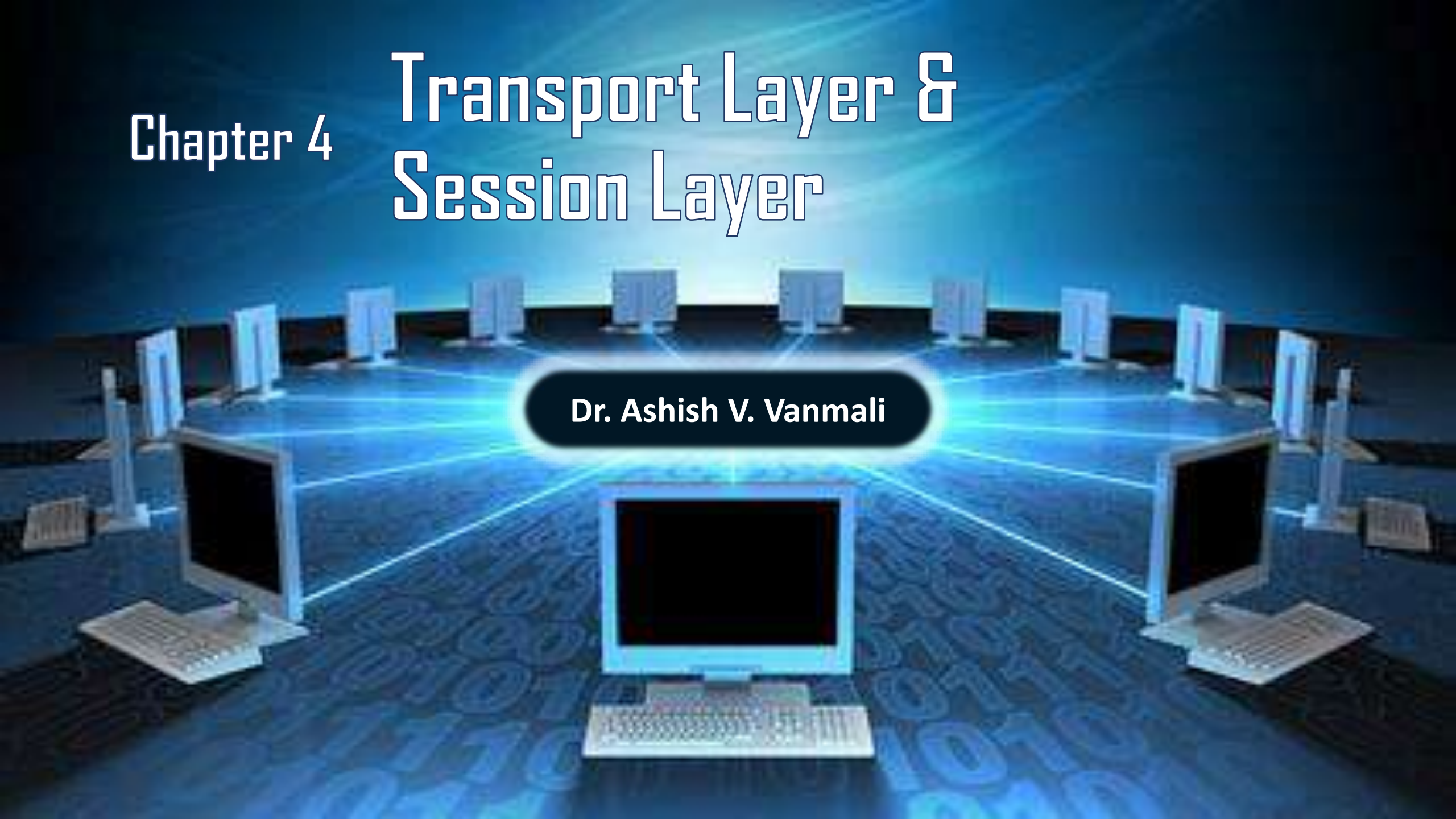
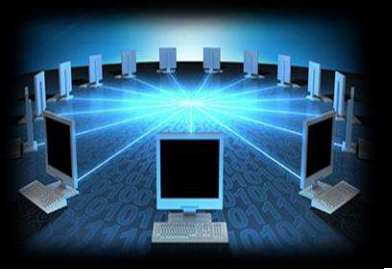
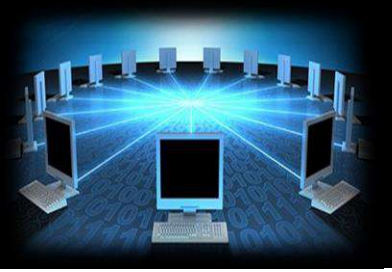

Chapter 4
Transport Layer &
Session Layer
Dr. Ashish V. Vanmali

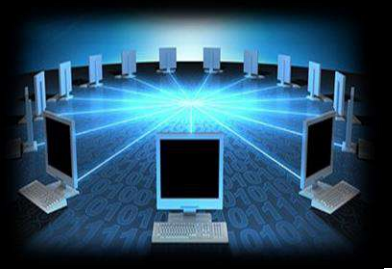


Outline of the Chapter

- **Transport Layer**
  - Transport Layer Services
  - User Datagram Protocol
  - Transmission Control Protocol
- **Session Layer**
  - Session Layer Design Issues
  - Session Layer Protocol - Remote Procedure Call (RPC)



Transport Layer Services

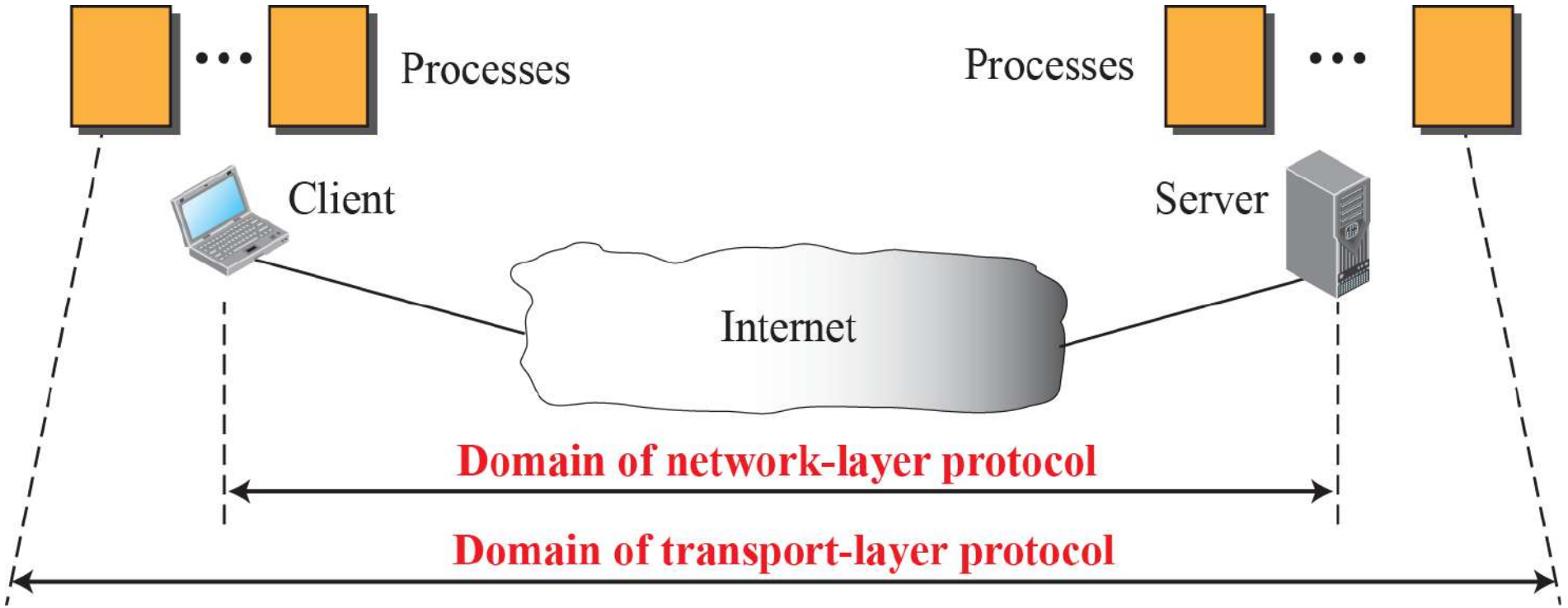


Transport Layer Services

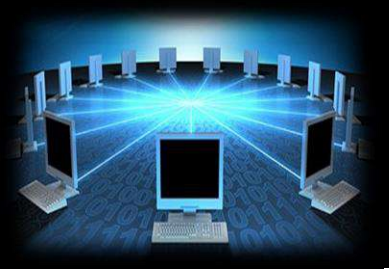
- The transport layer is located between the network layer and the application layer.
- The transport layer is responsible for providing services to the application layer; it receives services from the network layer.
- The transport layer provides a **process-to-process communication** between two application layers.
- Communication is provided using a logical connection, which means that the two application layers assume that there is an imaginary direct connection through which they can send and receive messages.

Transport Layer Services

❑ Process-to-Process Communication



Ref: Behrouz A. Forouzan, Data Communications and Networking, 6th Edition, Mc Graw Hill education.



Transport Layer Services

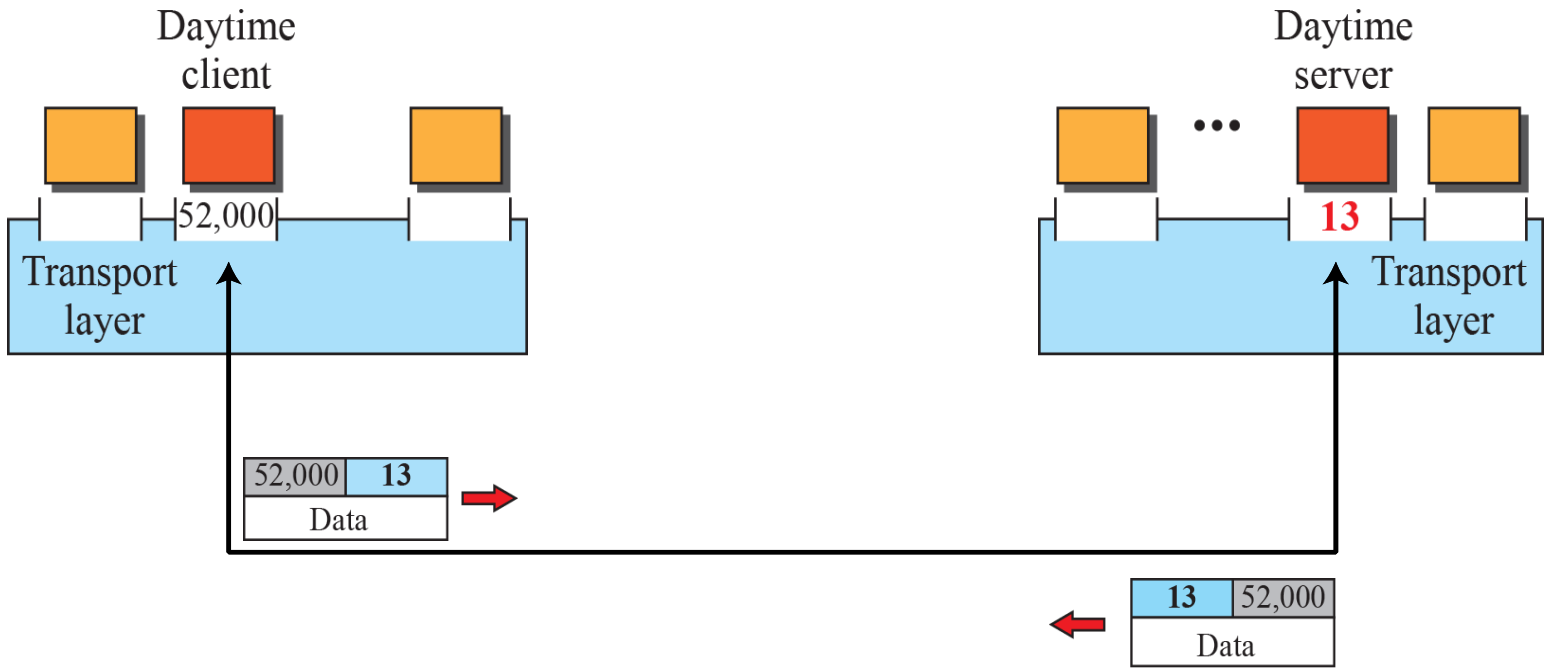
Process-to-Process Communication

- The most common way to achieve process-to-process communication is through the **client-server paradigm**.
- A process on the local host, called a **client**, needs services from a process usually on the remote host, called a **server**.
- Both client and servers may have several programs running at the same time.
- Therefore, for process-to-process communication, we must define the following:
  1. Local host
  2. Local process
  3. Remote host
  4. Remote process

