COMPUTER NETWORKS

Unit-IV

Transport Layer

Unit-IV (Transport Layer)

- Transport Layer
  - Transport Services
  - Elements of Transport protocols
  - Connection management
  - TCP and UDP protocols

Application

Transport

Network

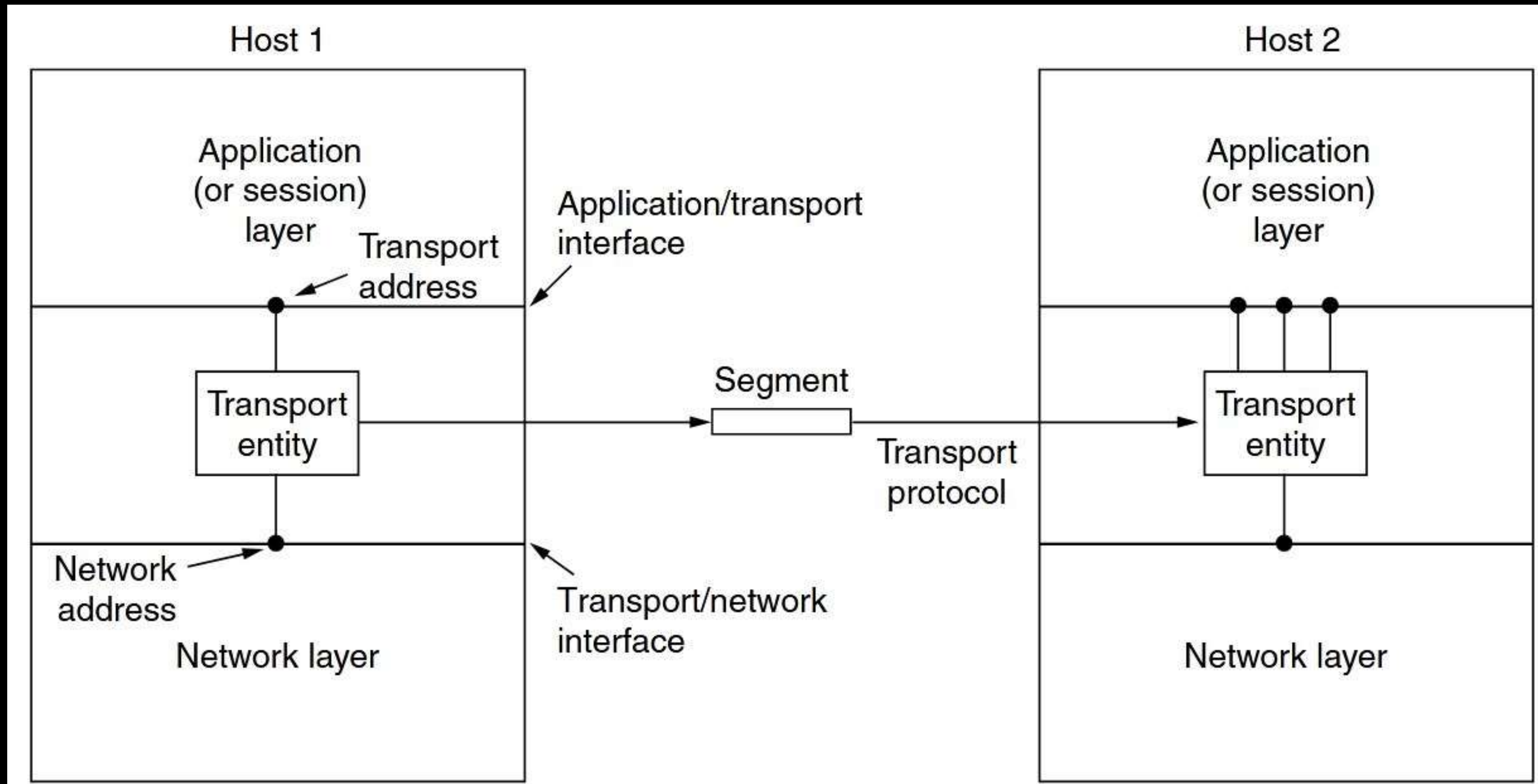
Link

Physical

Transport Layer

- Together with the network layer, the transport layer is the heart of the protocol hierarchy.
- The network layer provides end-to-end packet delivery using data-grams or virtual circuits.
- The transport layer builds on the network layer to provide data transport from a process on a source machine to a process on a destination machine with a desired level of reliability that is independent of the physical networks currently in use.

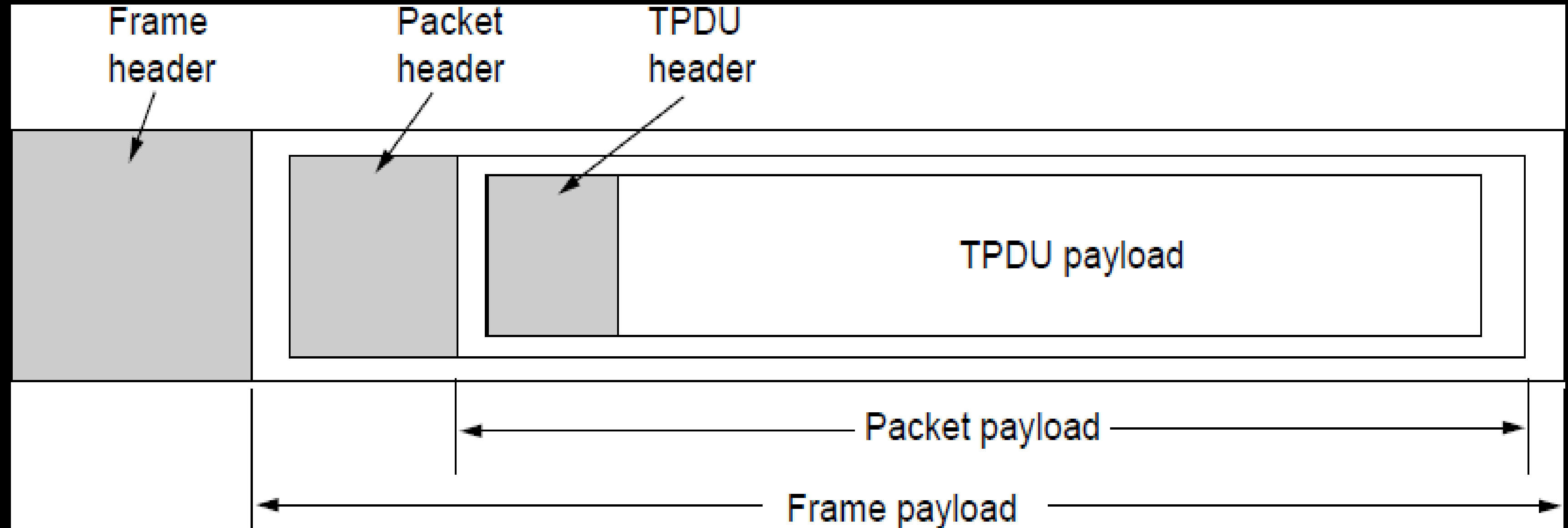
Services Provided to the Upper Layers



The Network, Transport, and Application layers

Services Provided to the Upper Layers

Transport layer sends segments in packets (in frames)



Services Provided to the Upper Layers

- The ultimate goal of the transport layer is to provide
  - efficient,
  - reliable, and
  - cost-effective data transmission service to its users, normally processes in the application layer.
- To achieve this, the transport layer makes use of the services provided by the network layer.
- The software and/or hardware within the transport layer that does the work is called the transport entity.
- The transport entity can be located in
  - the operating system kernel,
  - a library package bound into network applications,
  - a separate user process, or
  - the network interface card.

Differences between N/W and Transport Layer

- The transport code runs entirely on the users' machines, but the network layer mostly runs on the routers, which are operated by the ISP/Carrier.
- The users have no real control over the network layer, so they cannot solve the problem of poor service by using better routers or putting more error handling in the data link layer because they don't own the routers.
- The only possibility is to put on top of the network layer another layer that improves the quality of the service.

- In case of a connectionless network, packets are lost or mangled, the transport entity can detect the problem and compensate for it by using retransmissions.
- In a connection-oriented network, a transport entity is informed halfway through a long transmission that its network connection has been abruptly terminated, with no indication of what has happened to the data currently in transit, it can set up a new network connection to the remote transport entity.
- The bottom four layers can be seen as the transport service provider.
- The upper layer(s) are the transport service user.
- The transport layer is in a key position, since it forms the major boundary between the provider and user of the reliable data transmission service.
- It is the level that applications see.

Transport Service Primitives

- To allow users to access the transport service, the transport layer must provide some operations to application programs, that is, a transport service interface.
- The connection-oriented transport service is reliable.
- Of course, real networks are not error-free, but that is precisely the purpose of the transport layer to provide a reliable service on top of an unreliable network has its own interface.

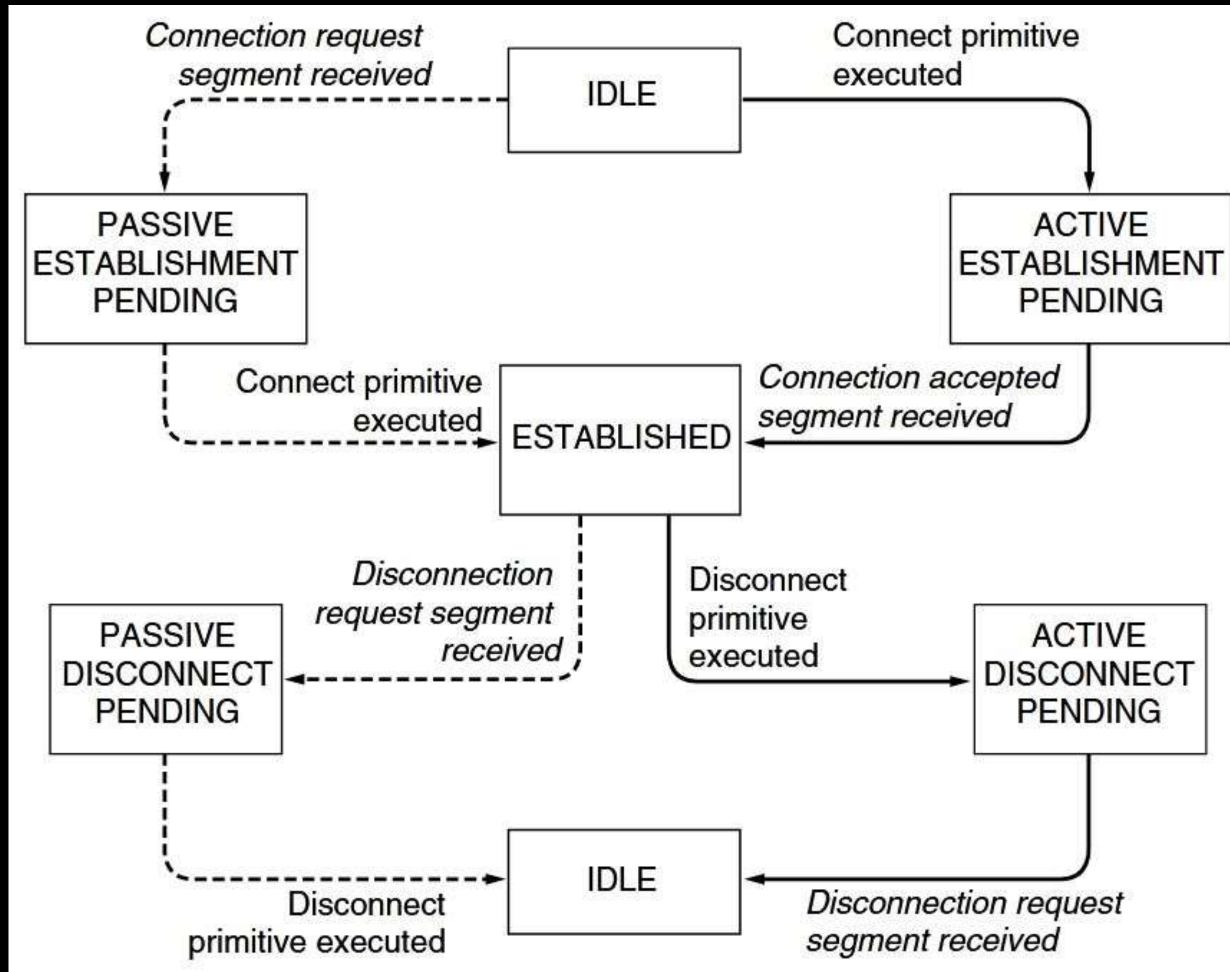
Transport Service Primitives

- Primitives that applications might call to transport data for a simple connection-oriented service:

- Client calls
  - CONNECT, SEND, RECEIVE, DISCONNECT
- Server calls
  - LISTEN, RECEIVE, SEND, DISCONNECT

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 Service Primitives



A 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.

