignments/port-numbers)
- Ports are classified into 3 segments:
  - Well Known Ports (0-1023)
  - Registered Ports (1024-49151)
  - Dynamic Ports (49152-65535)

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

Socket Library - Multiplexing

- Socket library provides a multiplexing tool on top of TSAPs to allow servers to service multiple clients
- It **simulate** the server using a different port to connect back to the client



Features of TCP Connections

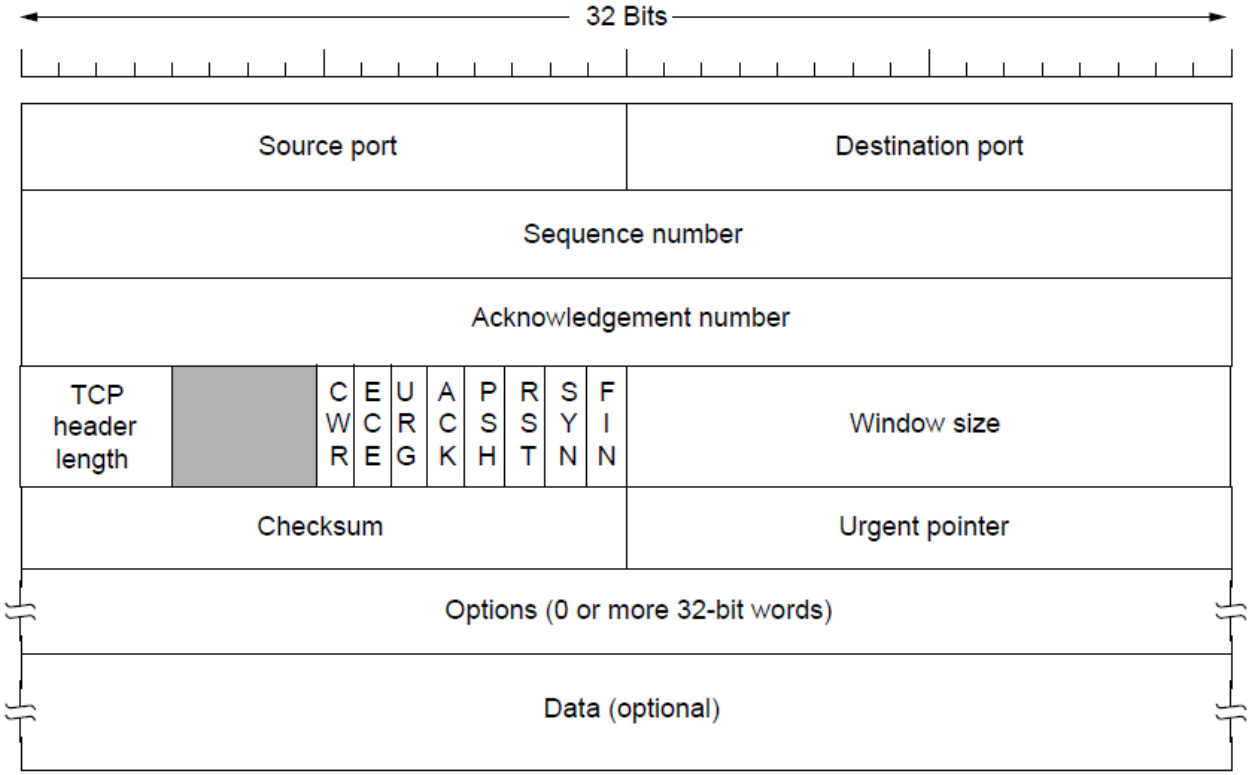
- TCP connections are:
- Full duplex - data in both directions simultaneously
- Point to point - exact pairs of senders and receivers
- Byte streams, not message streams - message boundaries are not preserved
- Buffer capable - TCP entity can choose to buffer prior to sending or not depending on the context
  - PUSH flag - indicates a transmission not to be delayed
  - URGENT flag - indicates that transmission should be sent immediately (priority above in process data)

TCP Protocol

- Data exchanged between TCP entities in segments - each segment has a fixed 20 byte header plus zero or more data bytes
- TCP entities decide how large segments should be within 2 constraints, namely:
  - 65,515 byte IP payload
  - Maximum Transfer Unit (MTU) - generally 1500 bytes
- Sliding window protocol - sender transmits and starts a timer, receiver sends back an acknowledgement which is the next sequence number expected - if sender's timer expires before acknowledgement, then the sender transmits the original segment again