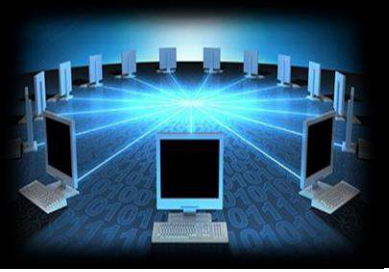
Transport Layer Services

Port Number

- The local host and the remote host are defined using **IP addresses**.
- To define the processes, we need second identifiers called **port numbers**.
- In the TCP/IP protocol suite, the port numbers are 16-bit integers between 0 and 65,535.



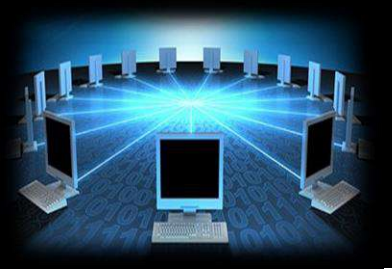
Ref: Behrouz A. Forouzan, Data Communications and Networking, 6th Edition, Mc Graw Hill education.



Transport Layer Services

Port Number

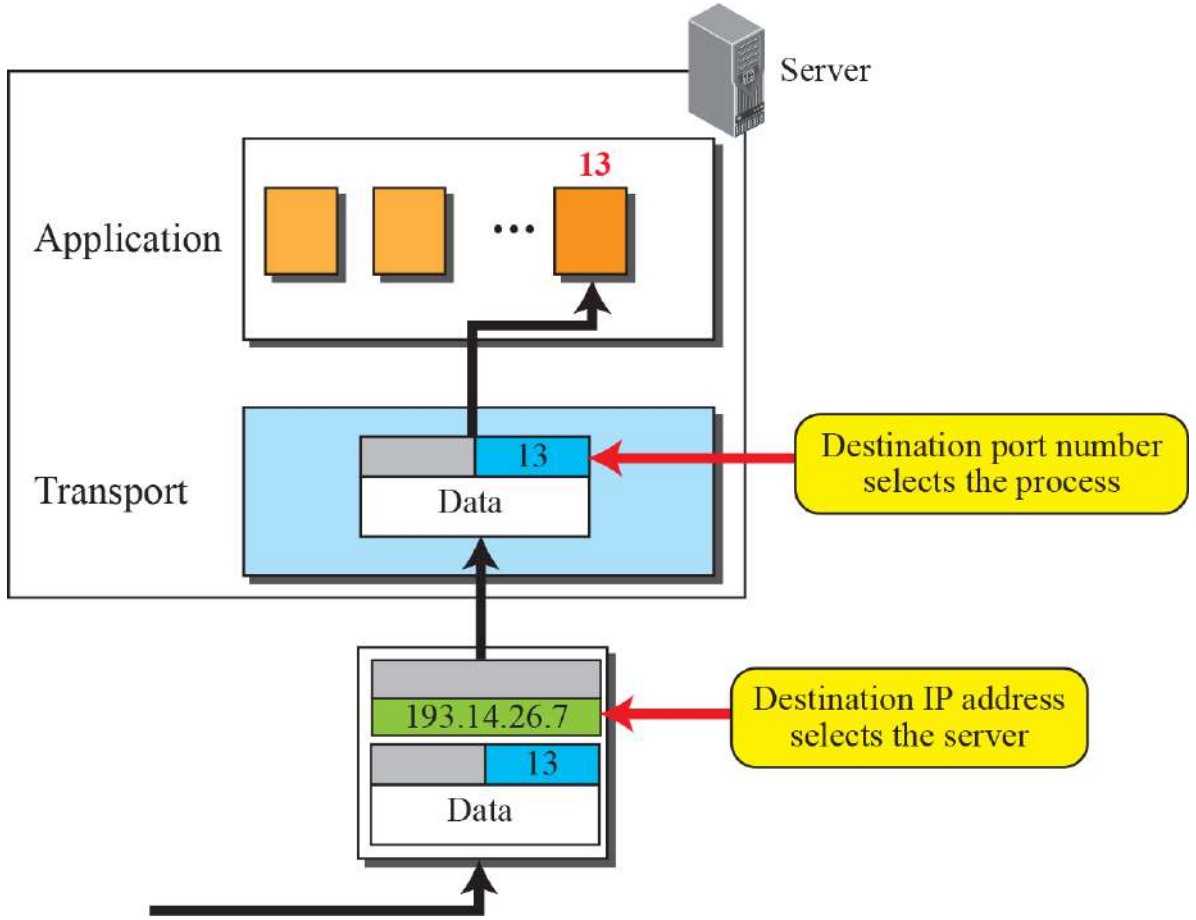
- The client program defines itself with a port number, called the **ephemeral port number**.
- An ephemeral port number is recommended to be greater than 1,023 for some client/server programs to work properly.
- The server process must also define itself with a port number.
- TCP/IP has decided to use universal port numbers for servers; these are called **well-known port numbers**.
- Every client process knows the well-known port number of the corresponding server process.



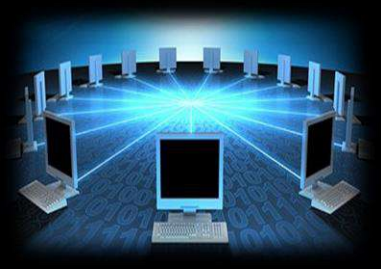
Transport Layer Services

Port Number

- IP addresses and port numbers play different roles in selecting the final destination of data.
- The destination IP address defines the host among the different hosts in the world.
- After the host has been selected, the port number defines one of the processes on this particular host



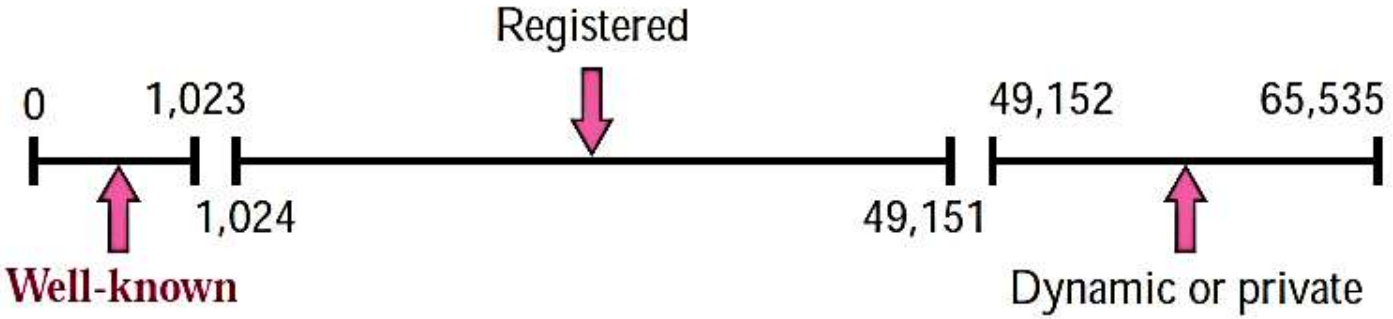
Ref: Behrouz A. Forouzan, Data Communications and Networking, 6th Edition, Mc Graw Hill education.



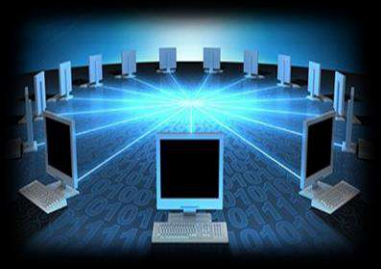
Transport Layer Services

ICANN Range of Port Numbers

- Internet Corporation for Assigned Names and Numbers (ICANN) has divided the port numbers into three ranges: well-known, registered, and dynamic (or private).
- **Well-known ports:** The ports ranging from 0 to 1,023 are assigned and controlled by ICANN. These are the well-known ports.
- **Registered ports:** The ports ranging from 1,024 to 49,151 are not assigned or controlled by ICANN. They can only be registered with ICANN to prevent duplication.
- **Dynamic ports:** The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used as temporary or private port numbers.



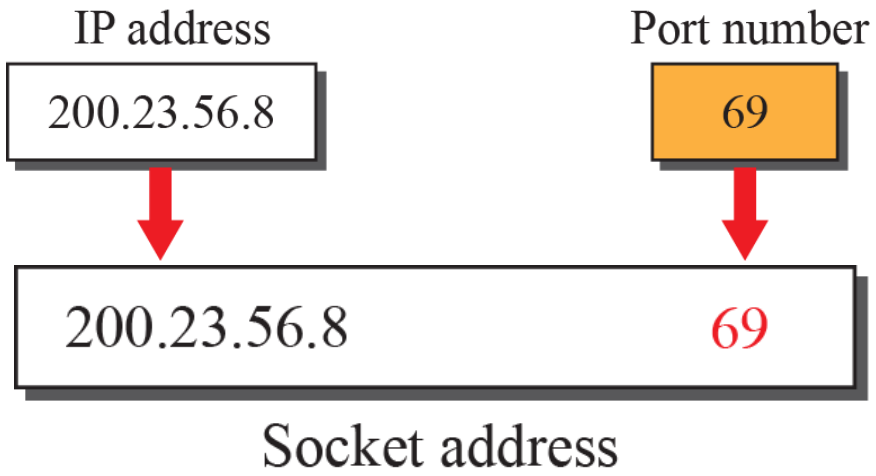
Ref: Behrouz A. Forouzan, Data Communications and Networking, 6th Edition, Mc Graw Hill education.



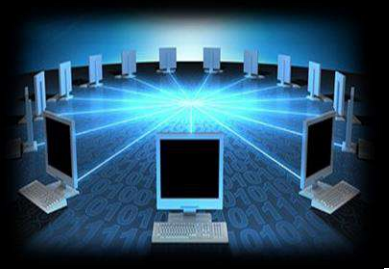
Transport Layer Services

Socket Address

- A transport-layer protocol in the TCP suite needs both the IP address and the port number, at each end, to make a connection.
- The combination of an IP address and a port number is called a **socket address**.
- The client socket address defines the client process uniquely just as the server socket address defines the server process uniquely



Ref: Behrouz A. Forouzan, Data Communications and Networking, 6th Edition, Mc Graw Hill education.