Berkeley Sockets

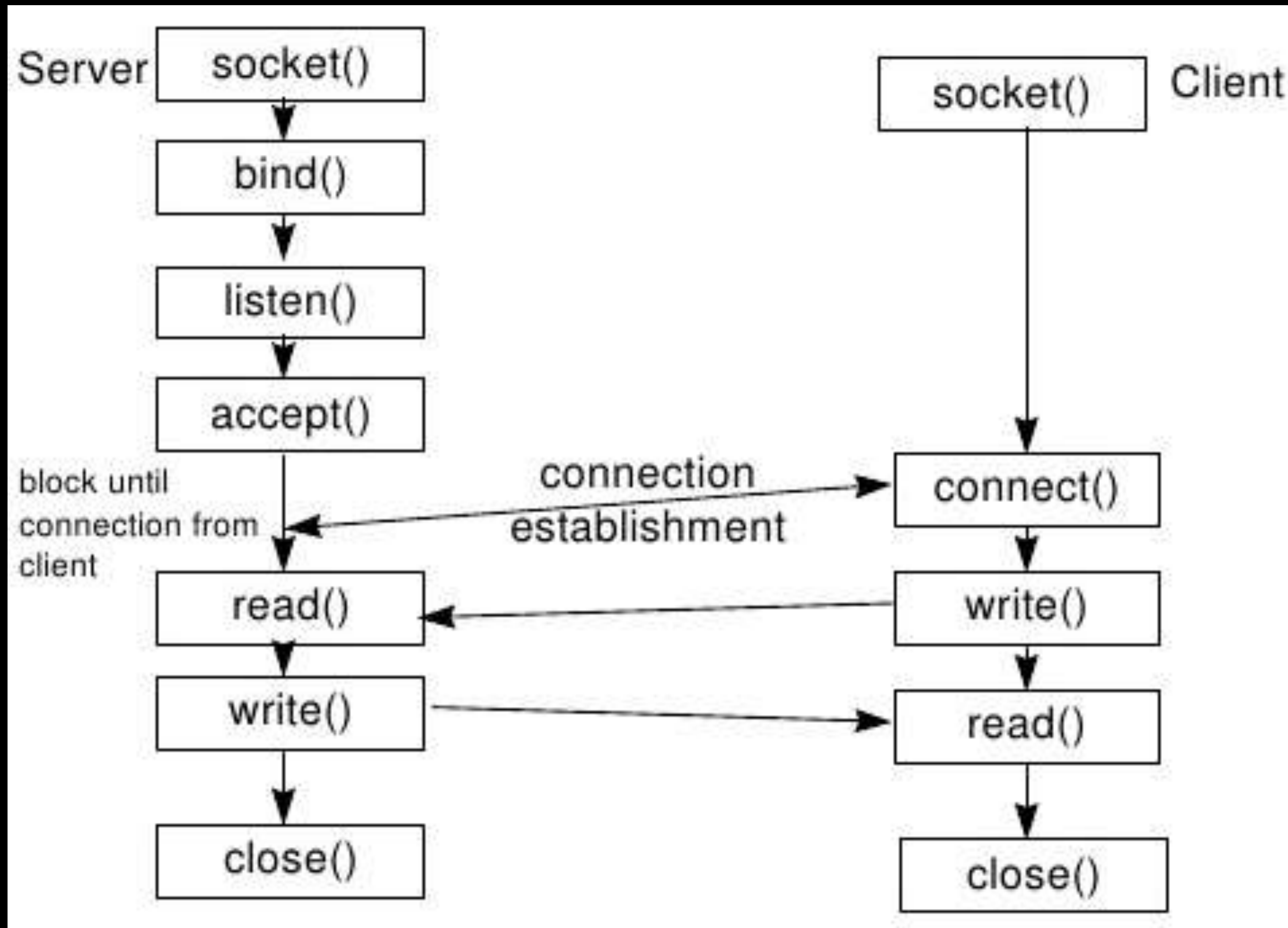
- Berkeley socket primitives are used widely for Internet programming on many operating.
- These sockets were first released as part of the Berkeley UNIX4.2 BSD software distribution in 1983.

Berkeley Socket Primitives

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

The socket primitives for TCP

Flowchart for TCP Client-Server



An Example of Socket Programming: Client-side C/C++ program

```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#define PORT 8080

int main(int argc, char const *argv[])
{
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char *hello = "Hello from client";
    char buffer[1024] = {0};
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form
    if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0)
    {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }
      if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) <
0)
    {
        printf("\nConnection Failed \n");
        return -1;
    }
    send(sock , hello , strlen(hello) , 0 );
    printf("Hello message sent\n");
    valread = read( sock , buffer, 1024);
    printf("%s\n",buffer );
    return 0;
}
```

An Example of Socket Programming: Server side C/C++ program

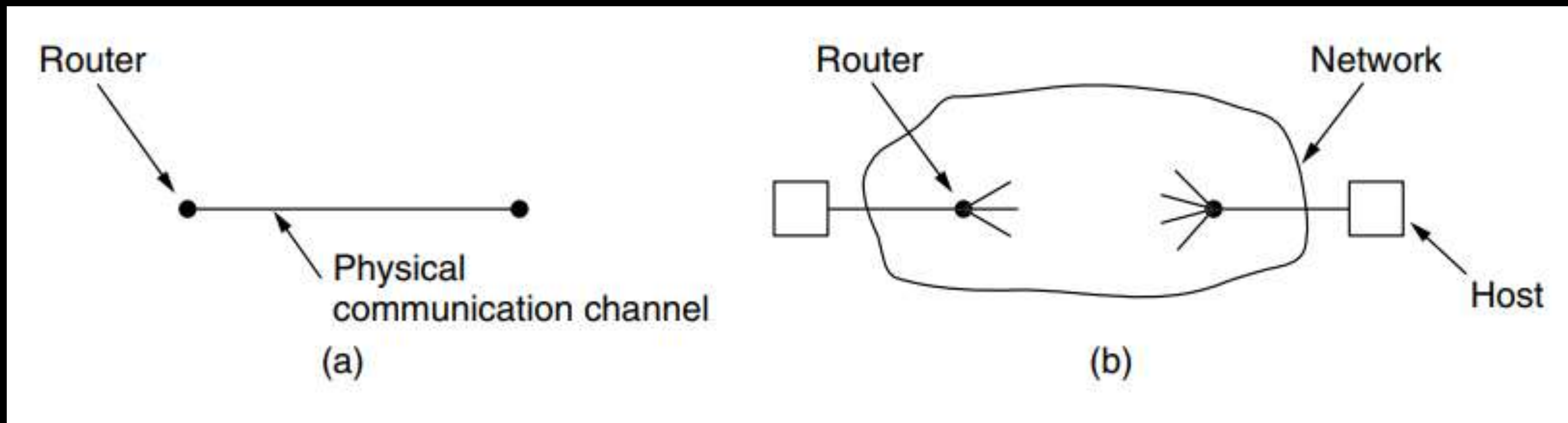
```
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#define PORT 8080
int main(int argc, char const *argv[])
{
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    char *hello = "Hello from server";

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
    {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Forcefully attaching socket to the port 8080
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR |
SO_REUSEPORT, &opt, sizeof(opt)))
    {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons( PORT );

// Forcefully attaching socket to the port 8080

    if (bind(server_fd, (struct sockaddr
*)&address, sizeof(address))<0)
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    if (listen(server_fd, 3) < 0)
    {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    if ((new_socket = accept(server_fd, (struct sockaddr
*)&address, (socklen_t*)&addrlen))<0)
    {
        perror("accept");
        exit(EXIT_FAILURE);
    }

    valread = read( new_socket , buffer, 1024);
    printf("%s\n",buffer );
    send(new_socket , hello , strlen(hello) , 0 );
    printf("Hello message sent\n");
    return 0;
}
```

Elements of Transport protocols

- The transport service is implemented by a transport protocol used between the two transport entities.
- In some ways, transport protocols resemble the data link protocols.
- Both have to deal with error control, sequencing, and flow control, among other issues.

Elements of Transport protocols

- At the data link layer, two routers communicate directly via a physical channel, whether wired or wireless, whereas at the transport layer, this physical channel is replaced by the entire network.



(a) Environment of the data link layer. (b) Environment of the transport layer.

Elements of Transport Protocols

Feature	Datalink Layer	Transport Layer
1.	Over point-to-point links such as wires or optical fiber, receiver address is not required.	Explicit addressing of destinations is required because multiple applications are run on a single machine.
2.	Establishing a connection over the wire is simple because the other end is physically available.	Initial connection establishment is complicated.
3.	The point-to-point nature of links has lesser delays and no complicated effects on data transmissions.	The network's ability to delay and duplicate packets can sometimes be disastrous and require special protocols to correctly transport information.
4.	A fixed number of buffers to outgoing line is available when a frame arrives.	A larger number of connections needs to be managed due to variations in the bandwidth.

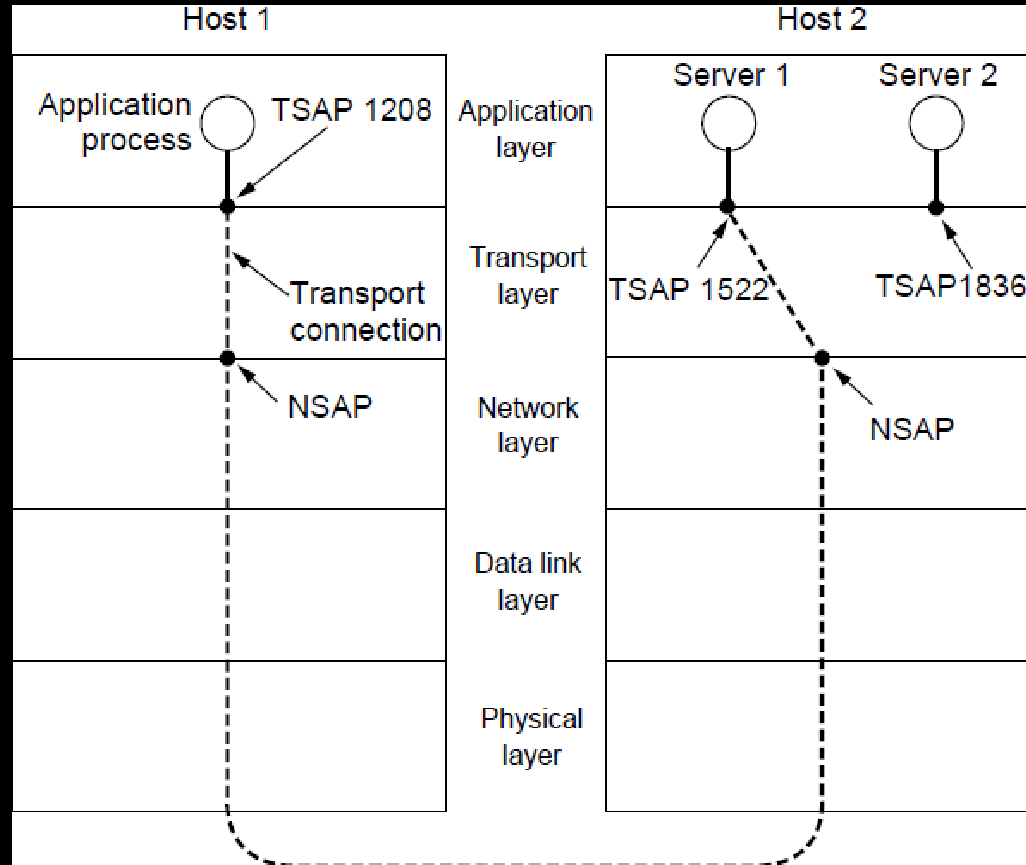
Elements of Transport Protocols

- Addressing
- Connection establishment
- Connection release
- Error control and flow control
- Multiplexing
- Crash recovery