The TCP Segment Header

- TCP header includes addressing (ports), sliding window (seq. / ack. number), flow control (window), error control (checksum) and more



The TCP Segment Header

- *Source port* and *Destination port* fields identify the local end points of the connection
- *Sequence number* and *Acknowledgement* number fields perform their usual functions (cumulative acknowledgement)
- *TCP header* length tells how many 32-bit words are contained in the TCP header
- *Window size* field tells how many bytes may be sent starting at the byte acknowledged
- *Checksum* is also provided for extra reliability. It checksums the header, the data
- *Options* field provides a way to add extra facilities not covered by the regular header
- *URG* is set to 1 if the *Urgent pointer* is in use. The Urgent pointer is used to indicate a byte offset from the current sequence number at which urgent data are to be found

The TCP Segment Header

- *CWR* and *ECE* are used to signal congestion when *ECN* (Explicit Congestion Notification) is used
- *ECE* is set to signal an ECN-Echo to a TCP sender to tell it to slow down when the TCP receiver gets a congestion indication from the network
- *CWR* is set to signal Congestion Window Reduced from the TCP sender to the TCP receiver so that it knows the sender has slowed down and can stop sending the ECN-Echo
- The *ACK* bit is set to 1 to indicate that the Acknowledgement number is valid. This is the case for nearly all packets; 0=ignore ACK number field
- *PSH* bit indicates PUSHed data. The receiver is hereby kindly requested to deliver the data to the application upon arrival and not buffer it until a full buffer has been received

The TCP Segment Header

- The *RST* bit is used to abruptly reset a connection that has become confused due to a host crash or some other reason. It is also used to reject an invalid segment or refuse an attempt to open a connection
- The *SYN* bit is used to establish connections. The connection request has SYN = 1 and ACK = 0 to indicate that the piggyback acknowledgement field is not in use. The connection reply does bear an acknowledgement, however, so it has SYN = 1 and ACK = 1. In essence, the SYN bit is used to denote both CONNECTION REQUEST and CONNECTION ACCEPTED, with the ACK bit used to distinguish between those two possibilities
- The *FIN* bit is used to release a connection. It specifies that the sender has no more data to transmit. However, after closing a connection, the closing process may continue to receive data indefinitely

TCP Connection Establishment and Release

- Connections established using three-way handshake
- Two simultaneous connection attempts results in only one connection (uniquely identified by end points)
- Connections released asynchronously (symmetric release)
- Timers used for lost connection releases (three army problem)

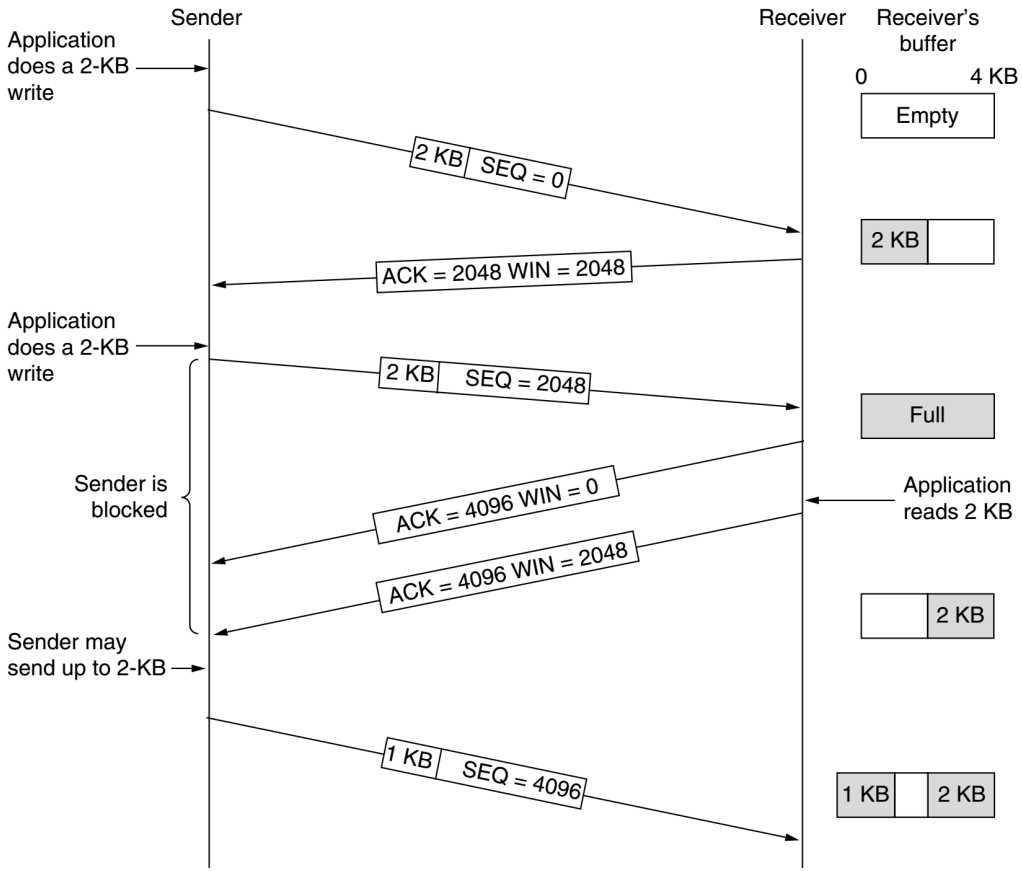
Modelling TCP Connection Management - States