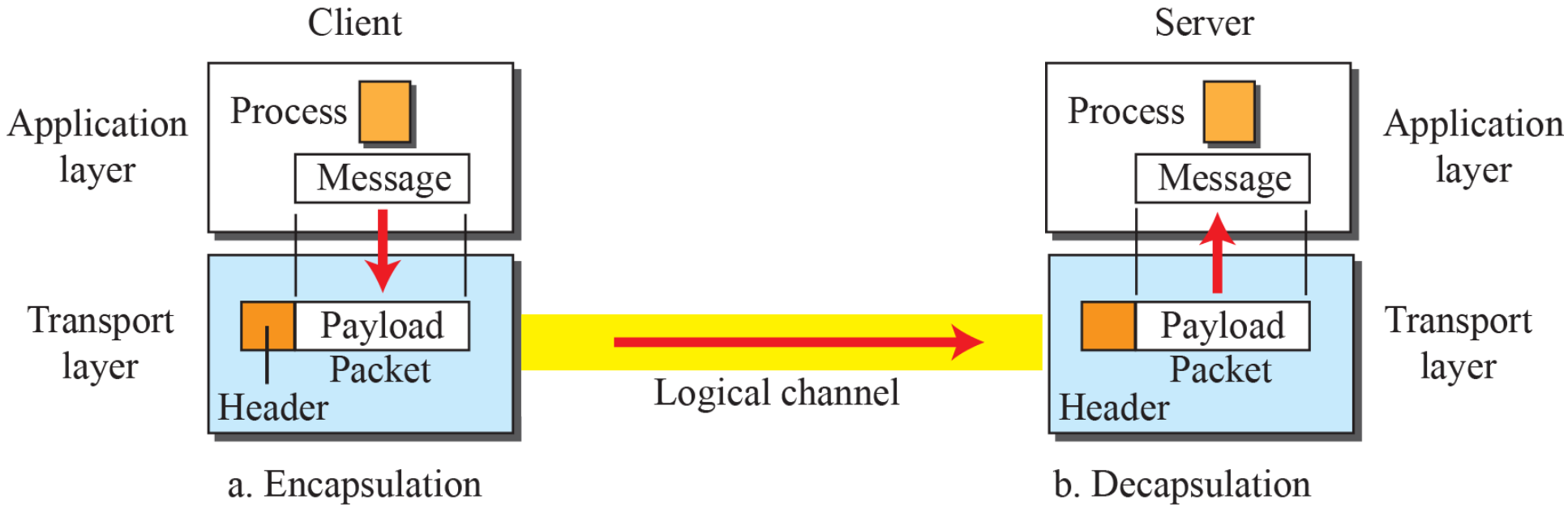
Transport Layer Services

Encapsulation and Decapsulation

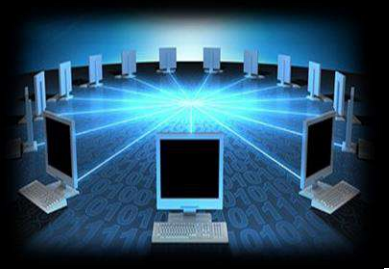
- To send a message from one process to another, the transport layer protocol encapsulates and decapsulates messages.
- **Encapsulation** happens at the sender site.
- When a process has a message to send, it passes the message to the transport layer along with a pair of socket addresses and some other pieces of information that depends on the transport layer protocol.
- The transport layer receives the data and adds the transport-layer header.
- **Decapsulation** happens at the receiver site.
- When the message arrives at the destination transport layer, the header is dropped and the transport layer delivers the message to the process running at the application layer.
- The sender socket address is passed to the process in case it needs to respond to the message received.

Transport Layer Services

❑ Encapsulation and Decapsulation



Ref: Behrouz A. Forouzan, Data Communications and Networking, 6th Edition, Mc Graw Hill education.



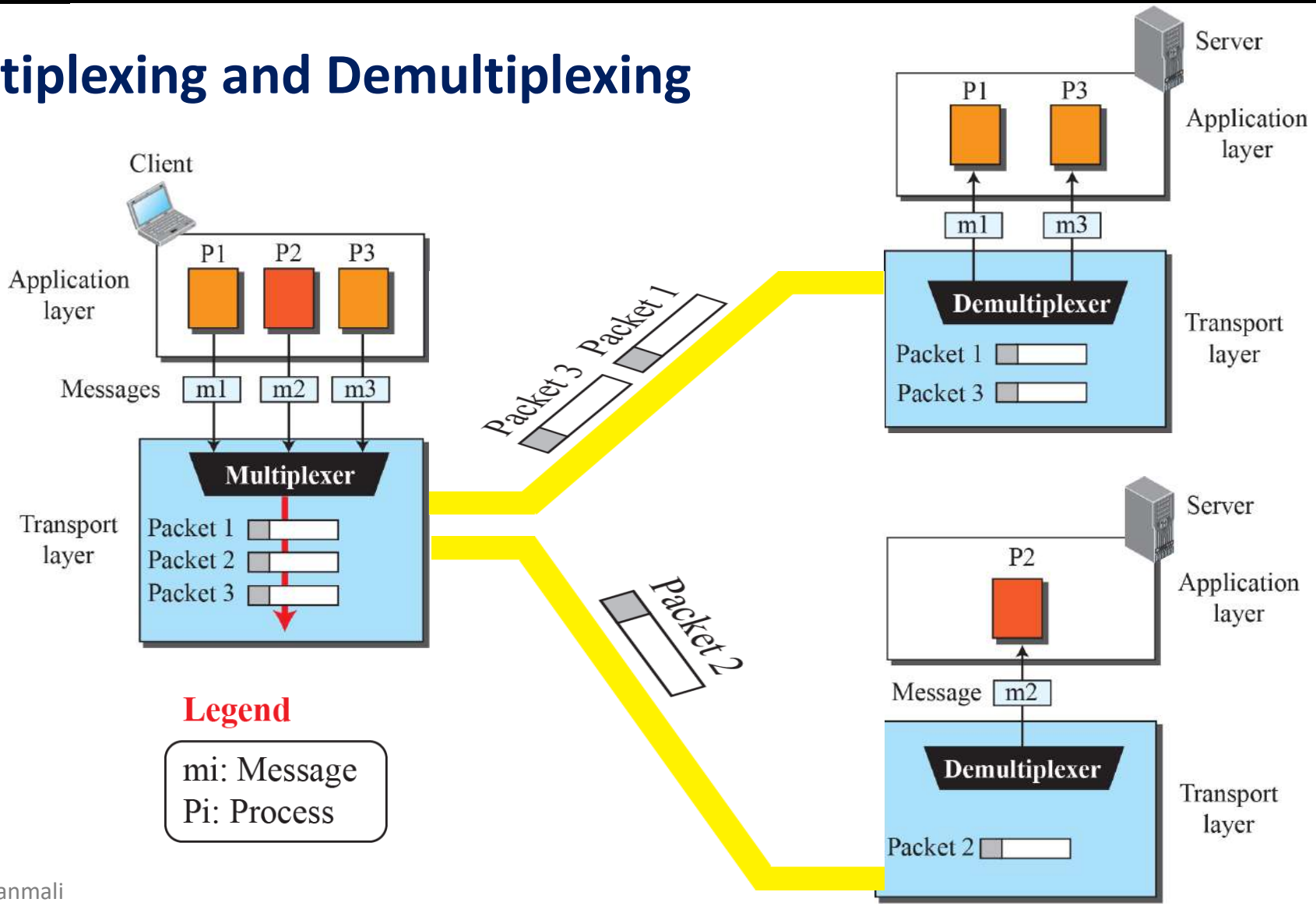
Transport Layer Services

Multiplexing and Demultiplexing

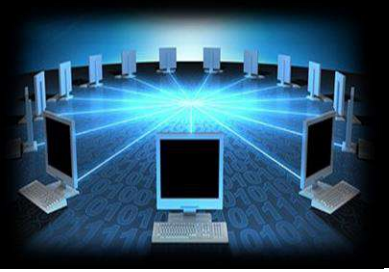
- Whenever an entity accepts items from more than one source, it is referred to as **multiplexing** (many to one); whenever an entity delivers items to more than one source, it is referred to as **demultiplexing** (one to many).
- The transport layer at the source performs multiplexing; the transport layer at the destination performs demultiplexing.
- The transport layer at the client site accepts three messages from the different processes and creates packets. It acts as a **multiplexer**.
- When the packets arrive at the server, the transport layer does the job of a **demultiplexer** and distributes the messages to different processes.

Transport Layer Services

❑ Multiplexing and Demultiplexing



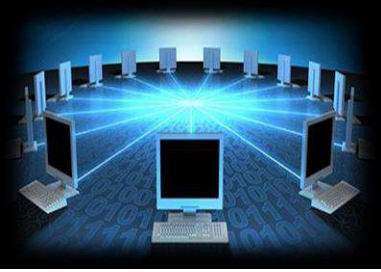
Ref: Behrouz A. Forouzan, Data Communications and Networking, 6th Edition, Mc Graw Hill education.



Transport Layer Services

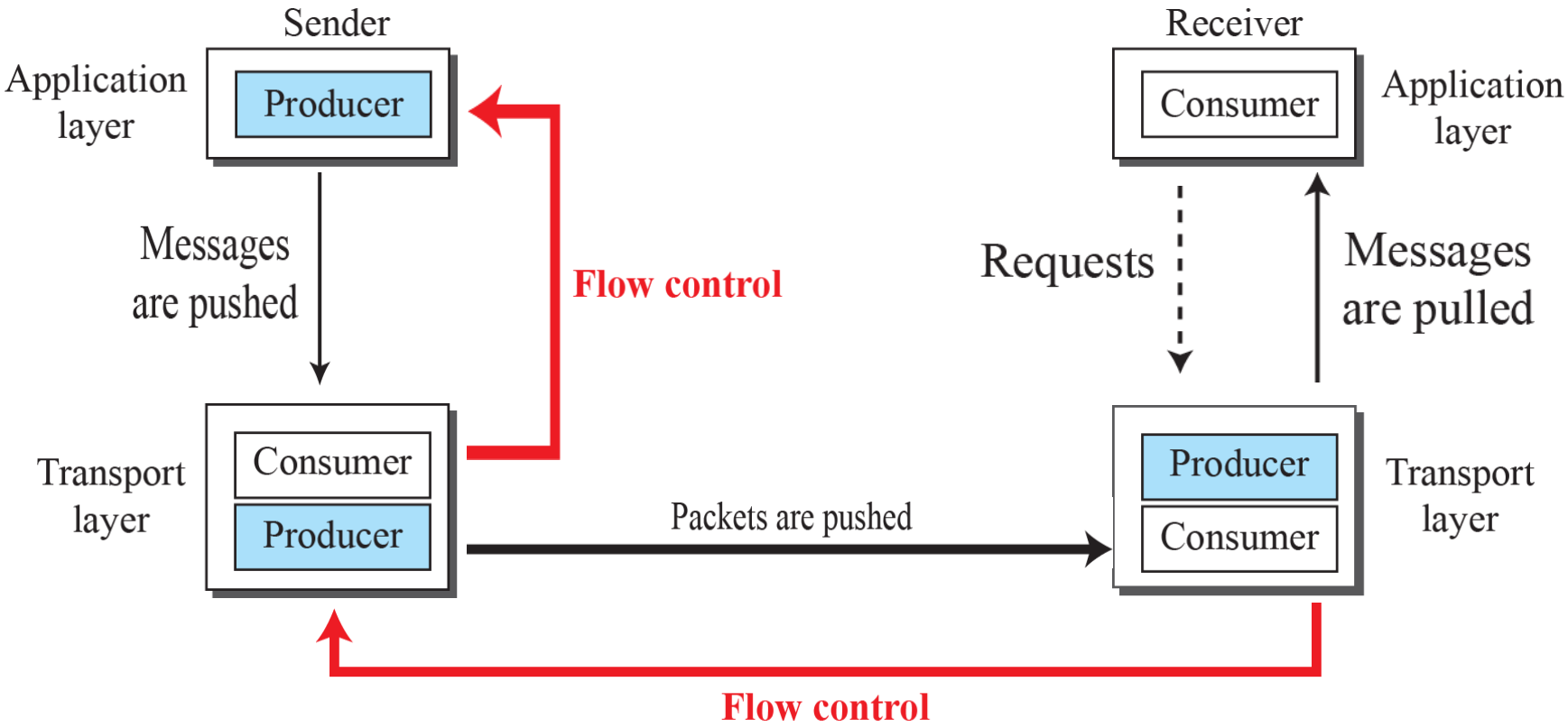
Flow Control at Transport Layer

- In communication at the transport layer, we are dealing with four entities: sender process, sender transport layer, receiver transport layer, and receiver process.
- We need at least two cases of flow control: from the sending transport layer to the sending application layer and from the receiving transport layer to the sending transport layer.
- Although flow control can be implemented in several ways, one of the solutions is normally to use two **buffers**. One at the sending transport layer and the other at the receiving transport layer.