Addressing

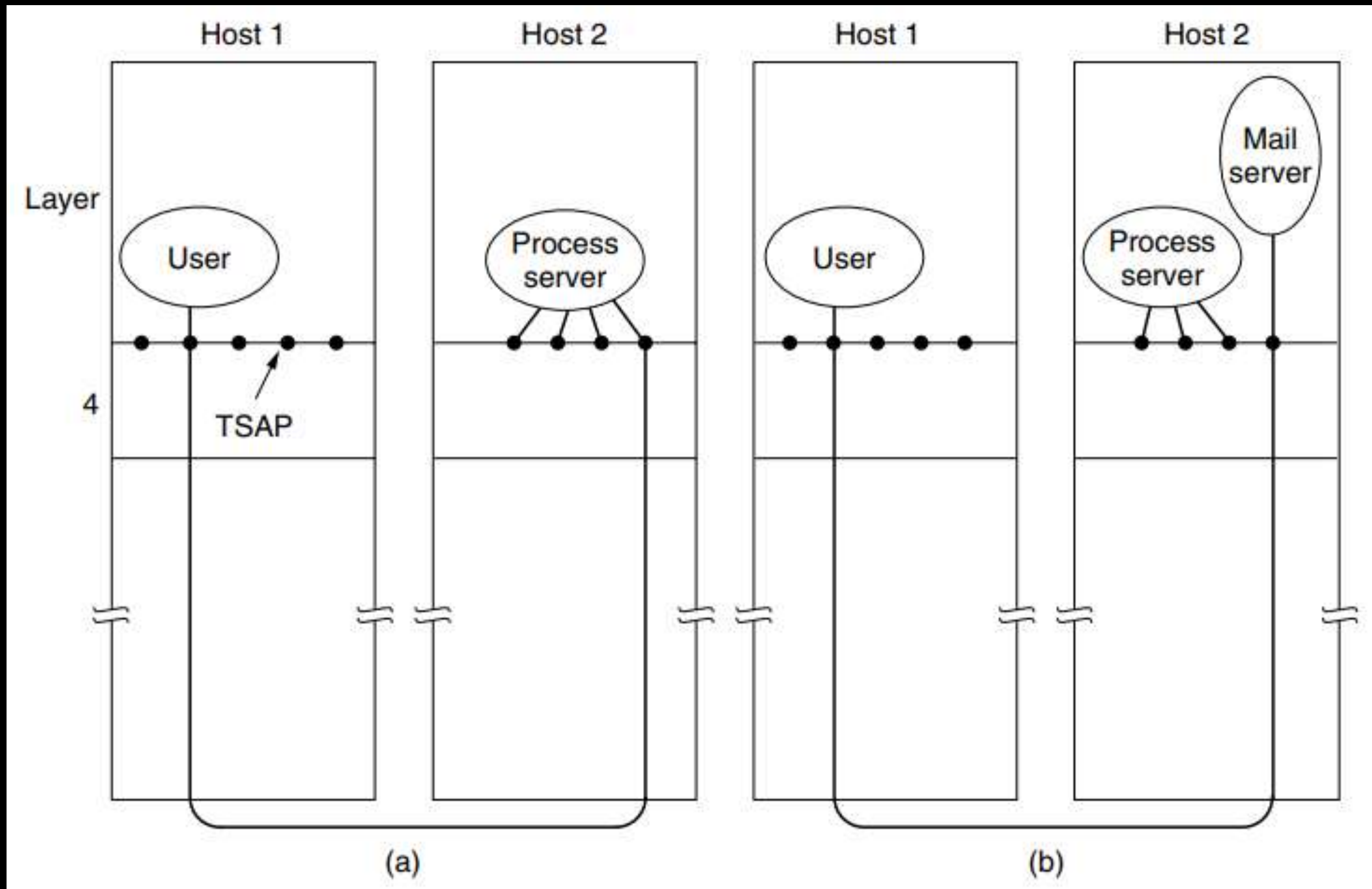
- When an application (e.g., a user) process wishes to set up a connection to a remote application process, it must specify which one to.
- The method normally used is to define transport addresses to which processes can listen for connection requests.
- In the Internet, these endpoints are called ports. We will use the generic term TSAP(Transport Service Access Point)to mean a specific endpoint in the transport layer

Addressing



- Transport layer adds TSAPs
- Multiple clients and servers can run on a host with a single network (IP) address
- TSAPs are ports for TCP/UDP

Addressing: Example of Mail server



- Transport layer adds TSAPs
- A user process in host 1 establishes a connection with a mail server in host 2 via a process server.

Addressing: Steps taken for connection setup for mail access

- A Process server (or super server) listens to a set of ports at the same time, waiting for a connection request.
- Potential users of a service begin by doing a CONNECT request, specifying the TSAP address of the service they want.
- If no server is waiting for them, they get a connection to the process server.
- The process server spawns the requested server, allowing it to inherit the existing connection with the user.
- The new server does the requested work, while the process server goes back to listening for new requests

Addressing

- Three types of Ports
  - Well-known Ports: ports 0 – 1023
  - Registered Ports: ports 1024 – 49151
  - Dynamic Ports: Ports 49152 – 65535
- Ports 0 through 49151 are formally registered worldwide through IANA.
- Other ports are not registered thru IANA but rather use a locally scoped port ID system

Connection Establishment

- Before Communication can happen between two end processes on two different machines, connection needs to be established.
- Unreliability of network needs to be handled.
- Key problem is to ensure reliability even though packets may be lost, corrupted, delayed, and duplicated.

Connection Establishment

- Packet lifetime can be restricted to a known maximum using one (or more) of the following techniques:
  1. Restricted network design.
  2. Putting a hop counter in each packet.
  3. Time-stamping each packet.

Connection Establishment

- Unreliability of network due to delayed and duplicated packets needs to be handled.
- Should not treat an old or duplicate packet as new (Use ARQ and checksums for loss/corruption)
- Solutions and Approach:
  - Don’t reuse sequence numbers within twice the MSL (Maximum Segment Lifetime) of 2T=240 secs
  - Three-way handshake for establishing connection

Connection Establishment: Maximum Segment Lifetime

- With packet lifetimes bounded, it is possible to devise a practical and foolproof way to reject delayed duplicate segments.
- The heart of the method is for the source to label segments with sequence numbers that will not be reused within T secs. The period, T, and the rate of packets per second determine the size of the sequence numbers.

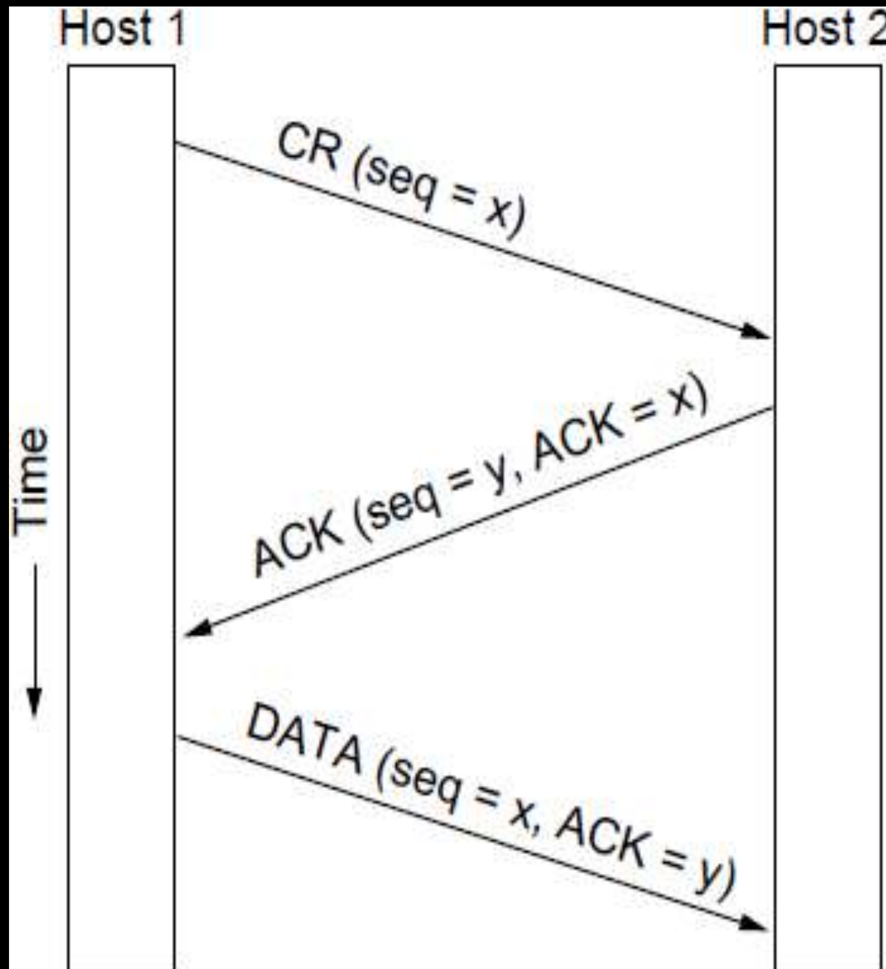
Connection Establishment: Maximum Segment Lifetime

- The clocks at different hosts need not be synchronized.
- The clock is assumed to continue running even if the host goes down.
- The clock-based method solves the problem of not being able to distinguish delayed duplicate segments from new segments.

Connection Establishment: Maximum Segment Lifetime

- Since we do not normally remember sequence numbers across connections at the destination, we still have no way of knowing if a CONNECTION REQUEST segment containing an initial sequence number is a duplicate of a recent connection.
- This snag does not exist during a connection because the sliding window protocol does remember the current sequence number
- To solve this problem the three-way handshake was introduced.
- This establishment protocol involves one peer checking with the other that the connection request is indeed current.

Connection Establishment: 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

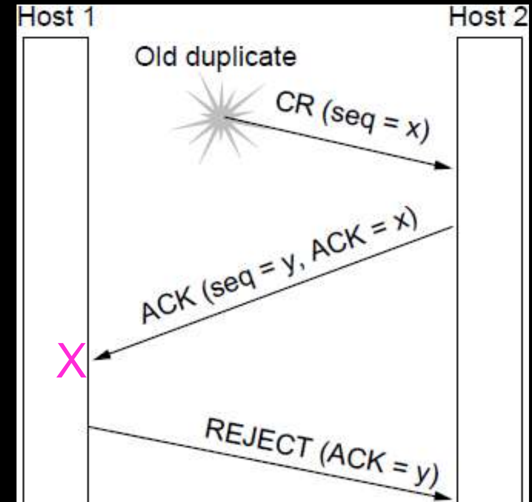
Connection Establishment: Three-way handshake

Three-way handshake protects against odd cases:

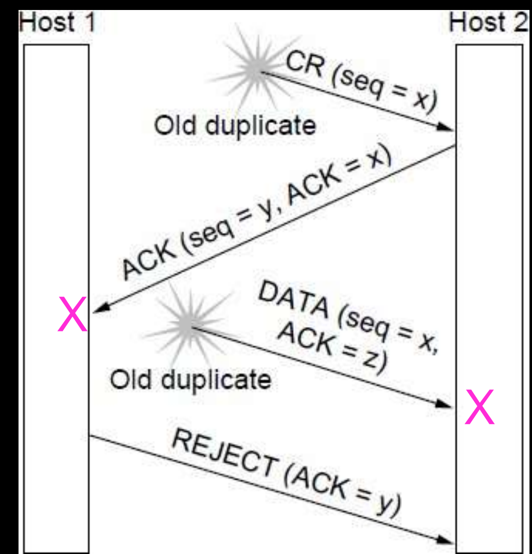
a) Duplicate CR. Spurious ACK does not connect

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

a)



b)

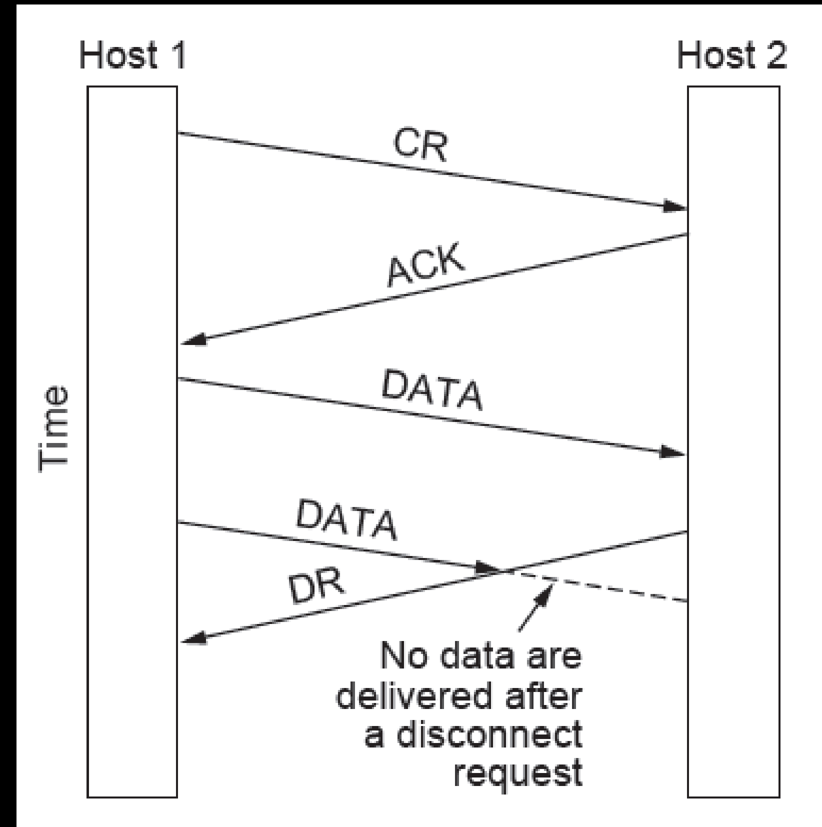


Connection Release

- There are two styles of terminating a connection
  - Asymmetric release and Symmetric release.
- Asymmetric release is the way the telephone system works: when one party hangs up, the connection is broken.
  - Asymmetric release is abrupt and may result in data loss.
- Symmetric release treats the connection as two separate unidirectional connections and requires each one to be released separately

Connection Release: Asymmetric

- Key problem is to ensure reliability while releasing (no loss of data)
- Soultion1: Asymmetric release (when one side breaks connection) is abrupt and may lose data