- The TCP connection finite state machine has more states than our simple example from earlier.

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

TCP Transmission Policy

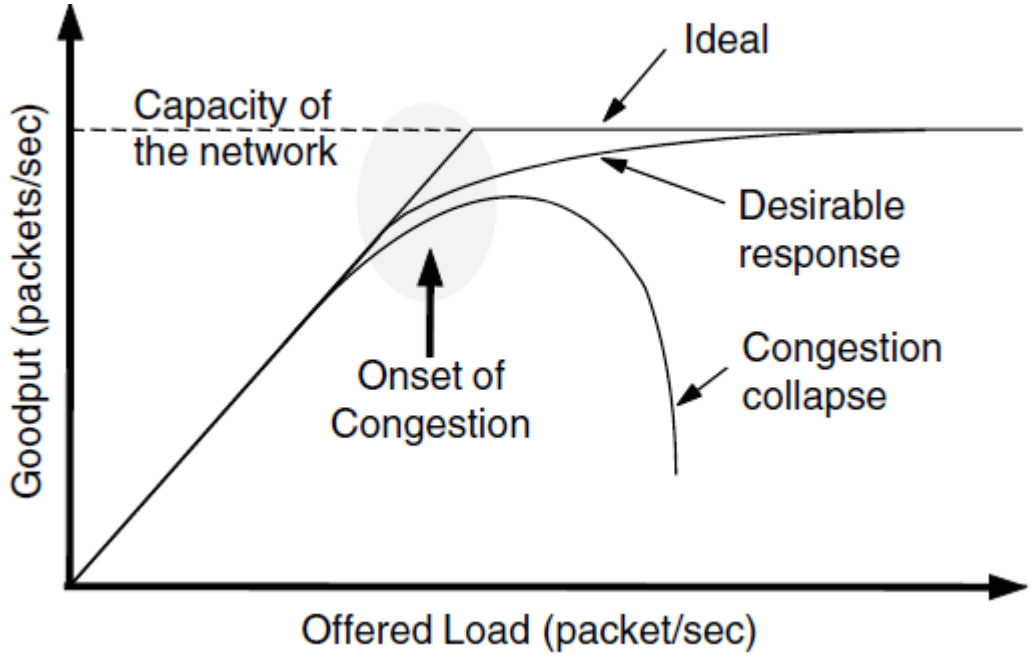
- TCP acknowledges bytes, not packets
- Receiver advertises window based on available buffer space



Congestion Control

Congestion Control

- Congestion results when too much traffic is offered; performance degrades due to loss/retransmissions
  - Goodput (=useful packets) trails offered load



Congestion Control vs Flow Control

- Flow control is an issue for point to point traffic, primarily concerned with preventing sender transmitting data faster than receiver can receive it
- Congestion control is an issue affecting the ability of the subnet to actually carry the available traffic, in a global context