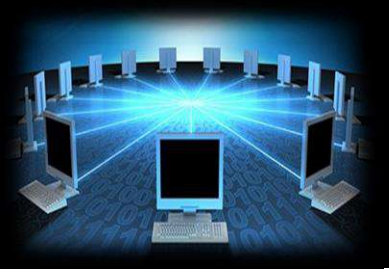


Transport Layer Services

❑ Flow Control at Transport Layer



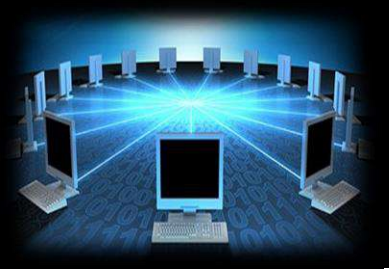
Ref: Behrouz A. Forouzan, Data Communications and Networking, 6th Edition, Mc Graw Hill education.



Transport Layer Services

Error Control at Transport Layer

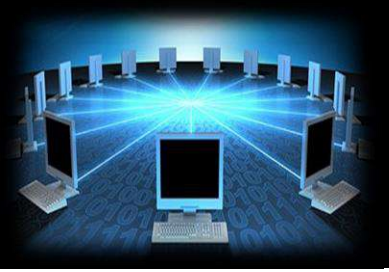
- Reliability can be achieved to add error control service to the transport layer.
- Error control at the transport layer is responsible to
  1. Detect and discard corrupted packets.
  2. Keep track of lost and discarded packets and resend them.
  3. Recognize duplicate packets and discard them.
  4. Buffer out-of-order packets until the missing packets arrive.



Transport Layer Services

Congestion Control at Transport Layer

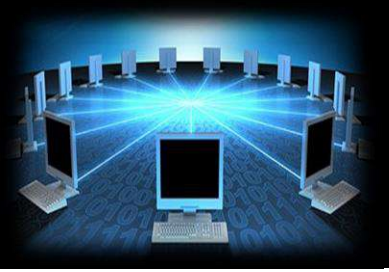
- Congestion control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion.
- In **open-loop congestion control**, policies are applied to prevent congestion before it happens.
- In open-loop mechanisms, congestion control is handled by either the source or the destination.
- **Closed-loop congestion control** mechanisms try to alleviate congestion after it happens. TCP uses this strategy to control its window size.
- Closed-loop congestion control provides feedback from the network or destination to the source.



Transport Layer Services

Connectionless and Connection-Oriented Services

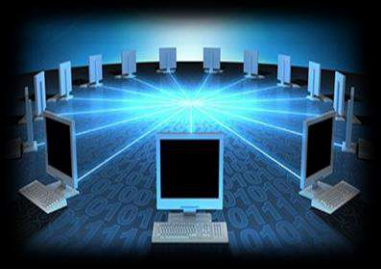
- A transport-layer protocol, like a network-layer protocol can provide two types of services: connectionless and connection-oriented.
- The nature of these services at the transport layer, however, is different from the ones at the network layer.
- At the network layer, a connectionless service may mean different paths for different datagrams belonging to the same message.
- At the transport layer, we are not concerned about the physical paths of packets.
- Connectionless service at the transport layer means independency between packets; connection-oriented means dependency.



Transport Layer Services

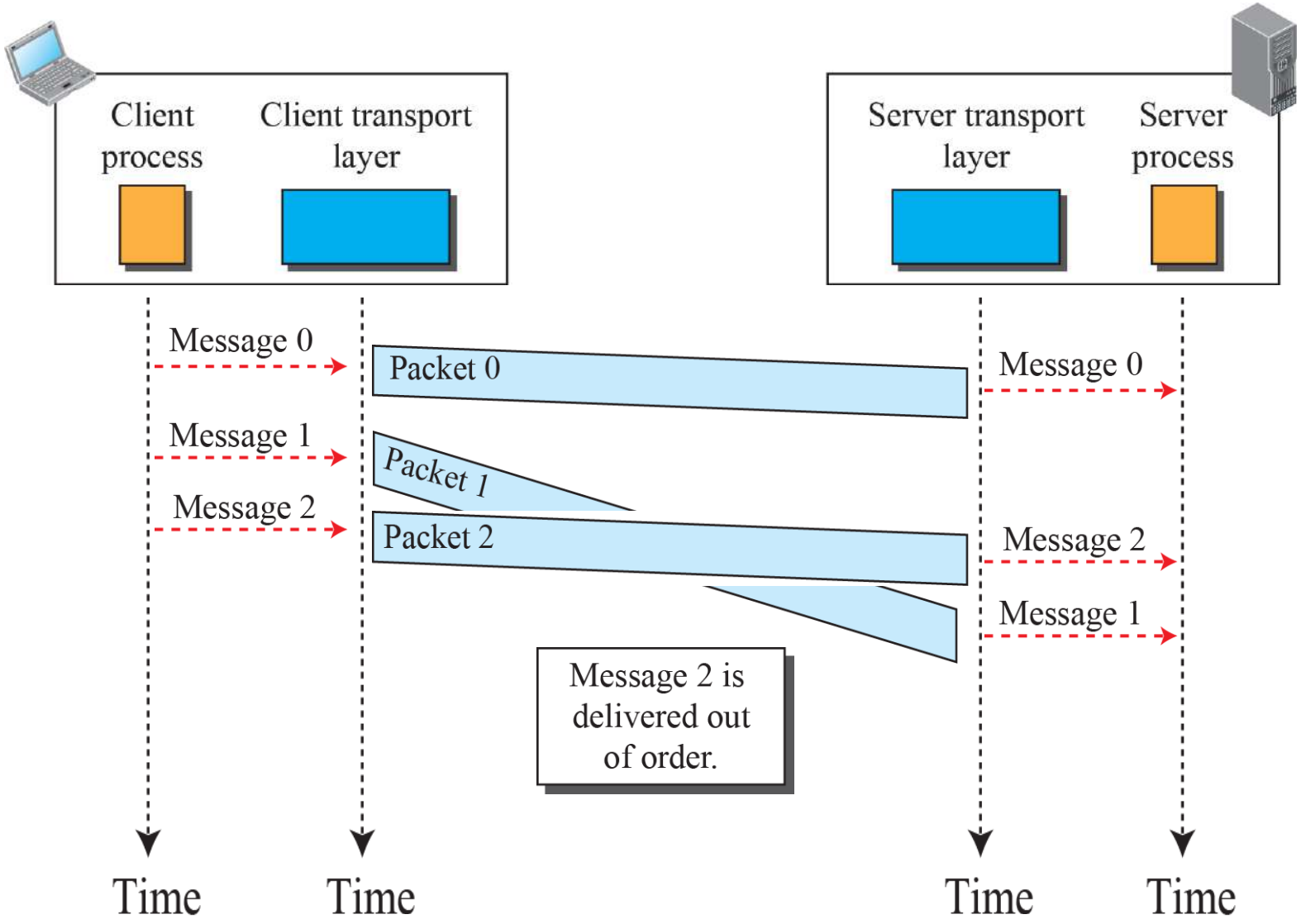
Connectionless Services

- Connectionless service do not employ flow control, error control, or congestion control.
- The source process (application program) needs to divide its message into chunks of data of the size acceptable by the transport layer and deliver them to the transport layer one by one.
- The transport layer treats each chunk as a single unit without any relation between the chunks.
- When a chunk arrives from the application layer, the transport layer encapsulates it in a packet and sends them using connectionless transport protocol.
- However, since there is no dependency between the packets at the transport layer, the packets may arrive out of order. Some chunks may even be lost.
- The above two problems arise from the fact that the two transport layers do not coordinate with each other.

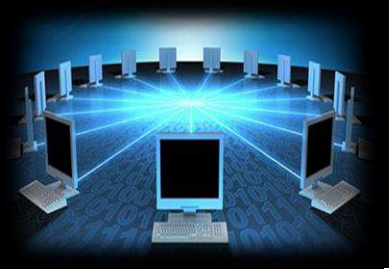


Transport Layer Services

- **Connectionless Services**



Ref: Behrouz A. Forouzan, Data Communications and Networking, 6th Edition, Mc Graw Hill education.