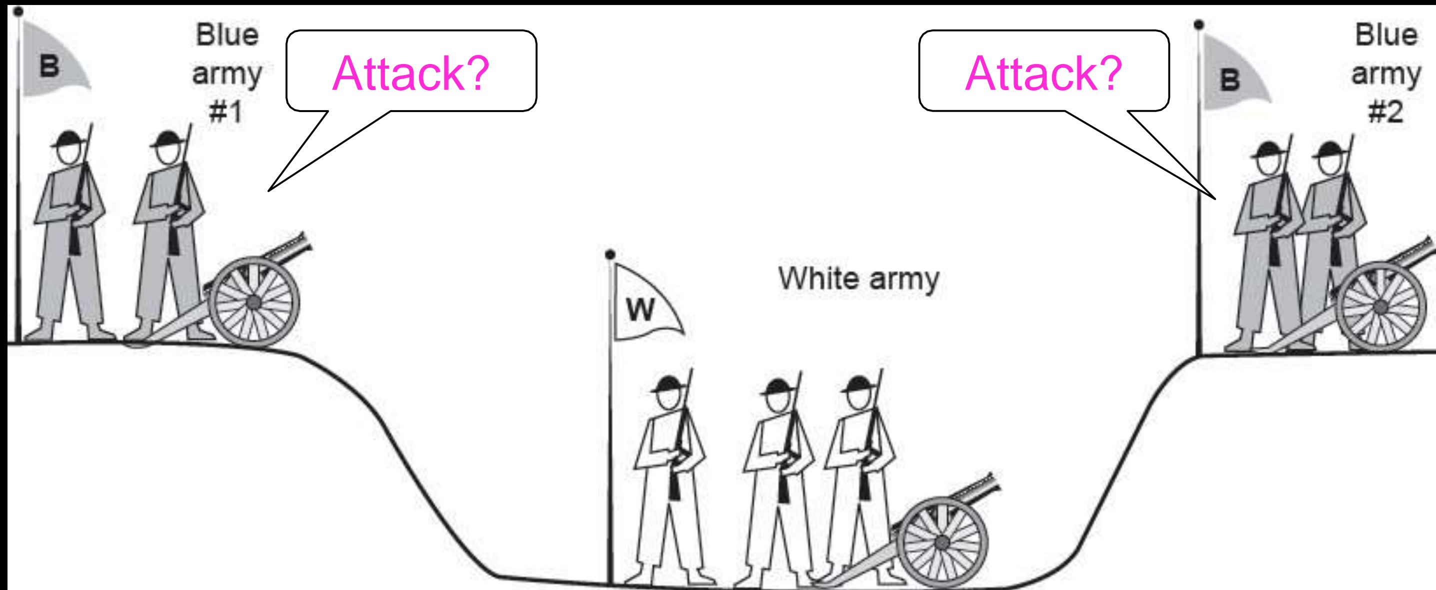
X

Connection Release: Symmetric

- A more sophisticated release protocol is needed to avoid data loss.
- One way is to use symmetric release, in which each direction is released independently of the other one.
- Here, a host can continue to receive data even after it has sent a DISCONNECT segment.
- Unfortunately, this protocol does not always work.

Connection Release

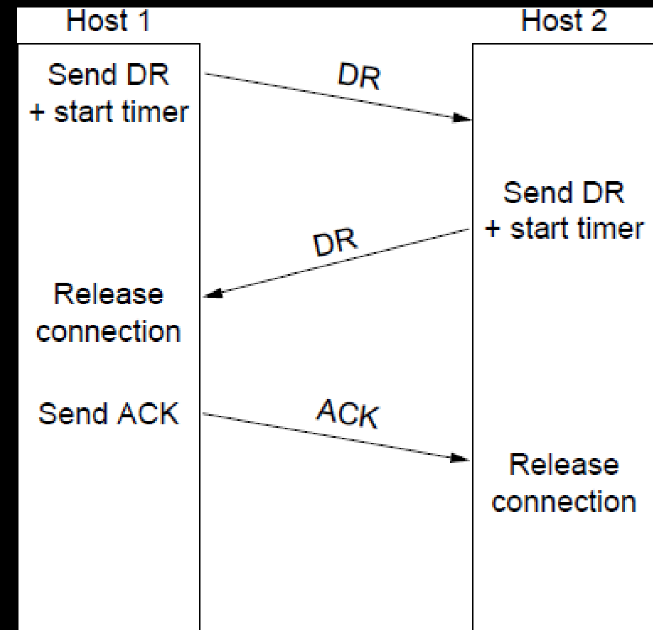
- Solution2: Symmetric release (both sides agree to release) can't be handled solely by the transport layer
  - Two-army problem shows pitfall of agreement



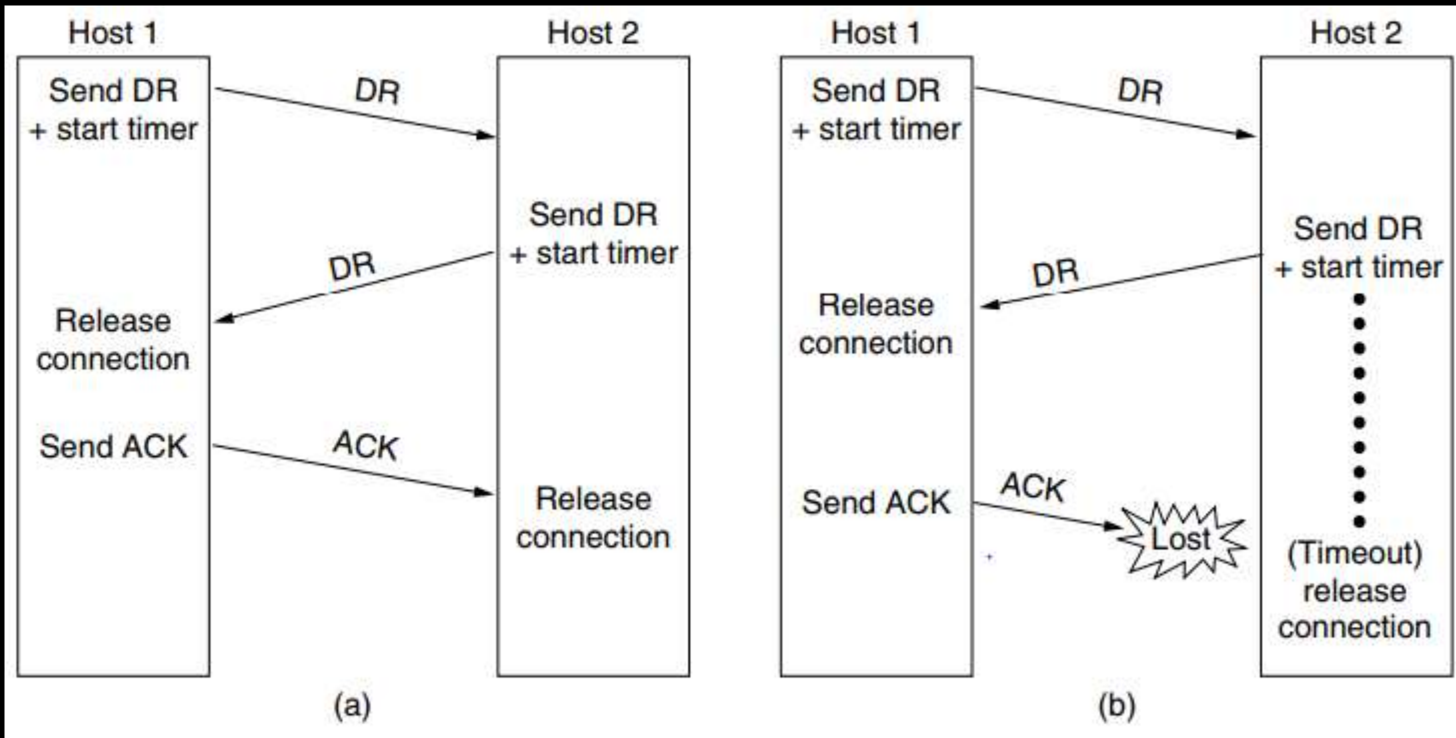
- A white army is encamped in a valley
- On both of the surrounding hillsides are blue armies.
- The white army is larger than either of the blue armies alone, but together the blue armies are larger than the white army.
- If either blue army attacks by itself, it will be defeated, but if the two blue armies attack simultaneously, they will be victorious.
- The blue armies want to synchronize their attacks.
- However, their only communication medium is to send messengers on foot down into the valley, they might be captured and the message lost.

- To see the relevance of the two-army problem to releasing connections, rather than to military affairs, just substitute ‘‘disconnect’’ for ‘‘attack.’’
- If neither side is prepared to disconnect until it is convinced that the other side is prepared to disconnect too, the disconnection will never happen.

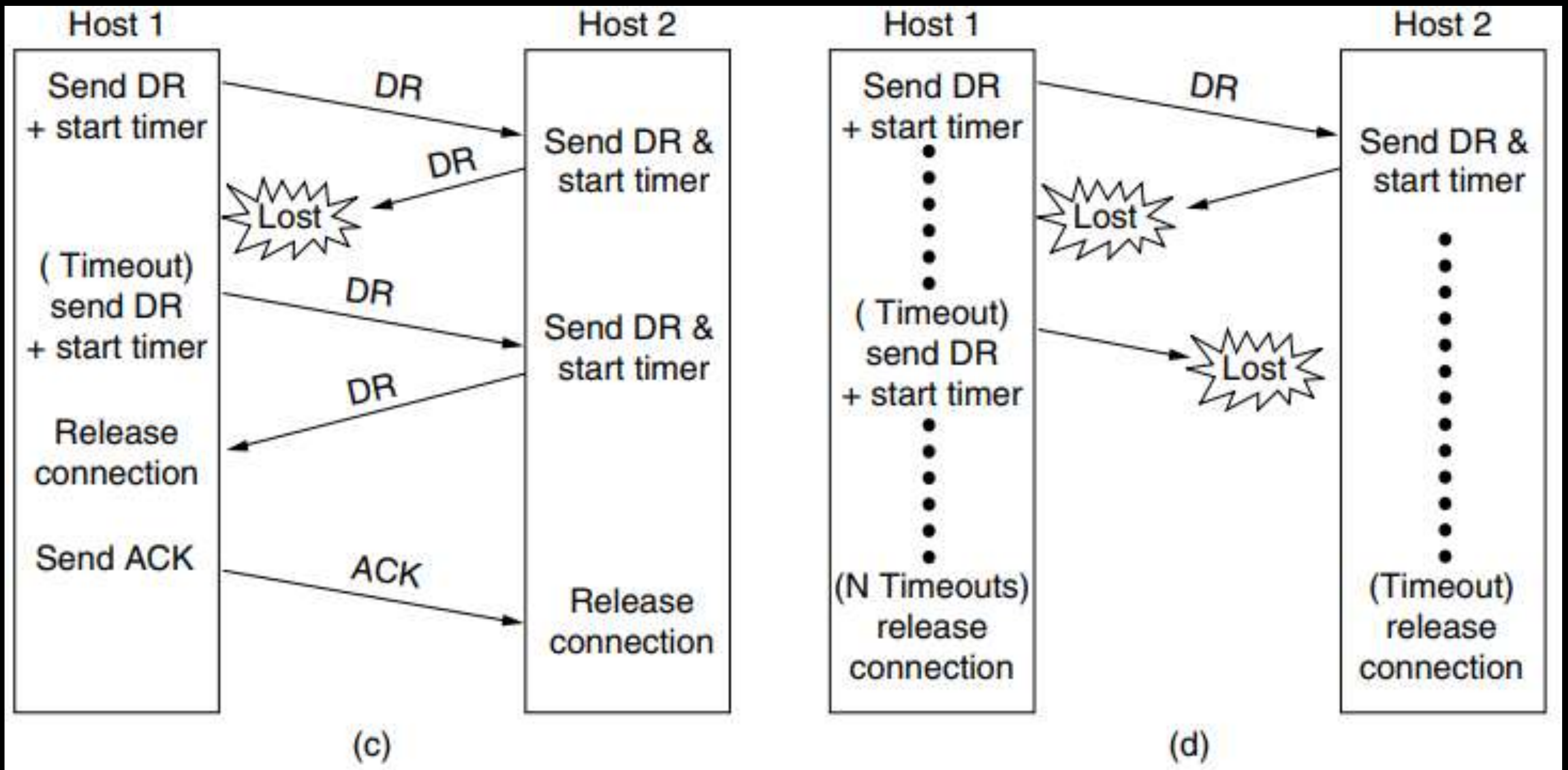
Connection Release



- Normal release sequence, initiated by transport user on Host 1
  - DR=Disconnect Request
  - Both DRs are ACKed by the other side



(a) Normal case of three-way handshake. (b) Final ACK lost.



(c) Response lost. (d) Response lost and subsequent DRs lost.

- The lesson here is that the transport user must be involved in deciding when to disconnect—the problem cannot be cleanly solved by the transport entities themselves.
- For example a web server can close the connection after it sends the response data to the client.
- The server can send the client a warning and abruptly shut the connection. If the client gets this warning, it will release its connection state then and there. If the client does not get the warning, it will eventually realize that the server is no longer talking to it and release the connection state.
- The client can then release the connection after successfully receiving the data.