Load Shedding

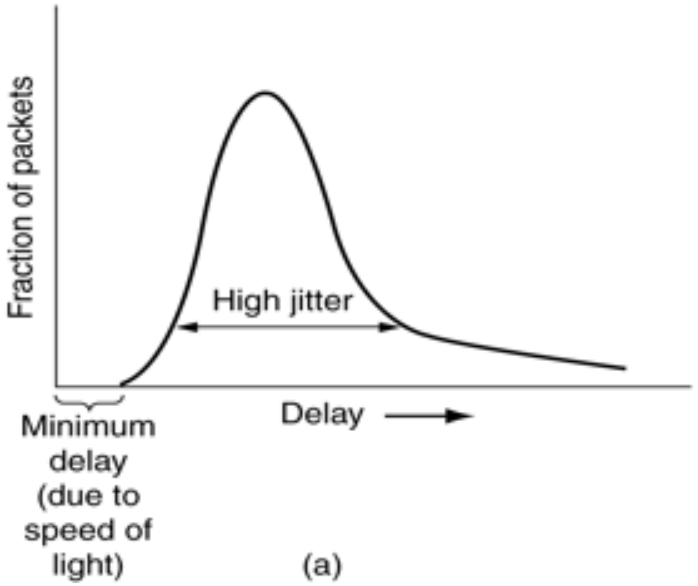
- When congestion control mechanisms fail, load shedding is the only remaining possibility - drop packets
- In order to ameliorate impact, applications can mark certain packets as priority to avoid discard policy (some applications have more stringent requirements than others)

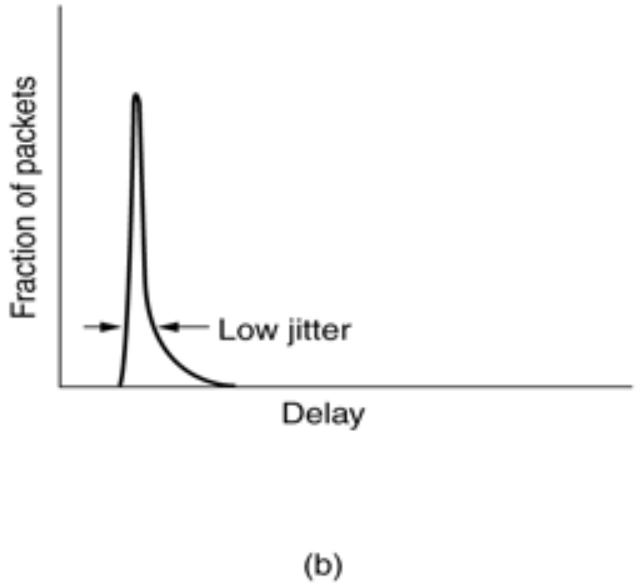
Quality of Service

- Expected network performance is an important criterion for a wide range of network applications
- Some engineering techniques are available to guarantee Quality of Service (QoS)
- 4 parameters: reliability, delay, jitter, bandwidth

Jitter Control

- Jitter is the variation in packet arrival times
  - a) high jitter
  - b) low jitter





Mechanisms for Jitter Control

- Jitter can be contained by determining the expected transit time of a packet
- Packets can be“shuffled”at each hop in order to minimise jitter - slower packets sent first, faster packets wait in a queue
- For certain applications jitter control is extremely important (eg Voice Over IP), as it directly affects the quality perceived by the application user

QOS Requirements

- Different applications care about different properties
  - We want all applications to get what they need

Application	Bandwidth	Delay	Jitter	Loss
Email	Low	Low	Low	Medium
File sharing	High	Low	Low	Medium
Web access	Medium	Medium	Low	Medium
Remote login	Low	Medium	Medium	Medium
Audio on demand	Low	Low	High	Low
Video on demand	High	Low	High	Low
Telephony	Low	High	High	Low
Videoconferencing	High	High	High	Low

"High" means a demanding requirement, e.g., low delay

Techniques for Good QoS #1

- Over-provisioning
  - more than adequate buffer, router CPU, and bandwidth (expensive and not scalable ... yet)
- Buffering
  - buffer received flows before delivery - increases delay, but smoothes out jitter, no effect in reliability or bandwidth
- Traffic Shaping
  - regulate the average rate of transmission and burstiness of transmission
  - “Buckets”
  - leaky bucket: finite internal queue (in a buffer), regulates outbound flow as well as inbound flow
  - token bucket: finite internal queue (in buffer), variable to maximum outbound flow

Techniques for Good QoS #2

- Resource Reservation
  - reserve bandwidth, buffer space, CPU in advance
- Admission Control
  - routers can decide based on traffic patterns whether to accept new flows, or reject/reroute them
- Proportional Routing
  - different traffic types for same destination split across multiple routes
- Packet Scheduling
  - fair queuing, weighted fair queueing