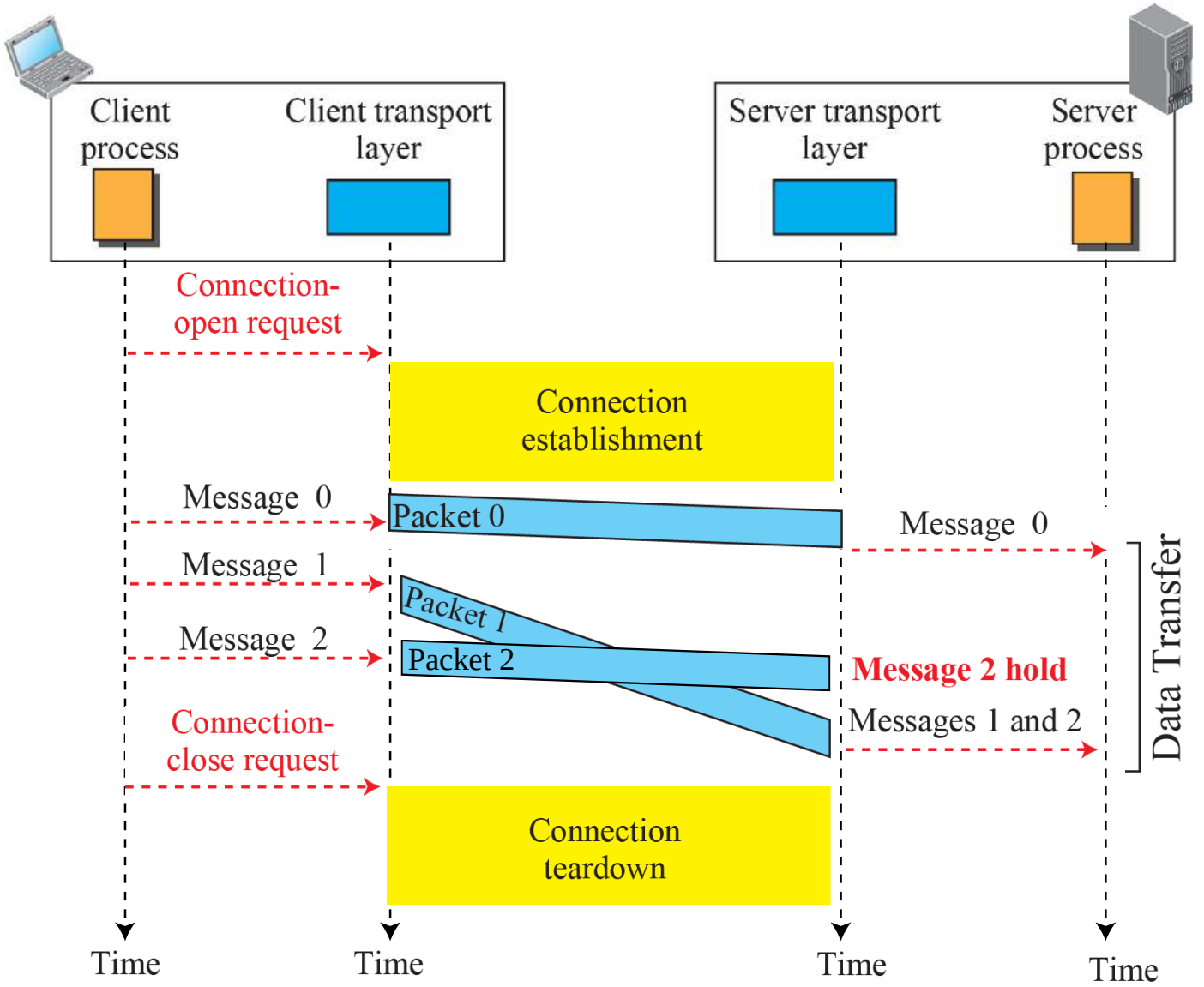
Transport Layer Services

Connection-oriented Services

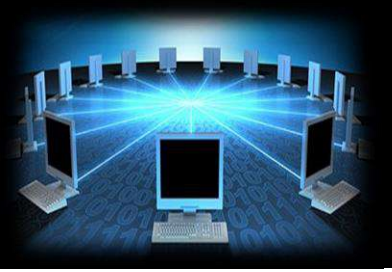
- In a connection-oriented service, the client and the server first need to establish a connection between themselves.
- The data exchange can only happen after the connection establishment.
- After data exchange, the connection needs to be teared down.
- We can implement flow control, error control, and congestion control in a connection-oriented protocol.

Transport Layer Services

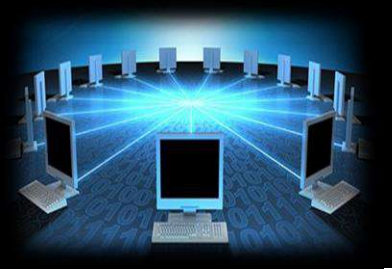
- **Connection-oriented Services**



Ref: Behrouz A. Forouzan, Data Communications and Networking, 6th Edition, Mc Graw Hill education.

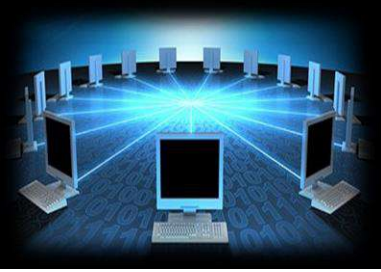


User Datagram Protocol (UDP)



User Datagram Protocol (UDP)

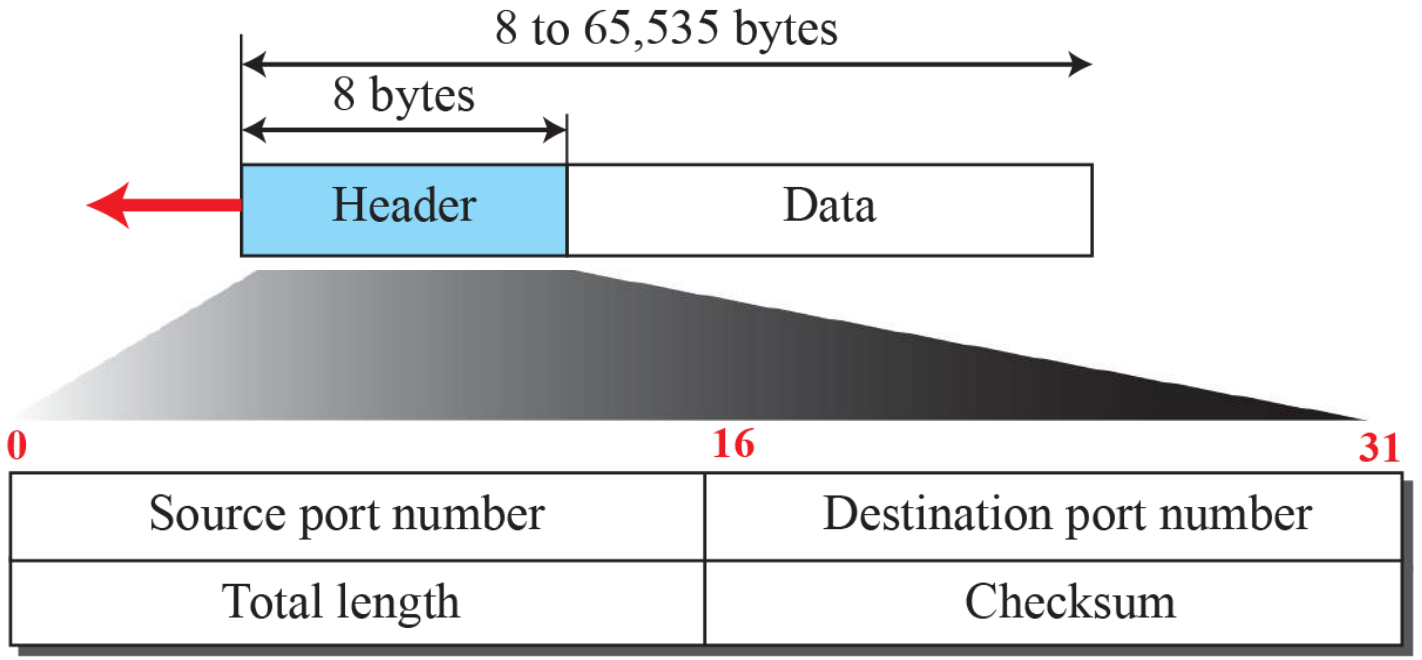
- The User Datagram Protocol (UDP) is a connectionless, unreliable transport protocol.
- It does not add anything to the services of IP except for providing process-to-process instead of host-to-host communication.
- UDP is a very simple protocol using a minimum of overhead.
- If a process wants to send a small message and does not care much about reliability, it can use UDP.
- Sending a small message by using UDP takes much less interaction between the sender and receiver than using TCP.



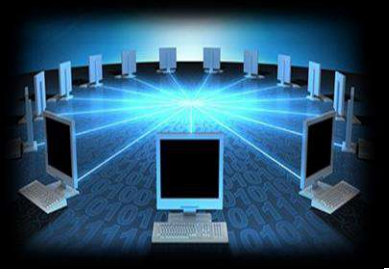
User Datagram Protocol (UDP)

User Datagram

- UDP packets, called user datagrams, have a fixed-size header of 8 bytes.



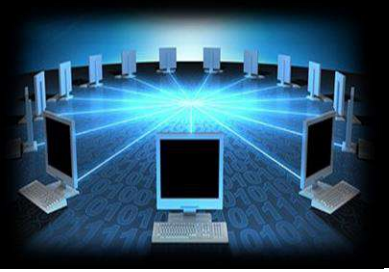
Ref: Behrouz A. Forouzan, Data Communications and Networking, 6th Edition, Mc Graw Hill education.



User Datagram Protocol (UDP)

User Datagram

- **Source port number:** This is the port number used by the process running on the source host. It is 16 bits long.
- **Destination port number:** This is the port number used by the process running on the destination host. It is also 16 bits long.
- **Length:** This is a 16-bit field that defines the total length of the user datagram, header plus data. (Note: UDP length = IP length – IP header's length)
- **Checksum:** This field is used to detect errors over the entire user datagram (header plus data).



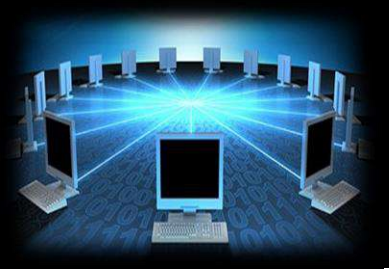
User Datagram Protocol (UDP)

User Datagram

Example: Following is the contents of a UDP header in hexadecimal format.

C B 8 4 0 0 0 D 0 0 1 C 0 0 1 C

a. What is the source port number?
b. What is the destination port number?
c. What is the total length of the user datagram?
d. What is the length of the data?
e. Is the packet directed from a client to a server or vice versa?

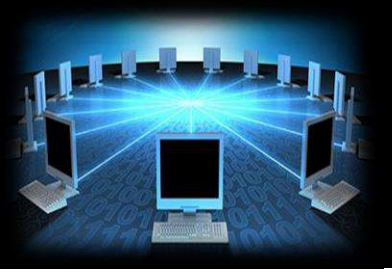


User Datagram Protocol (UDP)

User Datagram

Example:

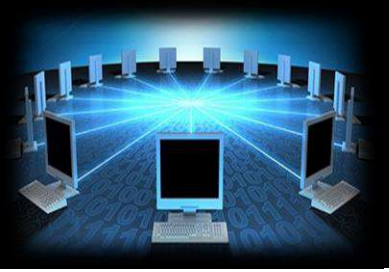
- Source port number = First four hexadecimal digits = $(\mathrm{CB84})_{16}$ = 52100
- Destination port number = Second four hexadecimal digits = $(\mathrm{000D})_{16}$ = 13
- Length of the user datagram = Third four hexadecimal digits = $(\mathrm{001C})_{16}$ = 28 bytes
- Length of the data = Length of whole packet – Length of header
  = Last four hexadecimal digits – 8
  = $(\mathrm{001C})_{16}$ – 8 = 20 bytes
- Since the destination port number is 13 (well-known port)(less than 1024), the packet is from the client to the server.



User Datagram Protocol (UDP)

UDP Services

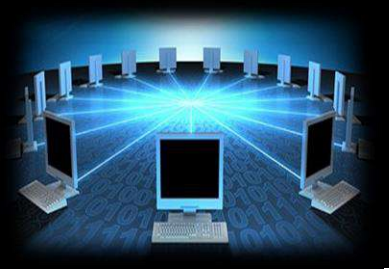
- **Process-to-Process Communication:**
  - UDP provides process-to-process communication using socket addresses, a combination of IP addresses and port number.
- **Connectionless Services:**
  - Each user datagram sent by UDP is an independent datagram.
  - There is no relationship between the different user datagrams even if they are coming from the same source process and going to the same destination program.
  - The user datagrams are not numbered.
  - There is no connection establishment and no connection termination.
  - Each datagram can travel on a different path.



User Datagram Protocol (UDP)

UDP Services

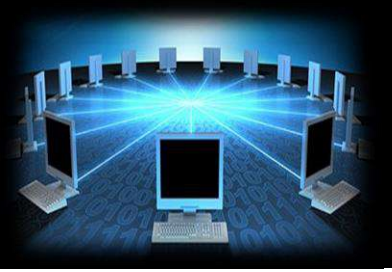
- **Flow Control:**
  - There is no flow control, and hence no window mechanism.
  - The receiver may overflow with incoming messages.
  - The lack of flow control means that the process using UDP should provide this service, if needed.
- **Error Control:**
  - There is no error control mechanism in UDP except for the checksum.
  - The sender does not know if a message has been lost or duplicated.
  - When the receiver detects an error through the checksum, the user datagram is silently discarded.
  - The process using UDP should provide this service, if needed.