Error Control and Flow Control

- Error Control: Ensure data is delivered without error (done in Data Link Layer(DLL) but there it over a single link, but here need to consider the network.) Foundation for error control is a sliding window (from Link layer) with checksums and retransmissions. i.e. CRC, ARQ, Stop-and-wait, sliding window

- Flow Control: Keeping a Fast transmitter from overrunning a slow receiver. Flow control manages buffering at sender/receiver

  - Issue is that data goes to/from the network and applications at different times

  - Window tells sender available buffering at receiver making a variable-size sliding window

  - Different buffer strategies trade efficiency / complexity

Error Control and Flow Control

- The solutions that are used at the transport layer are the same mechanisms as used by the data-link layer.
- Even though these mechanisms are used, there are differences in function and degree.
  - The transport layer checksum protects a segment while it crosses an entire network path which means It is an end-to-end check.
  - The link layer checks are not essential but nonetheless valuable for improving performance (since without them a corrupted packet can be sent along the entire path unnecessarily).

Error Control and Flow Control

- TCP connections have a bandwidth-delay product that is much larger than a single segment across a wifi link.
- For these situations, a large sliding window must be used. Stop-and-wait will cripple performance.
- Hence, we need to look at the issue of buffering data more carefully.
- With multiple connections, a substantial amount of buffering for the sliding windows, both at the sender and the receiver.

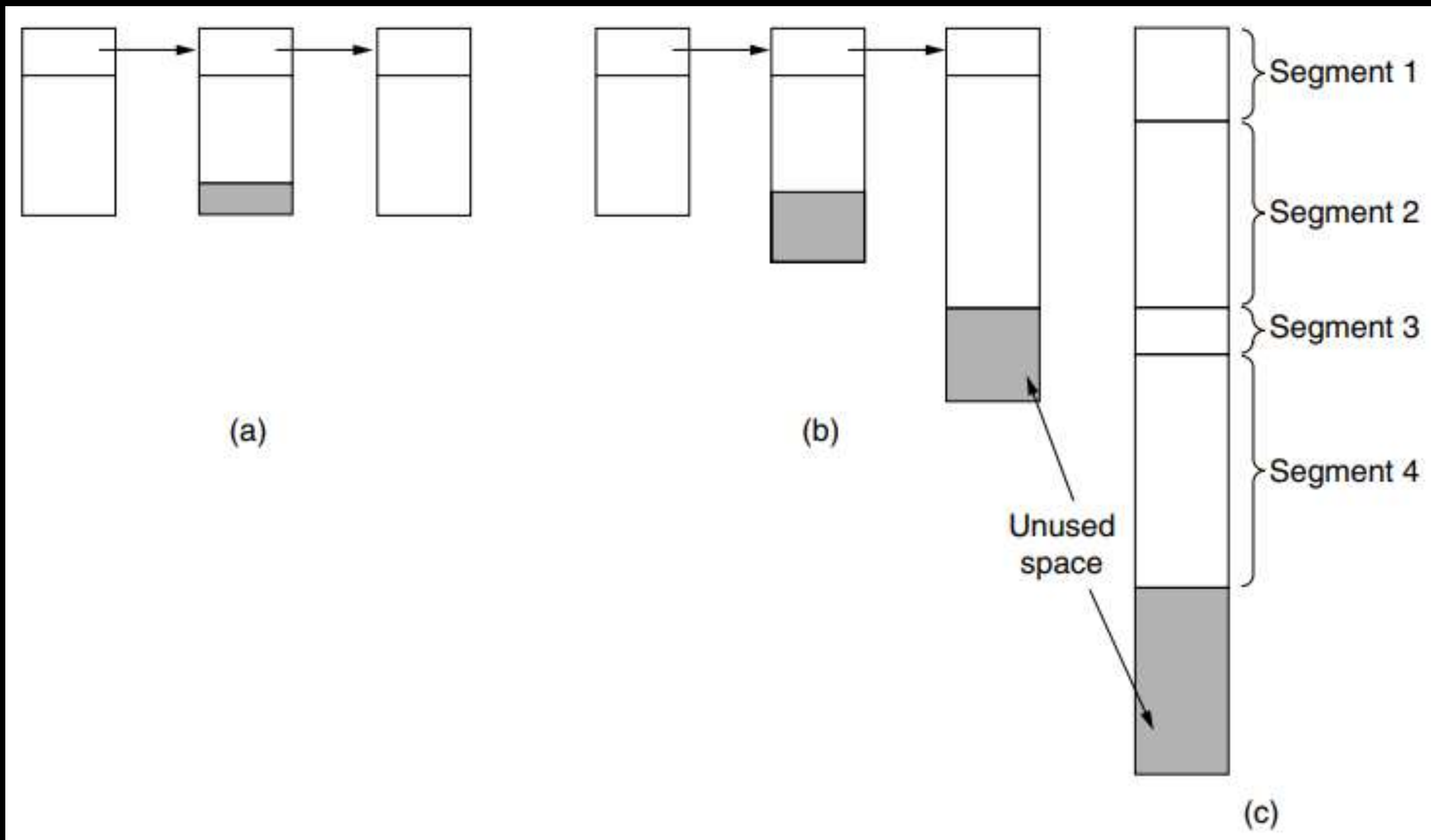
Buffer Management

- For low-bandwidth bursty traffic, sender buffers are used.
- For high-bandwidth traffic, receiver buffers are used.
- Problem: How to organize the buffer pool?
  - Approach to the buffer size problem is to use variable-sized buffers.
  - The advantage here is better memory utilization, at the price of more complicated buffer management.

Buffer Management

- The receiver may, for example, maintain a single buffer pool shared by all connections.
- When a segment comes in, an attempt is made to dynamically acquire a new buffer.
- If one is available, the segment is accepted; otherwise, it is discarded.
- The best trade-off between source buffering and destination buffering depends on the type of traffic carried by the connection

Buffer Management



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

Error Control and Flow Control (3)

Flow control example: A's data is limited by B's buffer

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

Dynamic buffer allocation. The arrows show the direction of transmission. An ellipsis (…) indicates a lost segment.

Buffer Management

- Problems with buffer allocation schemes can arise in datagram networks if control segments can get lost, which is a common case.
- Since control segments are not sequenced or timed out, deadlock can arise between communicating parties.
- To prevent this situation, each host should periodically send control segments giving the acknowledgement and buffer status on each connection.
- That way, the deadlock will be broken, sooner or later

Buffer Management

- When buffer space no longer limits the maximum flow, another bottleneck will appear: the carrying capacity of the network.
- What is needed is a mechanism that limits transmissions from the sender based on the network's carrying capacity rather than on the receiver's buffering capacity.

Multiplexing

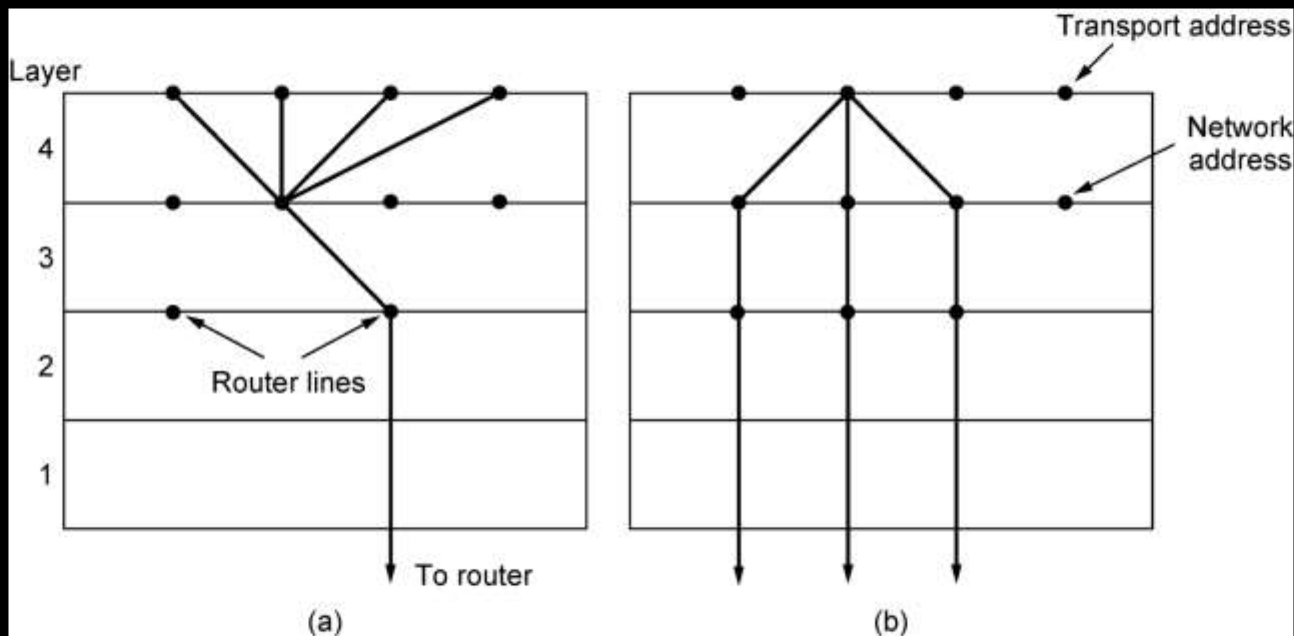
- In the transport layer, the need for multiplexing can arise in a number of ways
- If only one network address is available on a host, all transport connections on that machine have to use it.
- When a segment comes in, some way is needed to tell which process to give it to.
- This situation, called multiplexing.

Inverse Multiplexing

- When a host has multiple network paths that it can use.
- If a user needs more bandwidth or more reliability than one of the network paths can provide, a way out is to have a connection that distributes the traffic among multiple network paths on a round-robin basis.
- This modus operandi is called inverse multiplexing.

Multiplexing

- Kinds of transport / network sharing that can occur:
  - Multiplexing: connections share a network address
  - Inverse multiplexing: addresses share a connection



(a)Multiplexing. (b)Inverse multiplexing.

Crash Recovery

- If hosts and routers are subject to crashes or connections are long-lived (e.g., large software or media downloads), recovery from these crashes becomes an issue.
- Problem: How to recover from host crashes?
  - The server might send a broadcast segment to all other hosts, announcing that it has just crashed and requesting that its clients inform it of the status of all open connections.
  - Each client can be in one of two states: one segment outstanding, S1, or no segments outstanding, S0.
  - Based on only this state information, the client must decide whether to retransmit the most recent segment.

- S0: If a crash occurs after the acknowledgement has been sent but before the write has been fully completed.
- S1: The write has been done but the crash occurs before the acknowledgement can be sent.

- The server can be programmed in one of two ways:
  - acknowledge first or write first.
- The client can be programmed in one of four ways:
  - always retransmit the last segment,
  - never retransmit the last segment,
  - retransmit only in state S0, or
  - retransmit only in state S1.
- This gives eight combinations with each combination there is some set of events that makes the protocol fail.

- Three events are possible at the server:
  - sending an acknowledgement (A),
  - writing to the output process (W), and
  - crashing (C).
- The three events can occur in six different orderings:
  - AC(W),
  - AWC,
  - C(AW),
  - C(WA),
  - WAC, and
  - WC(A)

Crash Recovery and a sample protocol

	Strategy used by receiving host					
	First ACK, then write			First write, then ACK		
Strategy used by sending host	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 strategies.

- The result can be restated as ‘‘recovery from a layer N crash can only be done by layer N + 1, ’’ and then only if the higher layer retains enough status information to reconstruct where it was before the problem occurred.

Transport Control Protocol : TCP

- The TCP service model
- The TCP protocol
- The TCP segment header
- TCP connection establishment
- TCP connection release
- TCP sliding window
- TCP congestion control
- TCP timer management

TCP – Transmission Control Protocol

- TCP provides reliable, sequenced, end-to-end byte stream over an unreliable internetwork.
- TCP has been designed to adapt dynamically to internetwork difference viz. topology, bandwidths, delays, packet sizes etc.
- Defined in RFC 793 in Sept 1981(More info: https://www.ietf.org/standards/rfcs/)

TCP – Transmission Control Protocol

- TCP Entity accepts user data from local processes, breaks them up into pieces $\leq$ **64 KB** (in practice1460 bytes data to fit in an Ethernet frame with IP TCP headers) and sends each piece as a separate IP datagram.
- Note: Ethernet Frame data size is 40 – 1500 bytes.
- Upon arriving at a m/c the transport entity takes it and constructs the original byte stream.

The TCP Service Model

- To obtain the TCP service both the sender and receiver create endpoints using Sockets.
- Each socket has IP address(32 bit) and Port(16 bit)
  - Port is the TCP name for TSAP
- To get TCP service connection is to be explicitly established between a socket on one m/c and a socket on another m/c.