TCP and Congestion Control

- When networks are overloaded, congestion occurs, potentially affecting all layers
- Although lower layers (data and network) attempt to ameliorate congestion, in reality TCP impacts congestion most significantly because TCP offers methods to transparently reduce the data rate, and hence reduce congestion itself

Congestion Control: Design

- Two different problems exist
  - network capacity and receiver capacity
  - these should be dealt with separately, but compatibly
- The sender maintains two windows
  - Window described by the receiver
  - Congestion window
- Each regulates the number of bytes the sender can transmit – the maximum transmission rate is the minimum of the two windows

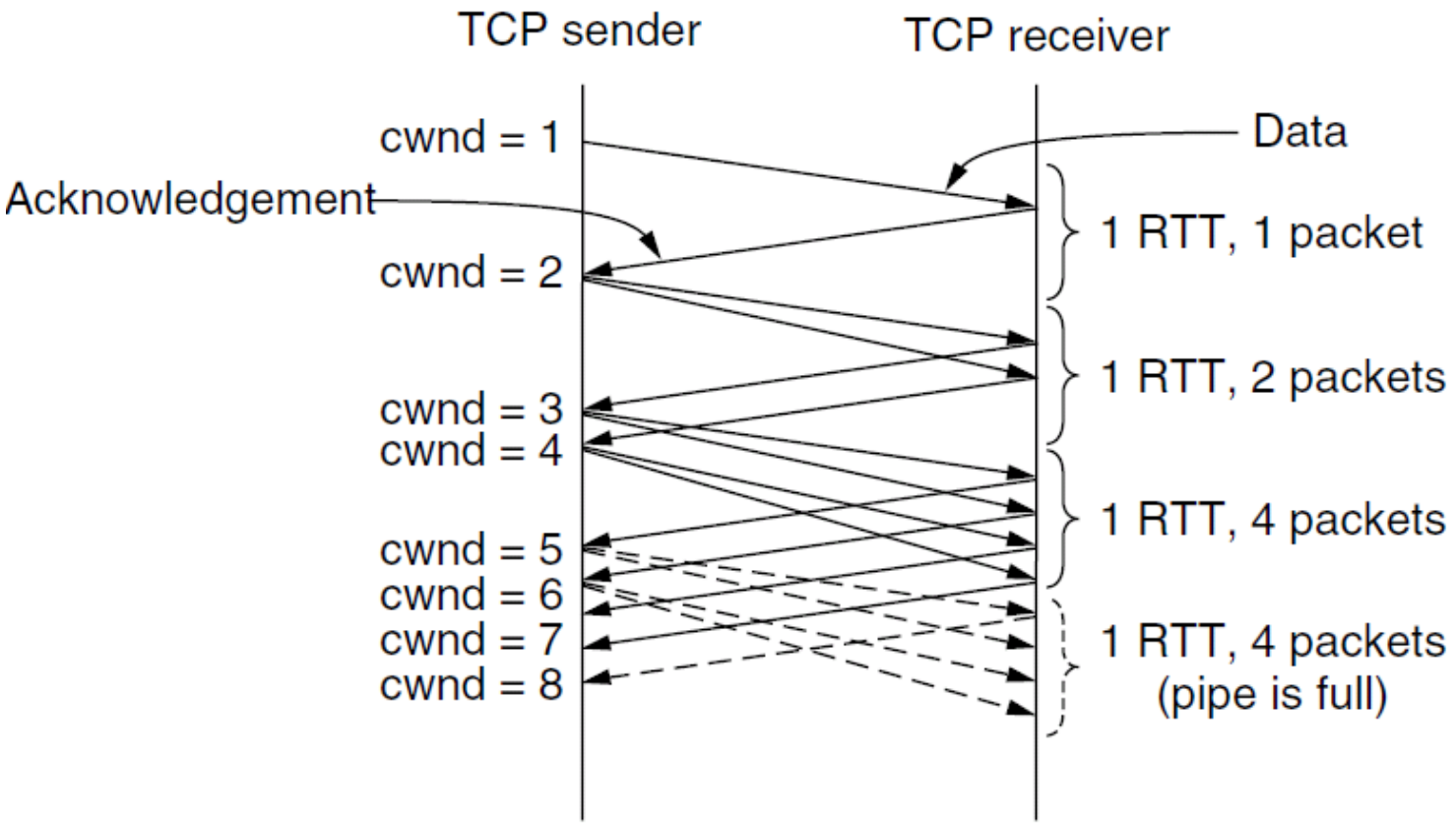
The TCP Approach to Congestion Control

- TCP adopts a defensive stance - open loop solution
  - At connection establishment, a suitable window size is chosen by the receiver based on its buffer size
  - If the sender is constrained to this size, then congestion problems will not occur due to buffer overflow at the receiver itself, but may still occur due to congestion within the network