User Datagram Protocol (UDP)

UDP Services

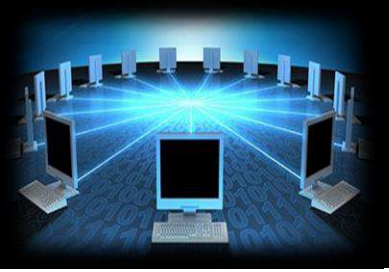
- **Congestion Control:**
  - UDP is a connectionless protocol and does not provide congestion control.
- **Encapsulation and Decapsulation:**
  - To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages.
- **Queuing:**
  - Queues are associated with ports.
  - At the client site, when a process starts, it requests a port number from the operating system.
  - Some implementations create both an incoming and outgoing queue associated with each process.
  - Other implementations create only an incoming queue associated with each process.



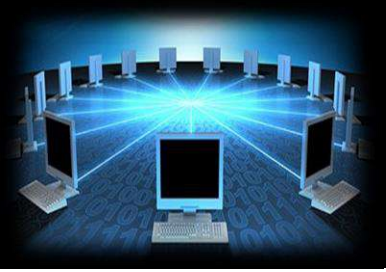
User Datagram Protocol (UDP)

UDP Applications

- UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control. It is not usually used for a process such as FTP that needs to send bulk data.
- UDP is suitable for a process with internal flow and error-control mechanisms. For example, the Trivial File Transfer Protocol (TFTP) process includes flow and error control. It can easily use UDP.
- UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.
- UDP is used for management processes such as SNMP.
- UDP is used for some route updating protocols such as Routing Information Protocol (RIP).
- UDP is normally used for real-time applications that cannot tolerate uneven delay between sections of a received message.

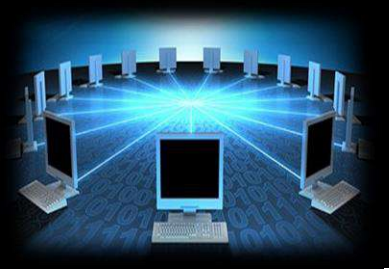


Transmission Control Protocol (TCP)



Transmission Control Protocol (TCP)

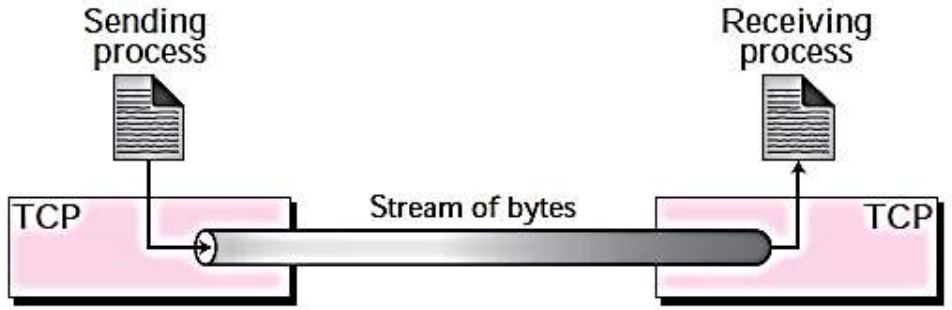
- Transmission Control Protocol (TCP) is a connection-oriented, reliable protocol.
- TCP explicitly defines connection establishment, data transfer, and connection teardown phases to provide a connection-oriented service.
- TCP uses a combination of GBN and SR protocols to provide reliability.



Transmission Control Protocol (TCP)

TCP Services

- **Process-to-Process Communication:**
  - TCP provides process-to-process communication using socket addresses, a combination of IP addresses and port number.
- **Stream Delivery Service:**
  - TCP allows the sending process to deliver data as a stream of bytes and allow the receiving process to obtain data as a stream of bytes.
  - Since the sending and receiving processes may not write or read data at the same rate, TCP needs **buffers** for storage.

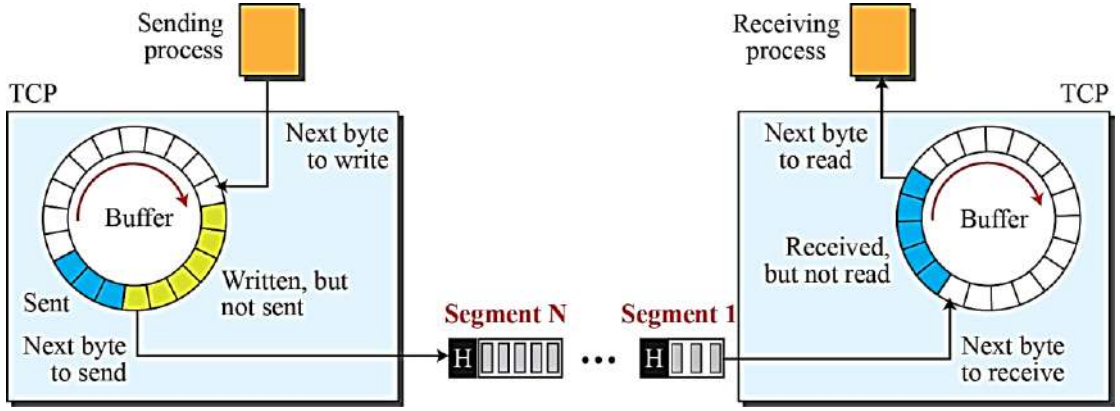


Ref: Behrouz A. Forouzan, Data Communications and Networking, 6th Edition, Mc Graw Hill education.

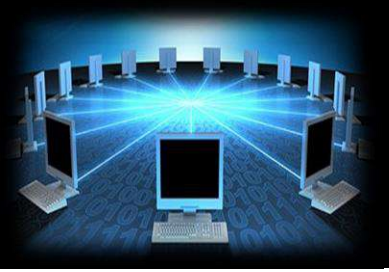
Transmission Control Protocol (TCP)

TCP Services

- **Segments:**
  - The IP layer, as a service provider for TCP, needs to send data in packets, not as a stream of bytes.
  - TCP groups a number of bytes together into a packet called a **segment**.
  - TCP adds a header to each segment and delivers the segment to the IP layer for transmission. The segments are encapsulated in an IP datagram and transmitted.
  - Segments are not necessarily of the same size.



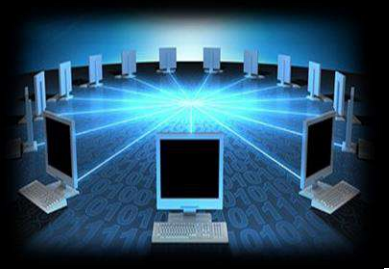
Ref: Behrouz A. Forouzan, Data Communications and Networking, 6th Edition, Mc Graw Hill education.



Transmission Control Protocol (TCP)

TCP Services

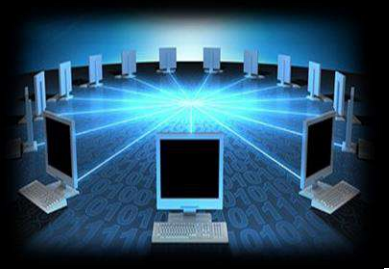
- **Full-Duplex Communication:**
  - TCP offers full-duplex service, where data can flow in both directions at the same time.
- **Multiplexing and Demultiplexing:**
  - Like UDP, TCP performs multiplexing at the sender and demultiplexing at the receiver.
- **Connection-Oriented Service:**
  - TCP, unlike UDP, is a connection-oriented protocol.
  - When a process at site A wants to send to and receive data from another process at site B, the following three phases occur:
    1. The two TCPs establish a virtual connection between them.
    2. Data are exchanged in both directions.
    3. The connection is terminated.



Transmission Control Protocol (TCP)

TCP Services

- **Reliable Service:**
  - TCP is a reliable transport protocol.
  - It uses an acknowledgment mechanism to check the safe and sound arrival of data.
- **Flow Control:**
  - TCP, unlike UDP, provides **byte-oriented flow control**.
  - The sending TCP controls how much data can be accepted from the sending process; the receiving TCP controls how much data can be sent by the sending TCP.



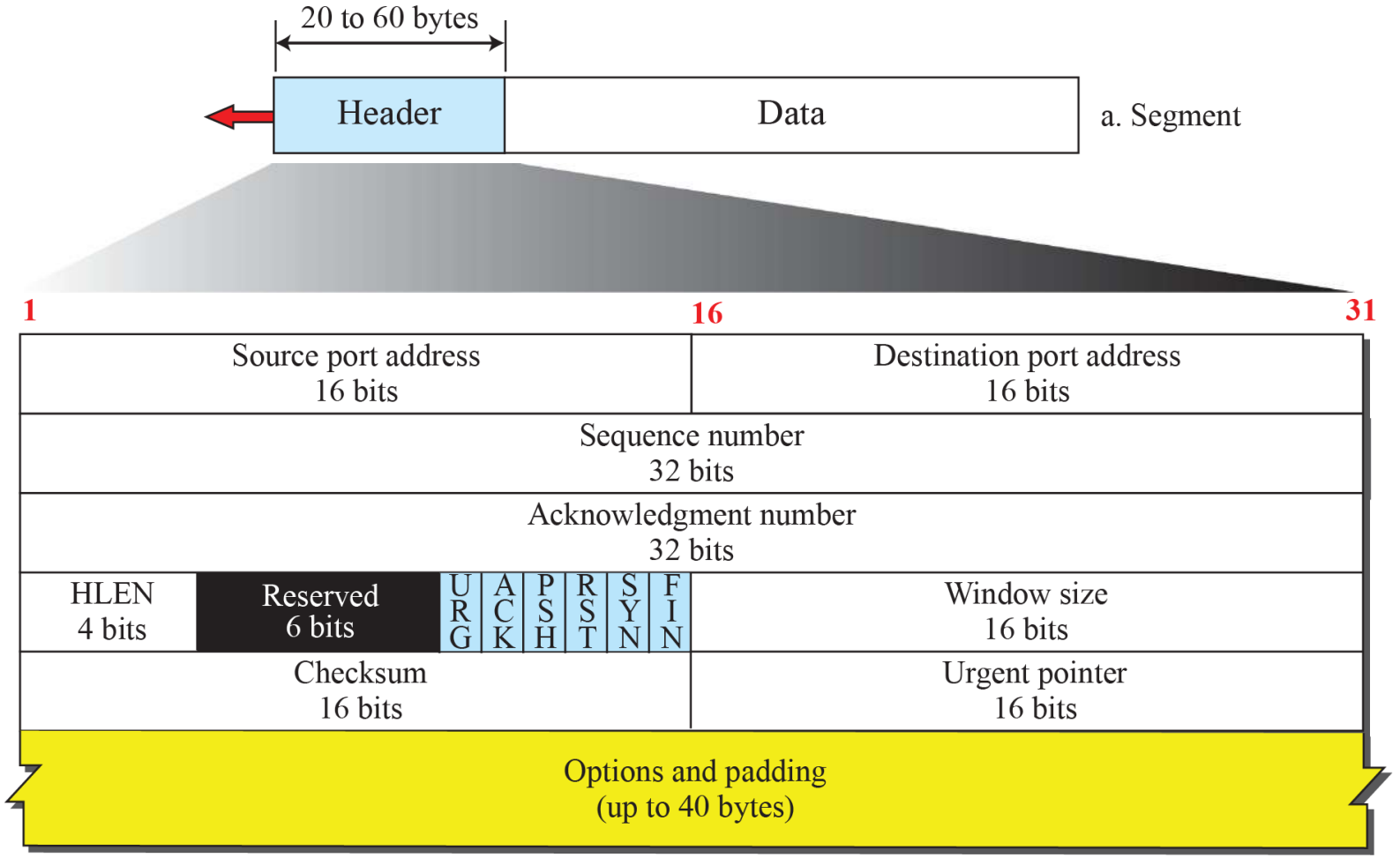
Transmission Control Protocol (TCP)

TCP Segment Format

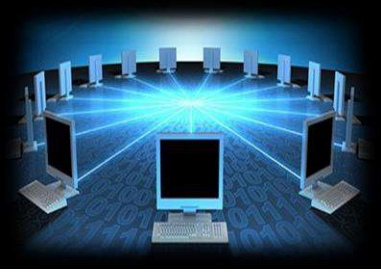
- A packet in TCP is called a **segment**.
- The segment consists of a header of 20 to 60 bytes, followed by data from the application program.
- The header is 20 bytes if there are no options and up to 60 bytes if it contains options.

Transmission Control Protocol (TCP)

TCP Segment Format



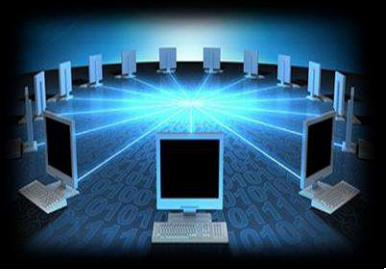
Ref: Behrouz A. Forouzan, Data Communications and Networking, 6th Edition, Mc Graw Hill education.



Transmission Control Protocol (TCP)

TCP Segment Format

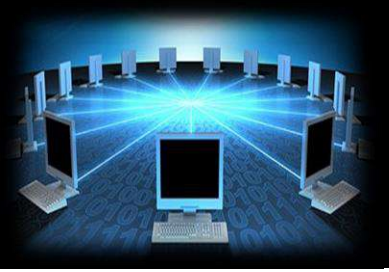
- **Source port address:**
  - This is a 16-bit field that defines the port number of the application program in the host that is sending the segment.
- **Destination port address:**
  - This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment.
- **Sequence number:**
  - This 32-bit field defines the number assigned to the first byte of data contained in this segment.
  - The sequence number tells the destination which byte in this sequence is the first byte in the segment.



Transmission Control Protocol (TCP)

TCP Segment Format

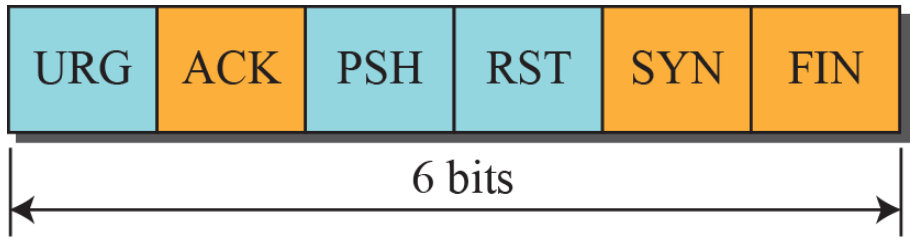
- **Acknowledgment number:**
  - This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party.
- **Header length:**
  - This 4-bit field indicates the number of 4-byte words in the TCP header.
  - The length of the header can be between 20 and 60 bytes.
  - Therefore, the value of this field is always between 5 (5 × 4 = 20) and 15 (15 × 4 = 60).
- **Reserved:**
  - This is a 6-bit field reserved for future use.



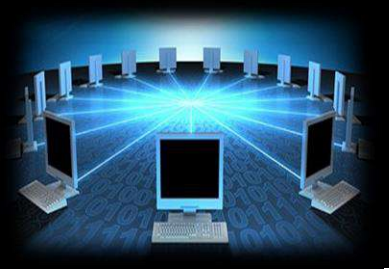
Transmission Control Protocol (TCP)

TCP Segment Format

- **Control:**
  - This field defines 6 different control bits or flags.
  - One or more of these bits can be set at a time.
  - These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.



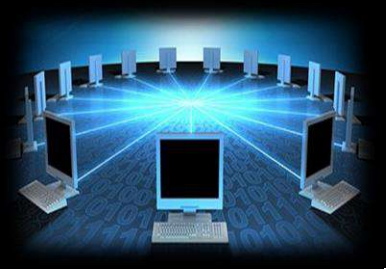
URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection



Transmission Control Protocol (TCP)

TCP Segment Format

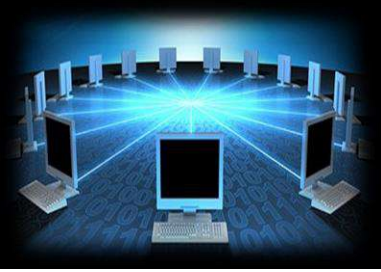
- **Window size:**
  - This field defines the window size of the sending TCP in bytes.
  - This value is normally referred to as the receiving window (rwnd) and is determined by the receiver.
- **Checksum:**
  - This 16-bit field contains the checksum for the header.
  - The use of the checksum in TCP is mandatory (It is optional in UDP).
- **Urgent pointer:**
  - This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data.
  - It defines a value that must be added to the sequence number to obtain the number of the last urgent byte in the data section of the segment.



Transmission Control Protocol (TCP)

TCP Segment Format

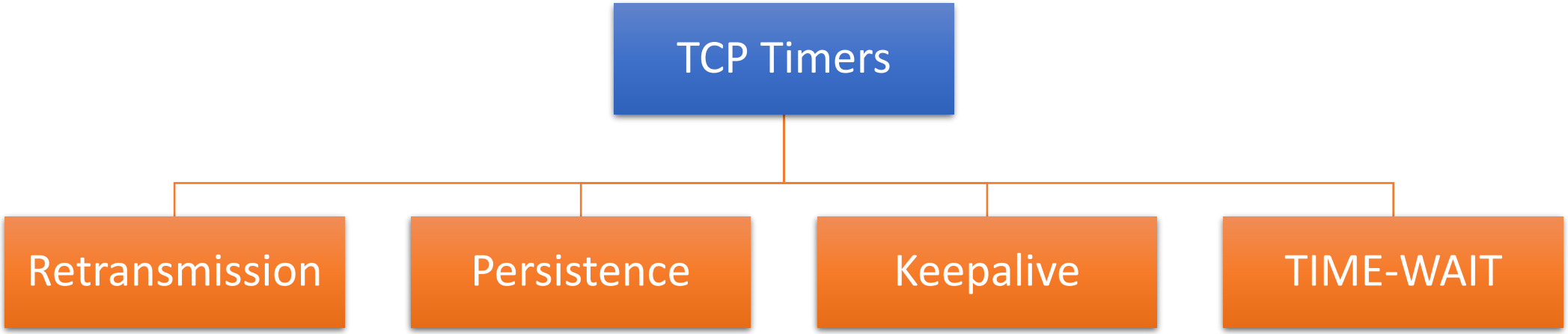
- **Options:**
  - The TCP header can have up to 40 bytes of optional information.
  - Options convey additional information to the destination or align other options.
  - There are two categories of options: 1-byte options and multiple-byte options.
  - The first category contains two types of options: end of option list and no operation.
  - The second category, in most implementations, contains five types of options: maximum segment size, window scale factor, timestamp, SACK-permitted, and SACK.

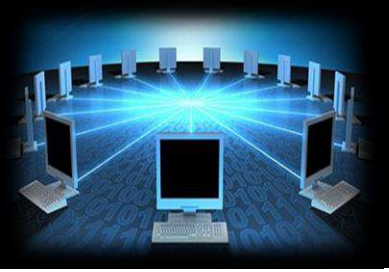


Transmission Control Protocol (TCP)

TCP Timers

- To perform its operation smoothly, most TCP implementations use at least four timers.

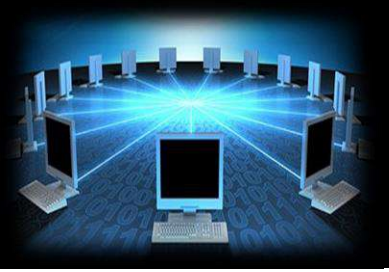




Transmission Control Protocol (TCP)

TCP Timers

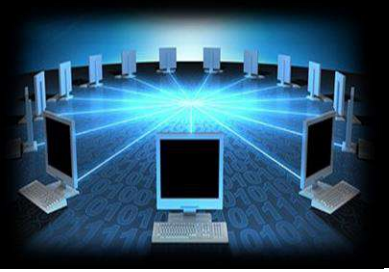
- **Retransmission Timer:**
  - To retransmit lost segments, TCP employs one retransmission timer (for the whole connection period) that handles the retransmission time-out (RTO), the waiting time for an acknowledgment of a segment.
  - Following rules are used for the retransmission timer:
    1. When TCP sends the segment in front of the sending queue, it starts the timer.
    2. When the timer expires, TCP resends the first segment in front of the queue and restarts the timer.
    3. When a segment (or segments) are cumulatively acknowledged, the segment (or segments) are purged from the queue.
    4. If the queue is empty, TCP stops the timer; otherwise, TCP restarts the timer.



Transmission Control Protocol (TCP)

TCP Timers

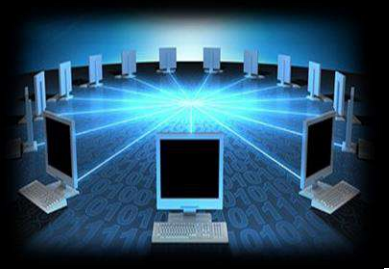
- **Persistence Timer:**
  - To deal with a zero-window-size deadlock situation, TCP uses a persistence timer.
  - When the sending TCP receives an acknowledgment with a window size of zero, it starts a persistence timer.
  - When the persistence timer goes off, the sending TCP sends a special segment called a **probe**.
  - This segment contains only 1 byte of new data.
  - It has a sequence number, but its sequence number is never acknowledged.
  - The probe causes the receiving TCP to resend the acknowledgment which was lost.



Transmission Control Protocol (TCP)

TCP Timers

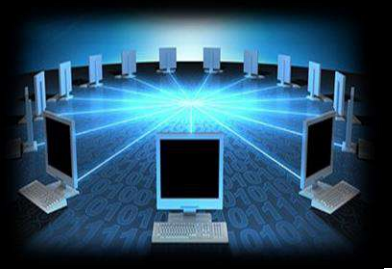
- **Keepalive Timer:**
  - A keepalive timer is used in some implementations to prevent a long idle connection between two TCPs.
  - Suppose that a client opens a TCP connection to a server, transfers some data, and becomes silent.
  - To remedy this situation, most implementations equip a server with a keepalive timer.
  - Each time the server hears from a client, it resets this timer.
  - The time-out is usually 2 hours. If the server does not hear from the client after 2 hours, it sends a probe segment.
  - If there is no response after 10 probes, each of which is 75 s apart, it assumes that the client is down and terminates the connection.



Transmission Control Protocol (TCP)

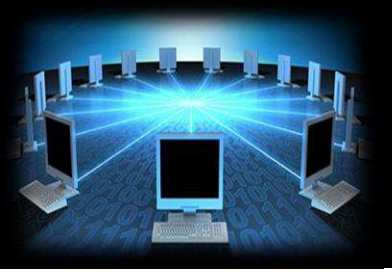
TCP Timers

- **TIME-WAIT Timer:**
  - The TIME-WAIT (2MSL) timer is used during connection termination.
  - The timer starts after sending the last ACK for 2nd FIN and closing the connection.
  - After a TCP connection is closed, it is possible for datagrams that are still making their way through the network to attempt to access the closed port.
  - The **quiet timer** is intended to prevent the just-closed port from reopening again quickly and receiving these last datagrams.
  - The quiet timer is usually set to twice the maximum segment lifetime (the same value as the Time-To-Live field in an IP header), ensuring that all segments still heading for the port have been discarded.



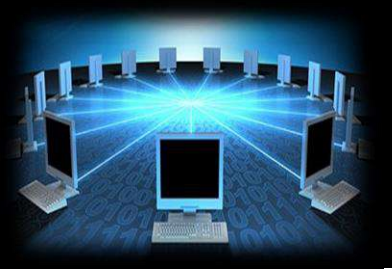
TCP v/s UDP

Basis	Transmission control protocol (TCP)	User datagram protocol (UDP)
Type of Service	TCP is a connection-oriented protocol. Connection-orientation means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting the data.	UDP is the Datagram-oriented protocol. This is because there is no overhead for opening a connection, maintaining a connection, and terminating a connection. UDP is efficient for broadcast and multicast types of network transmission.
Reliability	TCP is reliable as it guarantees the delivery of data to the destination router.	The delivery of data to the destination cannot be guaranteed in UDP.
Error checking mechanism	TCP provides extensive error-checking mechanisms. It is because it provides flow control and acknowledgment of data.	UDP has only the basic error checking mechanism using checksums.
Acknowledgment	An acknowledgment segment is present.	No acknowledgment segment.



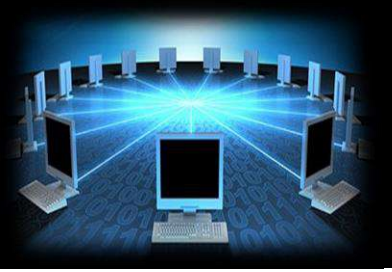
TCP v/s UDP

Basis	Transmission control protocol (TCP)	User datagram protocol (UDP)
Sequence	Sequencing of data is a feature of Transmission Control Protocol (TCP). this means that packets arrive in order at the receiver.	There is no sequencing of data in UDP. If the order is required, it has to be managed by the application layer.
Speed	TCP is comparatively slower than UDP.	UDP is faster, simpler, and more efficient than TCP.
Retransmission	Retransmission of lost packets is possible in TCP, but not in UDP.	There is no retransmission of lost packets in the User Datagram Protocol (UDP).
Header Length	TCP has a (20-60) bytes variable length header.	UDP has an 8 bytes fixed-length header.
Weight	TCP is heavy-weight.	UDP is lightweight.

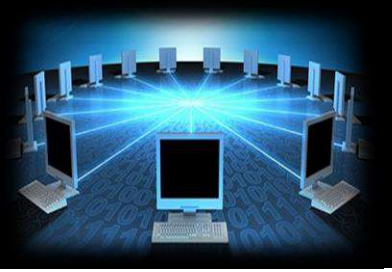


TCP v/s UDP

Basis	Transmission control protocol (TCP)	User datagram protocol (UDP)
Handshaking Techniques	Uses handshakes such as SYN, ACK, SYN-ACK	It's a connectionless protocol i.e. No handshake
Broadcasting	TCP doesn't support Broadcasting.	UDP supports Broadcasting.
Protocols	TCP is used by HTTP, HTTPs, FTP, SMTP and Telnet.	UDP is used by DNS, DHCP, TFTP, SNMP, RIP, and VoIP.
Stream Type	TCP connection is a byte stream.	UDP connection is message stream.
Overhead	Low but higher than UDP.	Very low.



Session Layer Design Issues

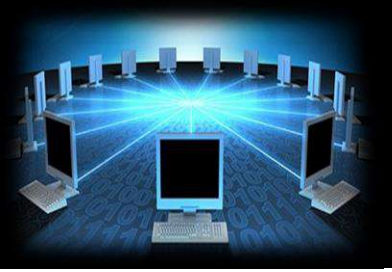


Session Layer Design Issues

- Session layer is the fifth layer in OSI model and is responsible for organizing, handling, synchronizing and terminating sessions between application processes for end users.

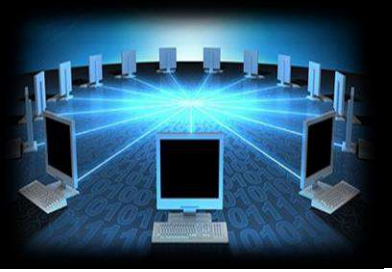
❑ Functions of Session Layer:

- **Dialog Control –**
  - Session layer allows two systems to enter into a dialog exchange mechanism which can either be full or half-duplex.
- **Managing Tokens –**
  - Session Layer manages token which prevents collisions.
- **Synchronization –**
  - Checkpoints (synchronization points) are the midway marks that are added after a particular interval during stream of data transfer.
  - The Session layer permits process to add these checkpoints.



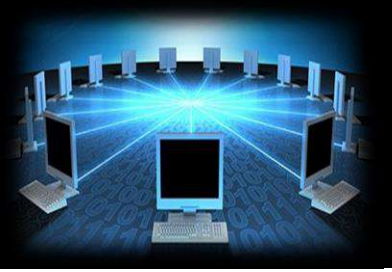
Session Layer Design Issues

- There are some design issues associated with the session layer:
  1. To permit machines to build-up sessions between them in a seamless manner.
  2. Offer improved types of services to the client
  3. To manage dialog control.
  4. To offer types of services such as token management and synchronization.



Remote Procedure Call (RPC) Protocol

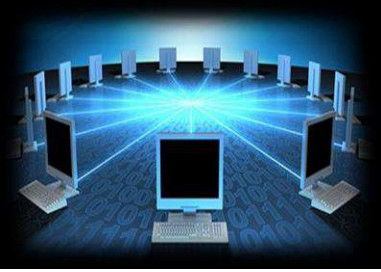
- It is a software communication protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network's details.
- It is also known as a **subroutine call** or a **function call**.
- RPC uses the **client-server** model.
- The requesting program is a client, and the service-providing program is the server.
- Like a local procedure call, an RPC is a synchronous operation requiring the requesting program to be suspended until the results of the remote procedure are returned.
- However, the use of lightweight processes or threads that share the same address space enables multiple RPCs to be performed concurrently.



Remote Procedure Call (RPC) Protocol

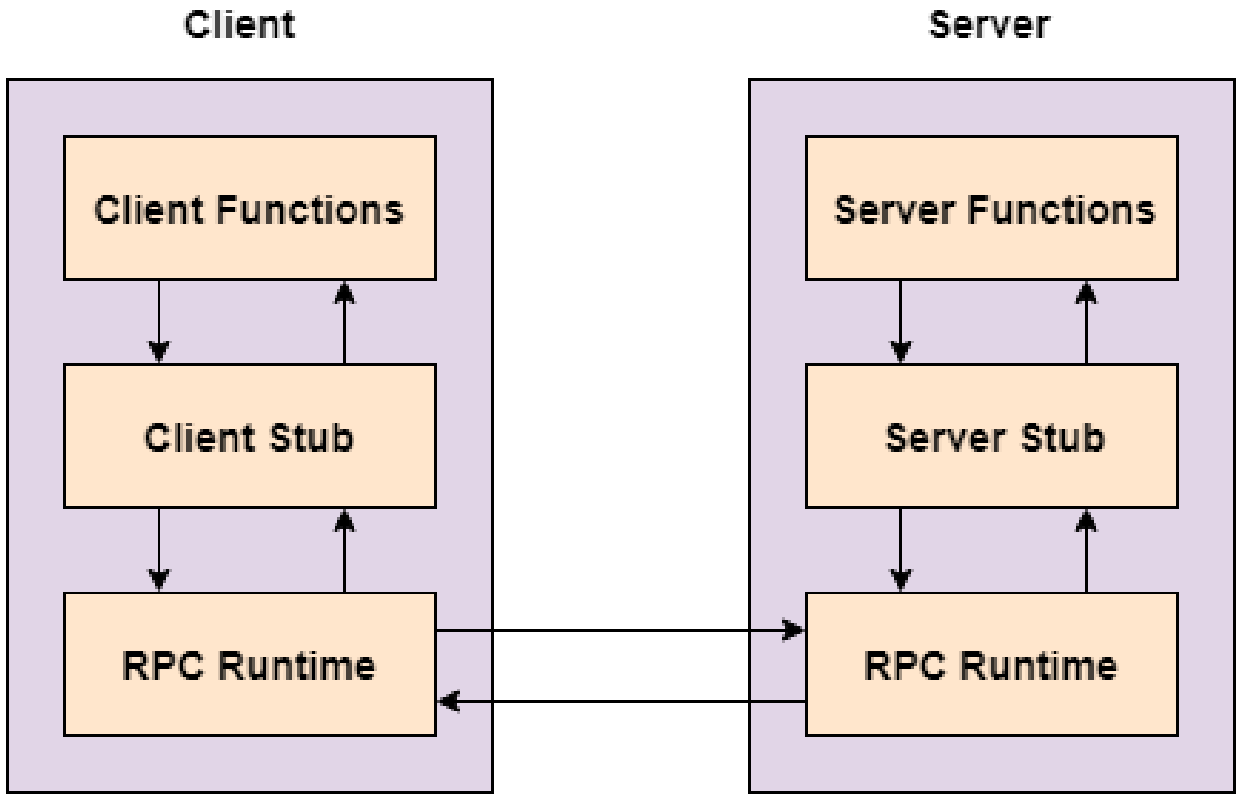
Sequence of events in RPC

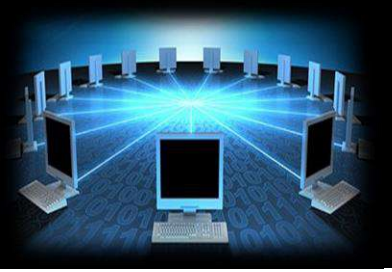
1. The client calls the client stub. The call is a local procedure call with parameters pushed onto the stack in the normal way.
2. The client stub packs the procedure parameters into a message and makes a system call to send the message. The packing of the procedure parameters is called **marshalling**.
3. The client's local OS sends the message from the client machine to the remote server machine.
4. The server OS passes the incoming packets to the server stub.
5. The server stub unpacks the parameters -- called **unmarshalling** -- from the message.
6. When the server procedure is finished, it returns to the server stub, which marshals the return values into a message. The server stub then hands the message to the transport layer.
7. The transport layer sends the resulting message back to the client transport layer, which hands the message back to the client stub.
8. The client stub unmarshalls the return parameters, and execution returns to the caller.



Remote Procedure Call (RPC) Protocol

❑ Sequence of events in RPC

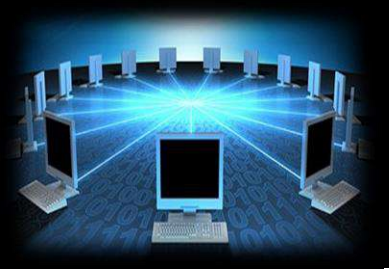




Remote Procedure Call (RPC) Protocol

Advantages of RPC

- Helps clients communicate with servers via the traditional use of procedure calls in high-level languages.
- Can be used in a distributed environment, as well as the local environment.
- Supports process-oriented and thread-oriented models.
- Hides the internal message-passing mechanism from the user.
- Requires only minimal effort to rewrite and redevelop the code.
- Provides abstraction, i.e., the message-passing nature of network communication is hidden from the user.
- Omits many of the protocol layers to improve performance.



Remote Procedure Call (RPC) Protocol

Disadvantages of RPC

- The client and server use different execution environments for their respective routines, and the use of resources (e.g., files) is also more complex. Consequently, RPC systems aren't always suited for transferring large amounts of data.
- RPC is highly vulnerable to failure because it involves a communication system, another machine and another process.
- There is no uniform standard for RPC; it can be implemented in a variety of ways.
- RPC is only interaction-based, and as such, it doesn't offer any flexibility when it comes to hardware architecture.