The TCP Service Model

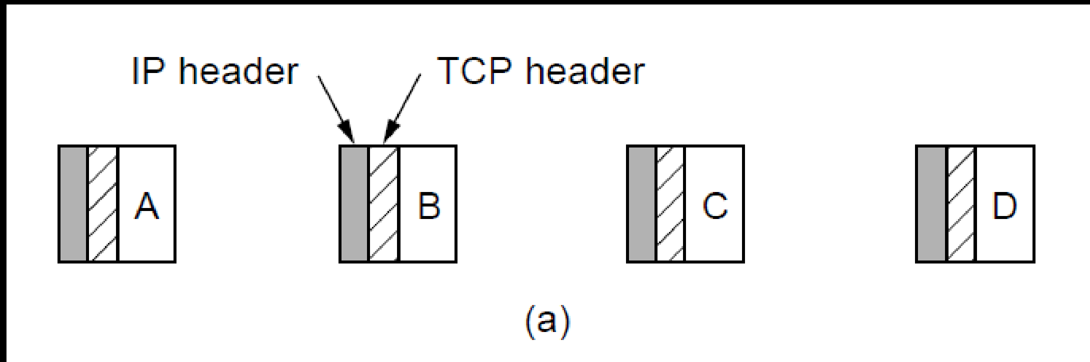
Port	Protocol	Use
20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

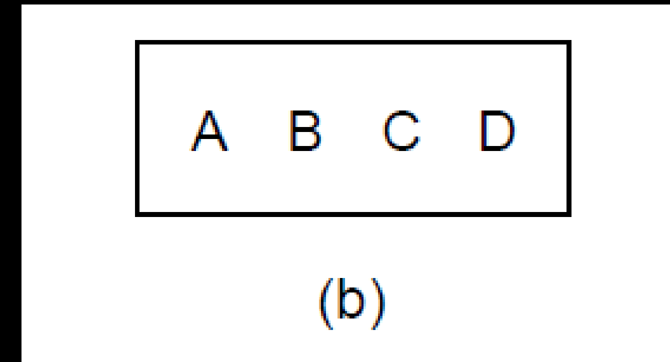
- Examples of some assigned ports
- Internet Daemon (inetd) attach itself to multiple ports and wait for the first connection request, then fork to that service

TCP Service Model

- All TCP connections are full duplex (both directions) and point-to-point (has exactly two end points)
- TCP doesn’t support multicasting or broadcasting.
- TCP is byte stream and not message stream.
- Message boundaries are not preserved end-to-end

The TCP Service Model:Byte stream example





- Example: A Process does four 512 bytes writes to a TCP stream
- Four 512-byte segments sent as separate IP diagrams
- The 2048 bytes of data delivered to the application in a single READ call

The TCP Service Model

a) TCP may send data immediately or buffer it to send it in one go when an application passes data to it.

b) To force data out -- uses the PUSH flag

c) Too many PUSH-es then all PUSH are collected together and sent.

d) URGENT – on pushing Ctrl-C to break-off remote computation, the sending application puts some control flag

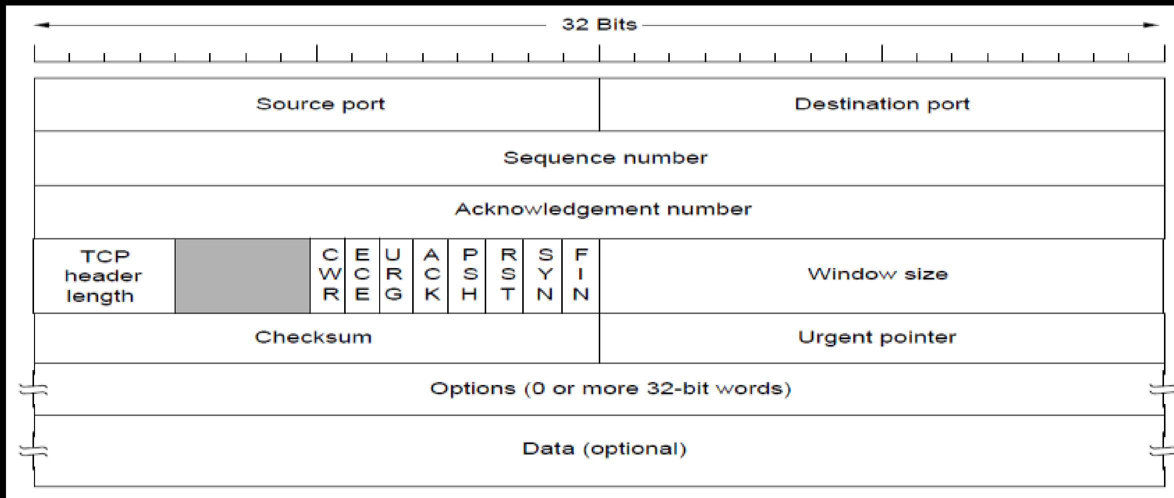
The TCP Protocol

- Every byte on TCP connection has its own 32- bit sequence no.
- Sending and Receiving Entities exchange data in the form of TCP segments.
- TCP segment consists of a fixed 20-byte header (plus an optional part) followed by zero or more data.
- Factors limiting segment size:
  - IP payload size (65515 excludes IP header)
  - Each Links MTU - Ethernet its 1500 byte.

TCP Header Size

- TCP segment can have data bytes up to 65,535 – 20 byte(IP header) - 20 byte (TCP header)
- That gives 65495 bytes.
  - Also note that 65535 = 64 KB where K = 1024

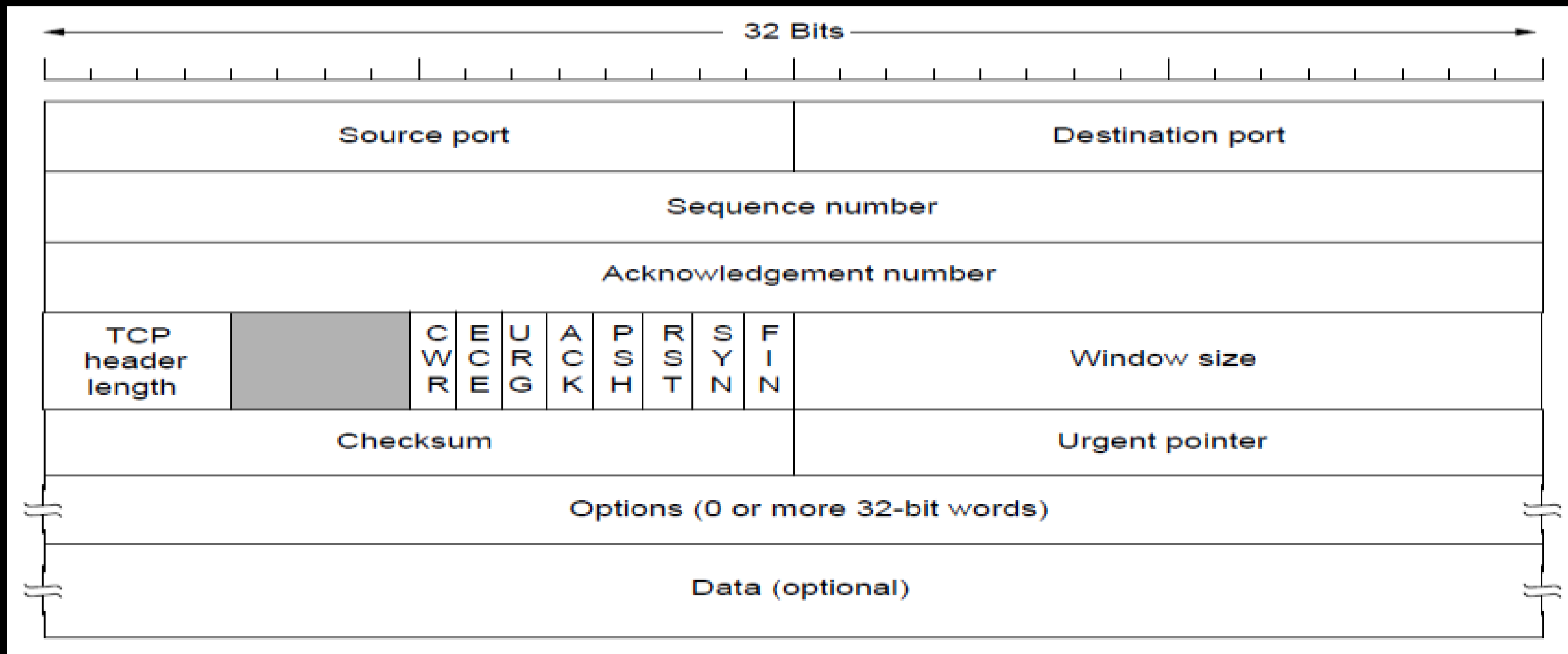
The TCP Segment Header



- PSH – Pushed
- RST – reset
- ACK =0 (REQUEST)
- ACK = 1(ACCEPT)
- ACK – Acknowledgement

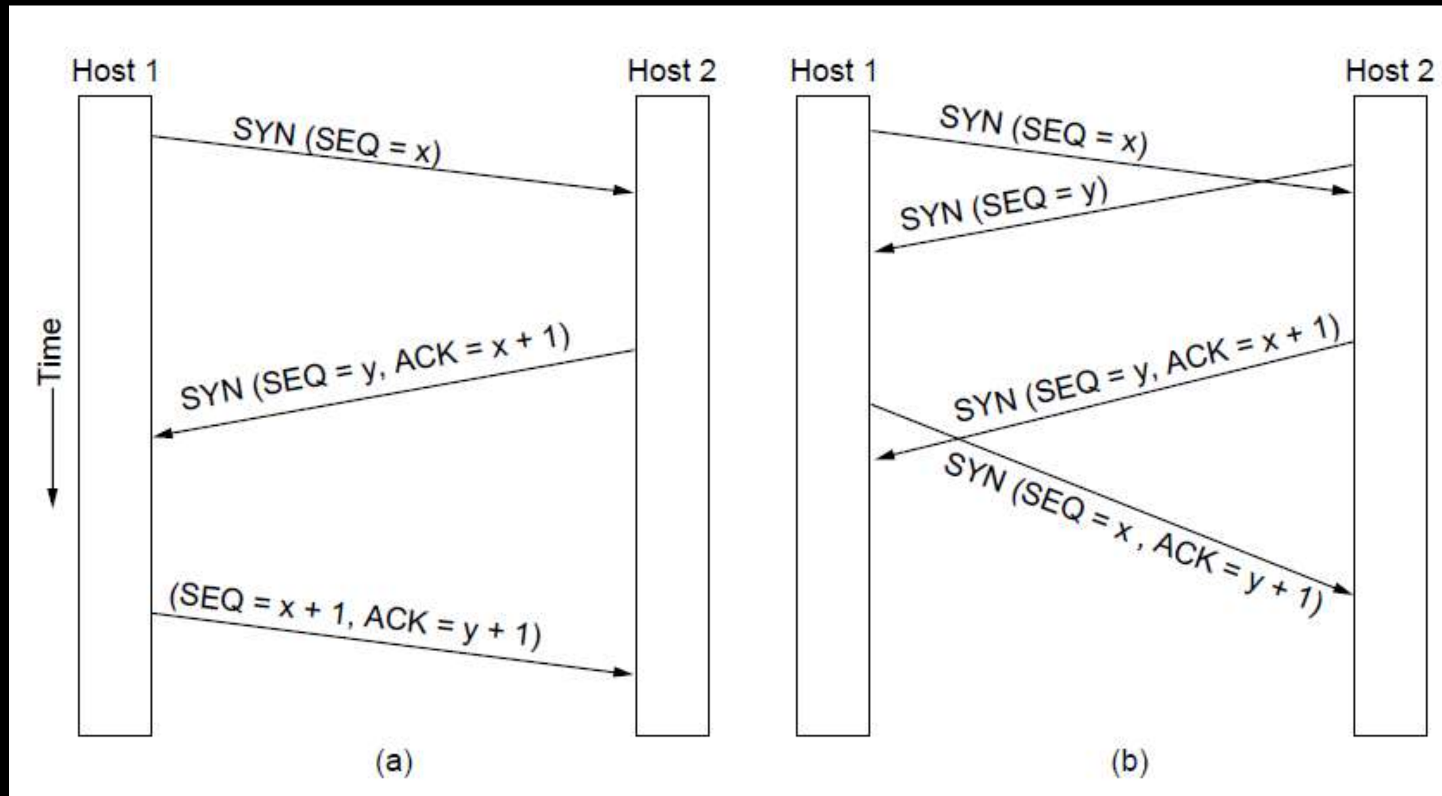
- Ack no = one more than what is next expected
- TCP Header Length – how many optional field
- URG – Urgent, (URGENT POINTER – OFFSET)
- SYN = 1 (CONNECTION REQUEST, CONNECTION ACEEPTED),
- WINDOW SIZE = How many buffers may be granted, can be zero.
- FIN bit is used for connection release. Both SYN, FIN have Sequence Nos.
- CWR/ECE – Congestion controlling bits
- ECE – Echo, CWR – Congestion window reduced

The TCP Segment Header



- Check Sum: TCP Header + Data + Pseudo Header
- **OPTIONs:** are of variable length and max up to 40 bytes. Each option has Type-value-length encoding.
- **MSS:** max segment size a host is willing to accept
  - (536 bytes is default, 536+20 TCP Hdr = 556 B every internet host has to accept.)
- **Window size:** Sndr & Rcvr can use to it to negotiate at time of connection establishment so as to scale the window size.