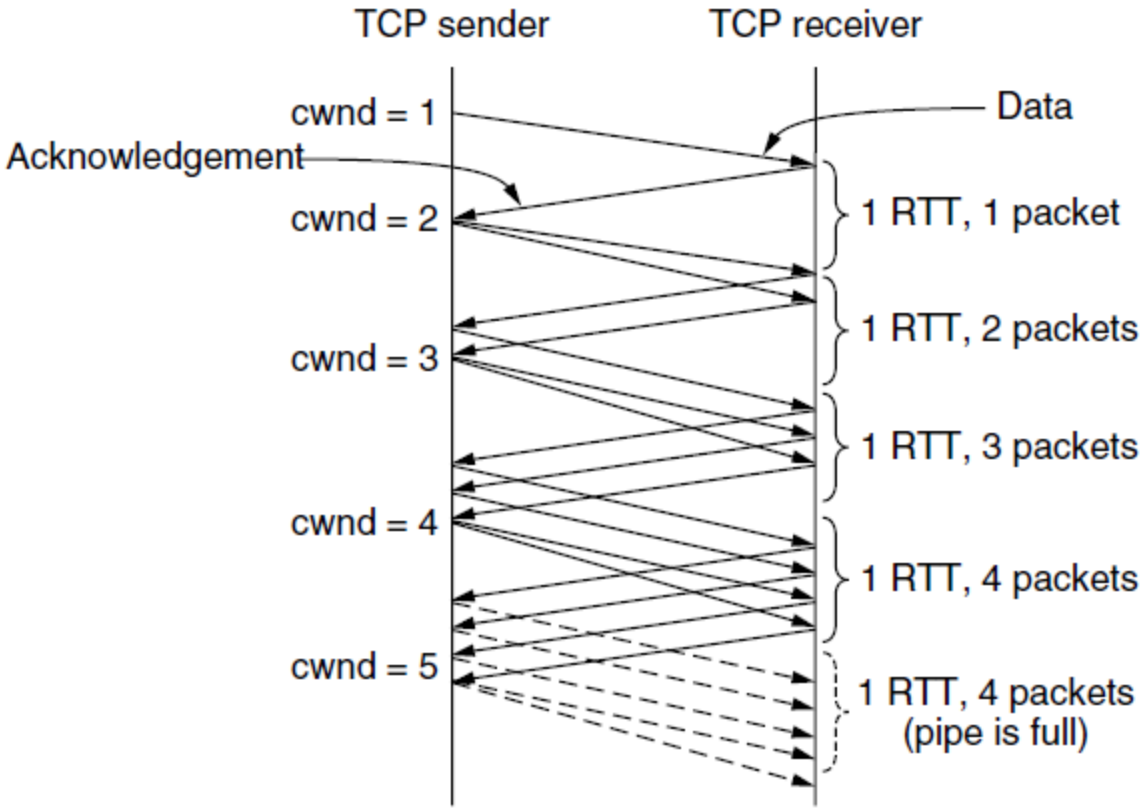
Incremental Congestion Control: Slow Start

- On connection establishment, the sender initializes the congestion window to the size of the maximum segment in use on the connection, and transmits one segment
- If this segment is acknowledged before the timer expires, the sender adds another segment's worth of bytes to the congestion window, making it two maximum size segments, and transmits two segments
- As each new segment is acknowledged, the congestion window is increased by one maximum segment size
- In effect, each set of acknowledgements doubles the congestion window - which grows until either a timeout occurs or the receiver's specified window is reached