TCP Connection Establishment

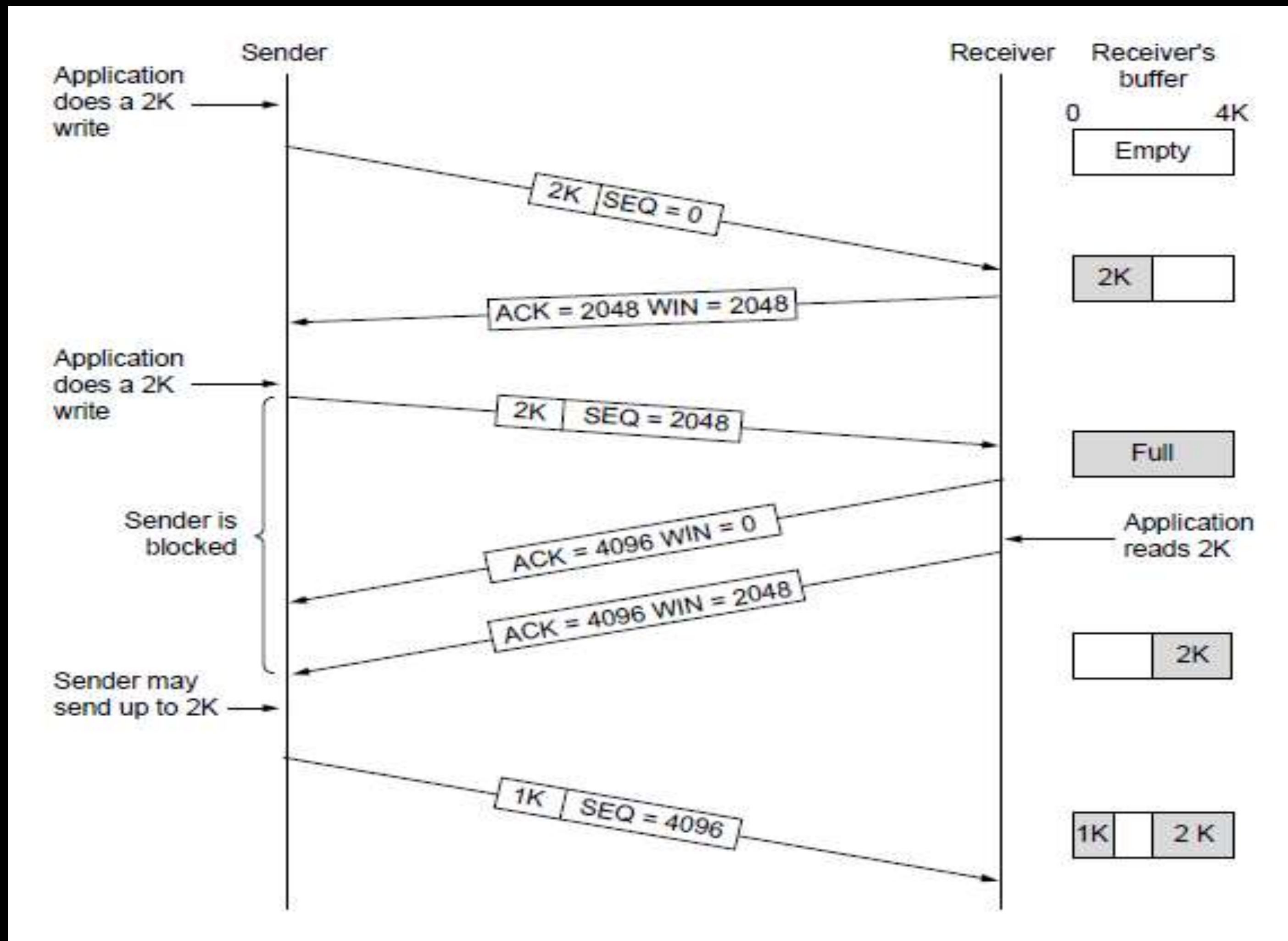


- TCP connection establishment in the normal case.
- Req: SYN=1, ACK=0, Rep: SYN=1, ACK=1, Reject: RST=1
- Simultaneous connection establishment on both sides – only one is set up as connections are identified by their end points.

TCP Connection Release

- Either party send with the FIN bit set
- When the FIN is acknowledged, that direction is shut down for new data
- Full closing (TWO FIN and TWO ACK)

TCP Window Mgmt – Sliding Window (1)



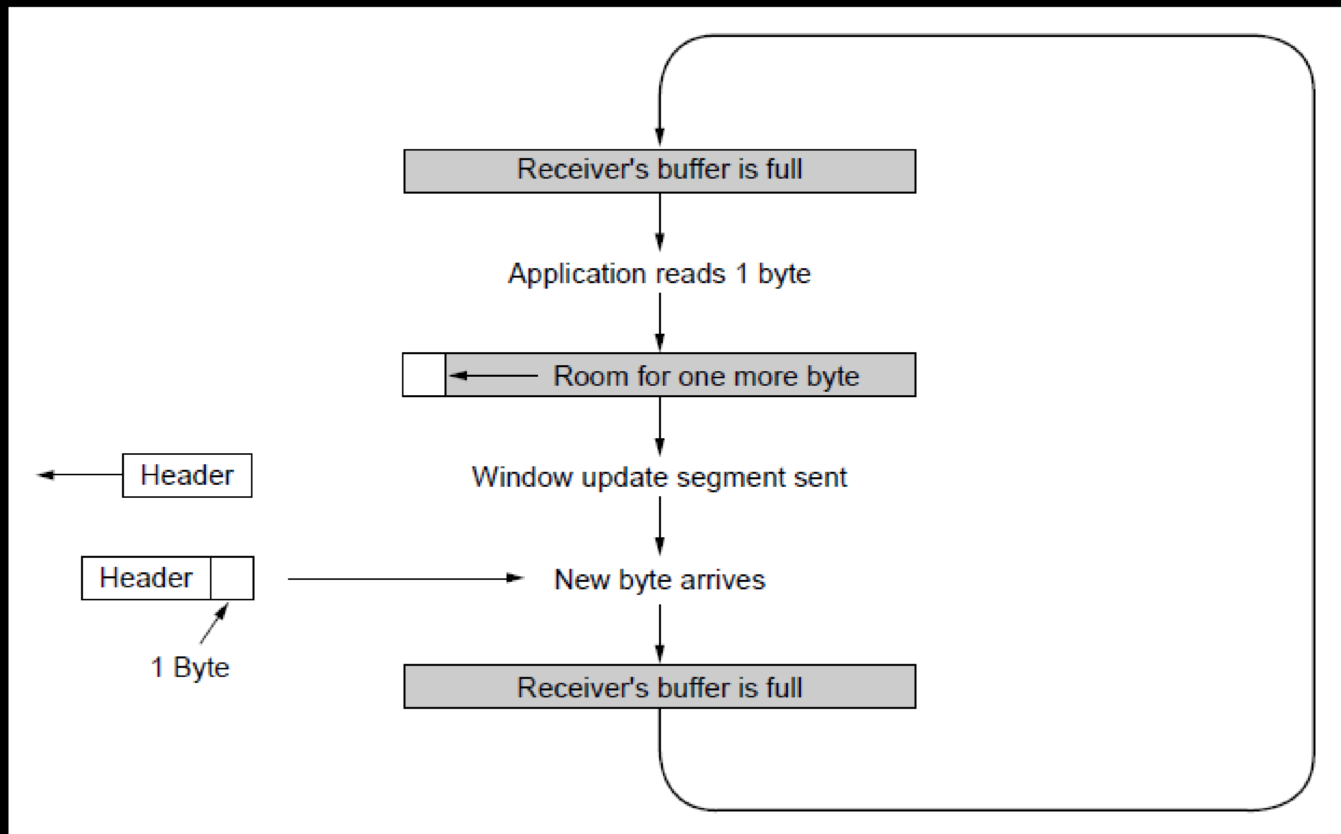
TCP Window Mgmt - Sliding Window(2)

- Window management in TCP follows that
- If window = 0, sender stops sending data with 2 exception
  - When sending Urgent bytes or
  - Sending 1- byte to make the receiver to re-announce the next byte expected.

Nagle's Algorithm (works at sender side)

- For Interactive Editor application--- Sending 1-byte would involve 162 bytes (40 to send, 40 to ACK, 41 to update, 40 to ack)
- Nagle's Algorithm – When data comes into the sender one byte at a time, just send the first byte and buffer the rest until the outstanding byte is acknowledged

TCP Window Mgmt- Sliding Window at receiver side (2)

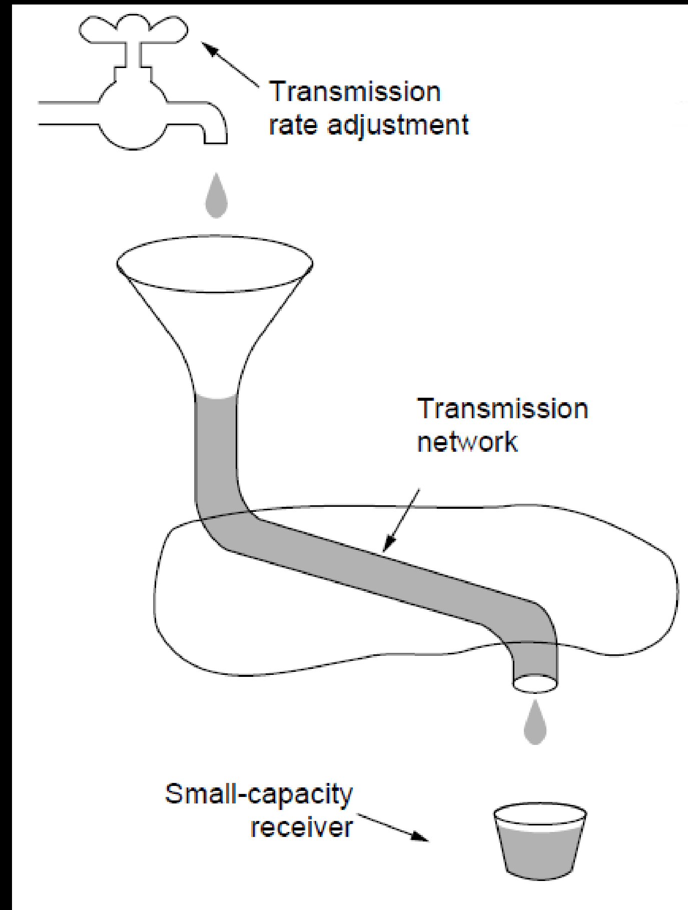


- Silly window syndrome … Clark's solution – prevent receiver from sending a window update for 1 byte.
- Specifically the receiver should not send a window update until it get the maximum segment advertised free

Cumulative Acknowledgements

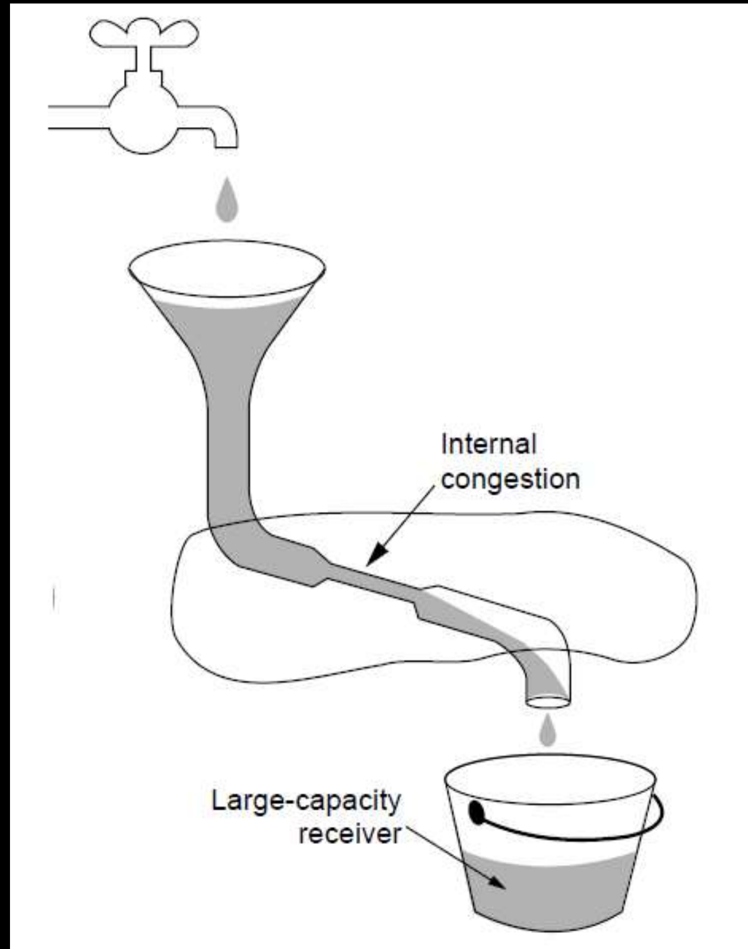
- Receiver – Block READ from the application until a large chunk of data arrives
- Out of order – 0, 1,2,4,5,6,7 - acknowledge up to 2 and wait for segment 3 to be transmitted.

TCP Congestion Control: Regulating the Sending Rate (1)



A fast network feeding a low-capacity receiver

Regulating the Sending Rate (2)

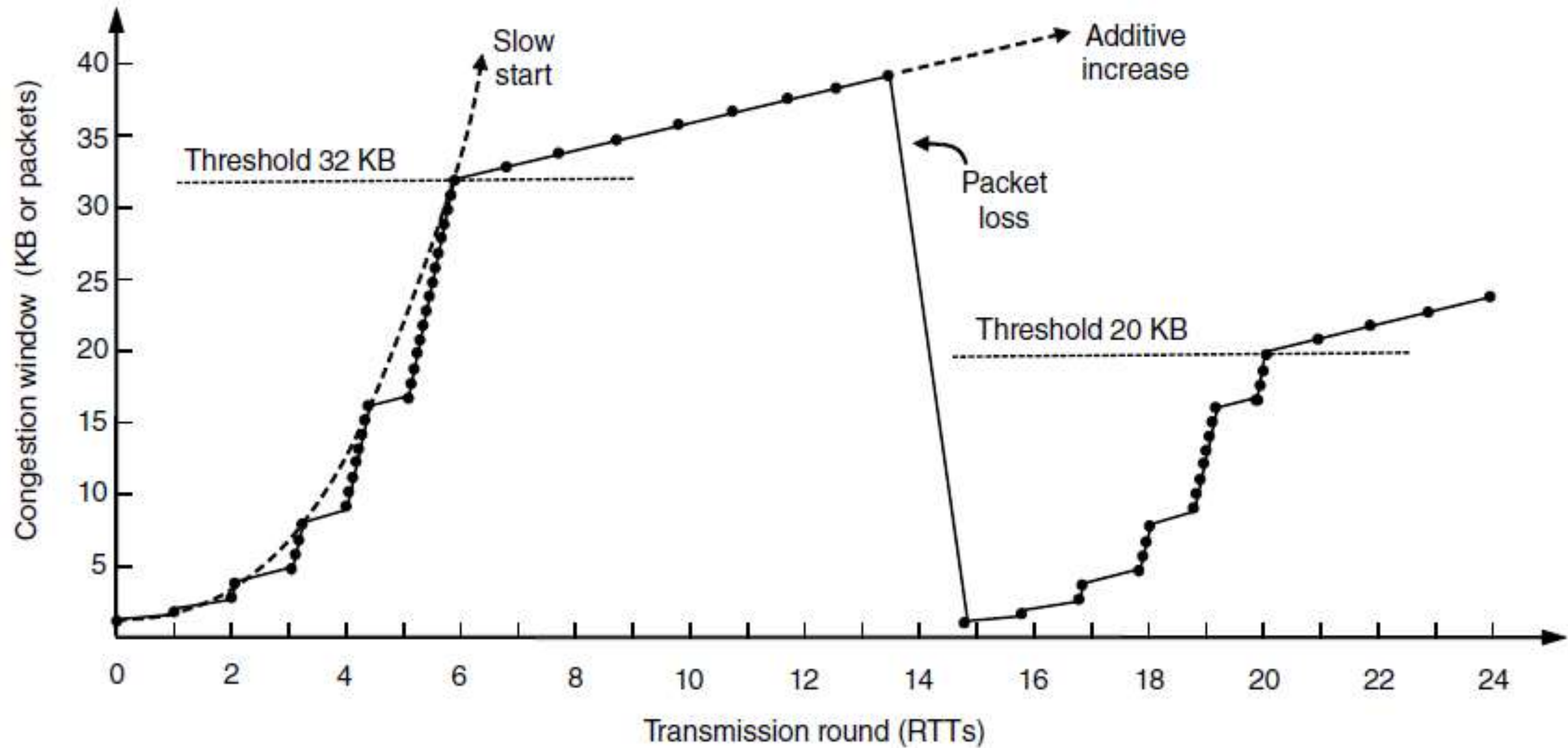


A slow network feeding a high-capacity receiver

TCP Congestion Control

- Two windows are used in TCP
  - The window a receiver grants
  - Congestion window
- When using Slow start – 1024 byte - window size moves exponentially
- When a threshold is set it grows linearly

TCP Congestion Control (3)



Slow start followed by additive increase in TCP

TCP Timer Management

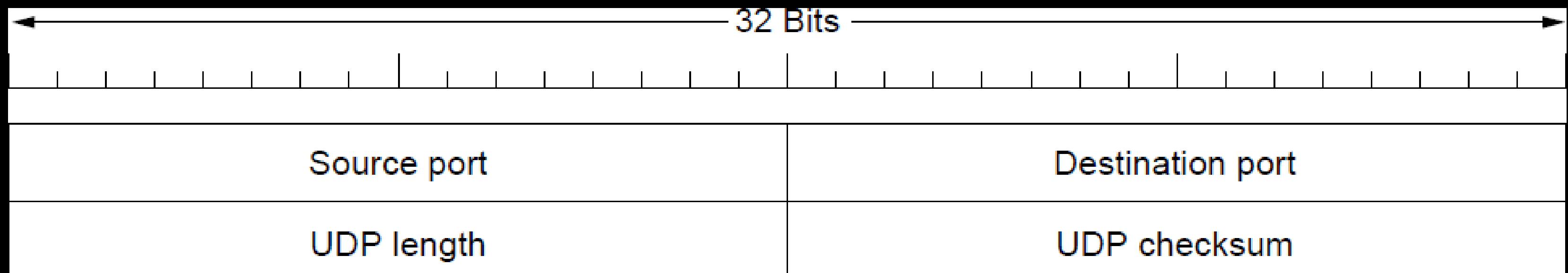
- TCP uses multiple timers (at-least conceptually) to do its work. The most important of these is the RTO (Retransmission Time Out).
- When a segment is sent, a retransmission timer is started.
- If the segment is acknowledged before the timer expires, the timer is stopped.
- If, on the other hand, the timer goes off before the acknowledgement comes in, the segment is retransmitted (and the timer is started again).

User Datagram Protocol – UDP: Introduction to UDP

- An unreliable, connectionless transport layer protocol.
  - Protocol No for UDP is 17
  - UDP useful in client-server situations where client sends a short request to the server and expects a short reply back. If time-out retransmit rather than establish connection. Eg: DNS application
  - Does not do flow-control, congestion control or retransmission upon receiving bad segment.
  - Provides interface to IP protocol with added feature of demultiplexing multiple processes using ports & optional end-to-end error detection.

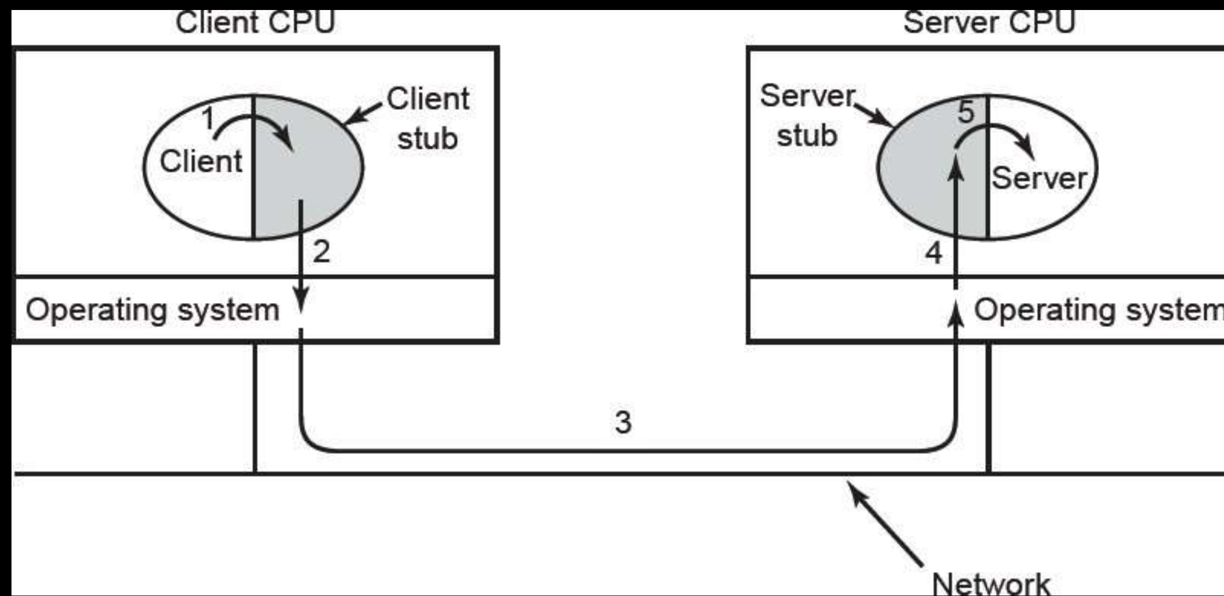
Introduction to UDP

- UDP (User Datagram Protocol) Header is of 8 bytes
- Fields are : ports (TSAPs), length and checksum.
- UDP Length field includes 8 byte header and the data.
- Source Port - needed when reply is to be sent back to source.
- Destination port: Receiver's Port of the outgoing segment.
- It checksums the header, the data and a conceptual IP pseudo-header.



RPC (Remote Procedure Call)

- RPC connects applications over the network with the familiar abstraction of procedure calls
- Stubs package parameters/results into a message
- UDP with retransmissions is a low-latency transport



RPC (Remote Procedure Call)

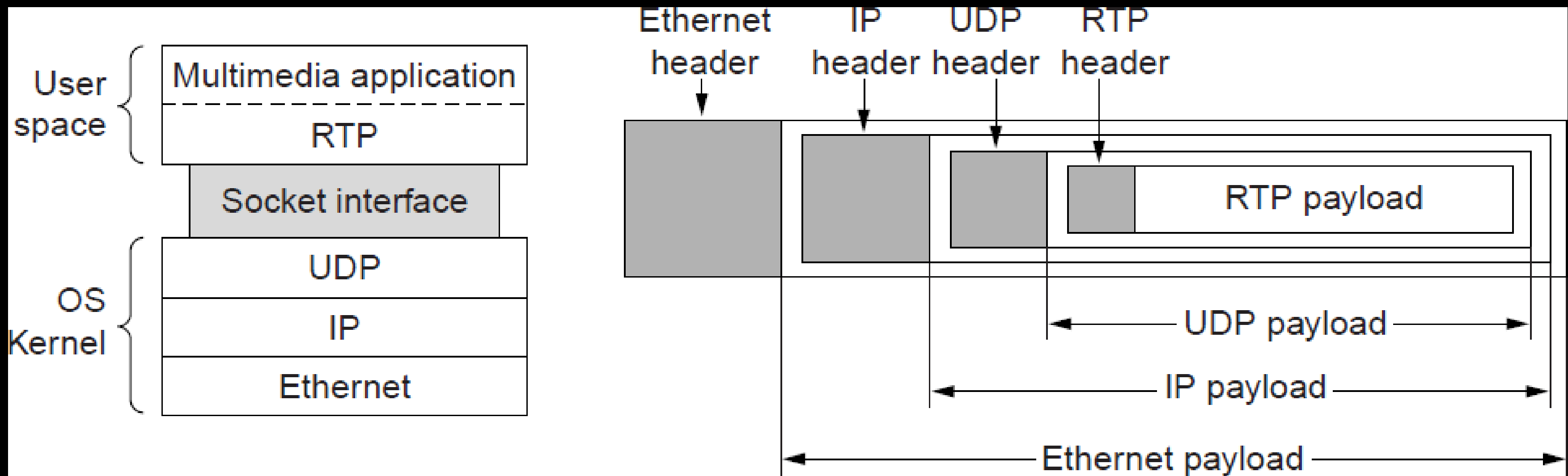
- Steps in RPC

1. Client calls the client stub
2. Client Stub packs the parameters into message and makes a system call to send the message (Packing of parameters is Marshalling)
3. OS sends the message from client m/c to server m/c
4. OS passes message to server stub. (Unpacking of parameters is Un-Marshalling, server stub does it).
5. Server stub calls the server procedure.

Reply traces the same path in the other direction.

Real-Time Protocol (1)

- RTP (Real-time Transport Protocol) provides support for sending real-time media over UDP
- Often implemented as part of the application



Real-Time Protocol (2)

- RTP (Real-time Transport Protocol) provides support for sending real-time media over UDP
  - Often implemented as part of the application
  - It has two parts
    - First Part - For Transporting audio and video data in packets.
    - Second Part - Relates to processing that takes care mostly at the receiver to play out the audio and video at the right time.

The Real Time Control Protocol (RTCP)

- The Real Time Control Protocol (RTCP) is a companion protocol to RTP that is also defined in RFC 3550.
- RTCP handles feedback, synchronization, and user interfaces to real time operations.
- Instructor was tasked by Microsoft to work within the IETF to develop RTCP to flexibly enable a standards-based user control for the Microsoft Netshow product's distributed multimedia Internet streaming operations: streaming fast forward, pause, stop, rewind, skip to a predetermined point, etc. It can change resolutions and do a wide-variety of control tasks to the streamed multimedia.