Slow Start

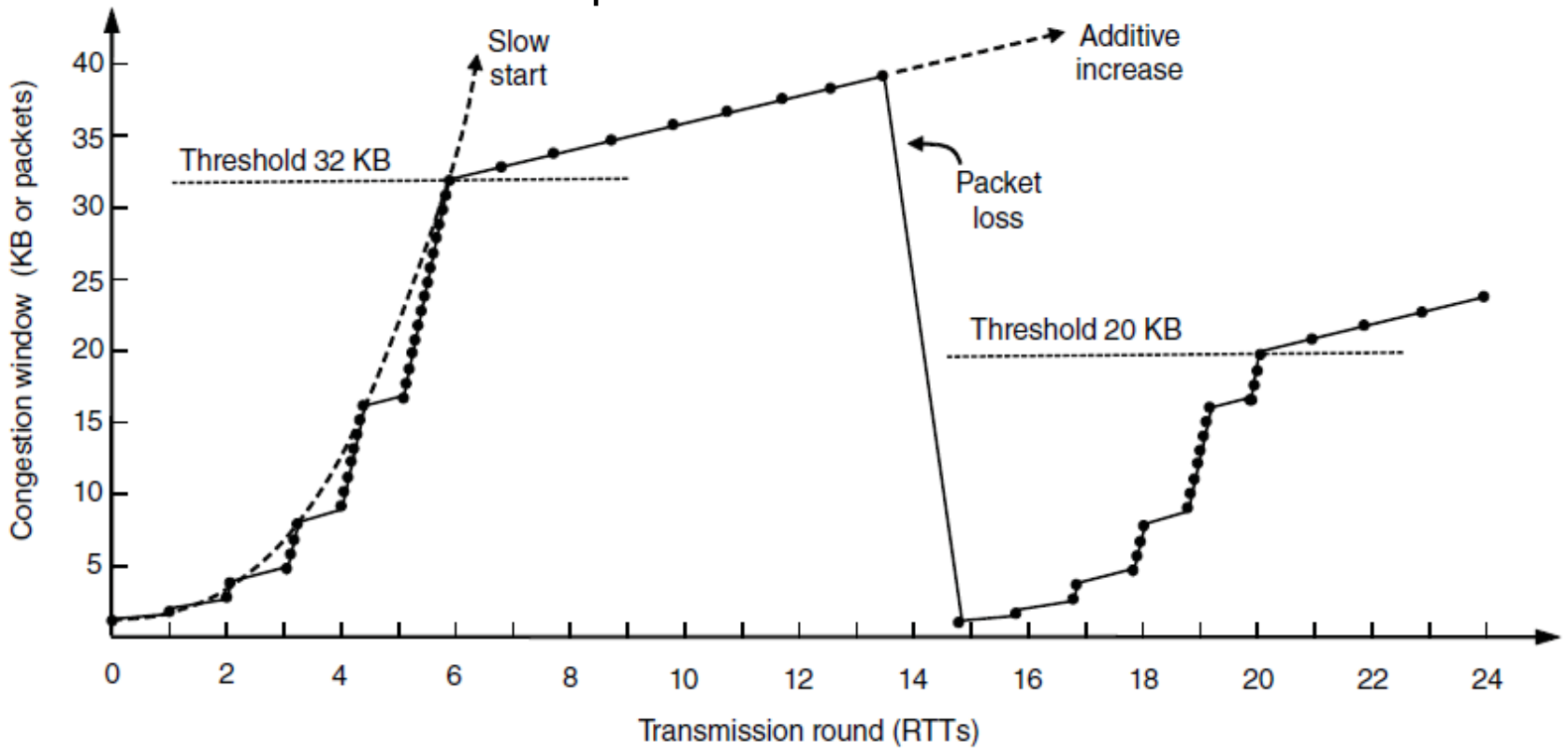

TCP sender
TCP receiver
cwnd = 1
Data
Acknowledgement
cwnd = 2
1 RTT, 1 packet
1 RTT, 2 packets
cwnd = 3
cwnd = 4
1 RTT, 4 packets
cwnd = 5
cwnd = 6
cwnd = 7
cwnd = 8
1 RTT, 4 packets
(pipe is full)

Additive increase



Internet Congestion Control Illustrated

Slow start followed by additive increase (TCP Tahoe)
Threshold is half of previous loss cwnd



General Network Policies Affecting Congestion

Layer	Policies
Transport	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy • Timeout determination
Network	• Virtual circuits versus datagram inside the subnet • Packet queueing and service policy • Packet discard policy • Routing algorithm • Packet lifetime management
Data link	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy