Unit-IV

Transport Layer

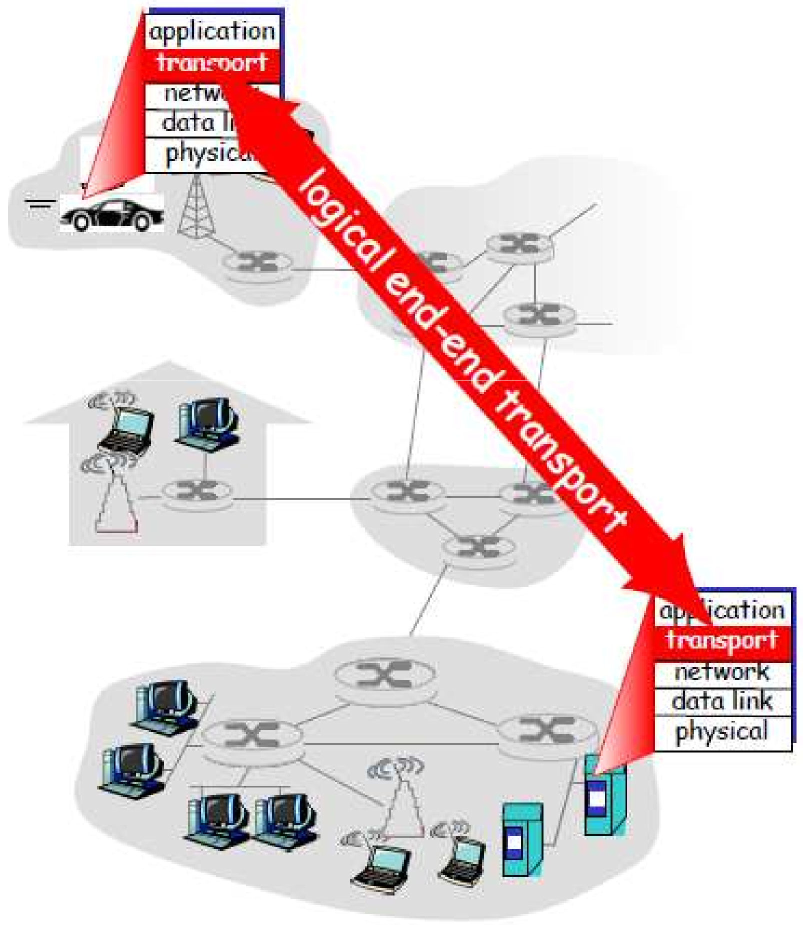
Introduction: Transport Layer

- Located between the application layer and the network layer in the TCP/IP protocol suite.
- *Provides services to the application layer and receives services from the network layer*.
- Acts as a liaison between a client program and a server program, a process-to-process connection.
- An end-to-end logical vehicle for transferring data from one point to another in the Internet.

Introduction: Transport Layer (2/2)

- 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 datagrams 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.
- Provides the abstractions that applications need to use the network.
- Without transport layer, the whole concept of layered protocols would make little sense.

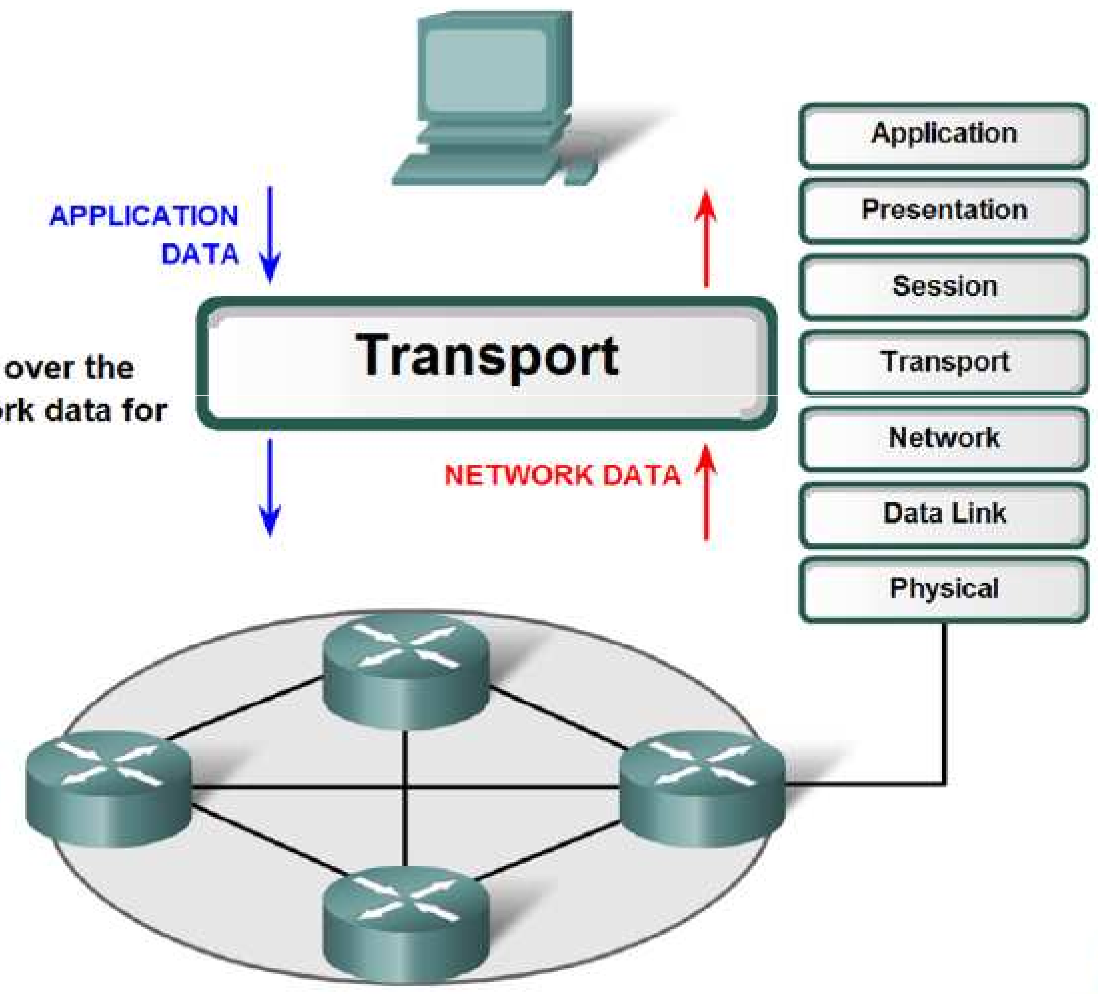
Overview (1/2)



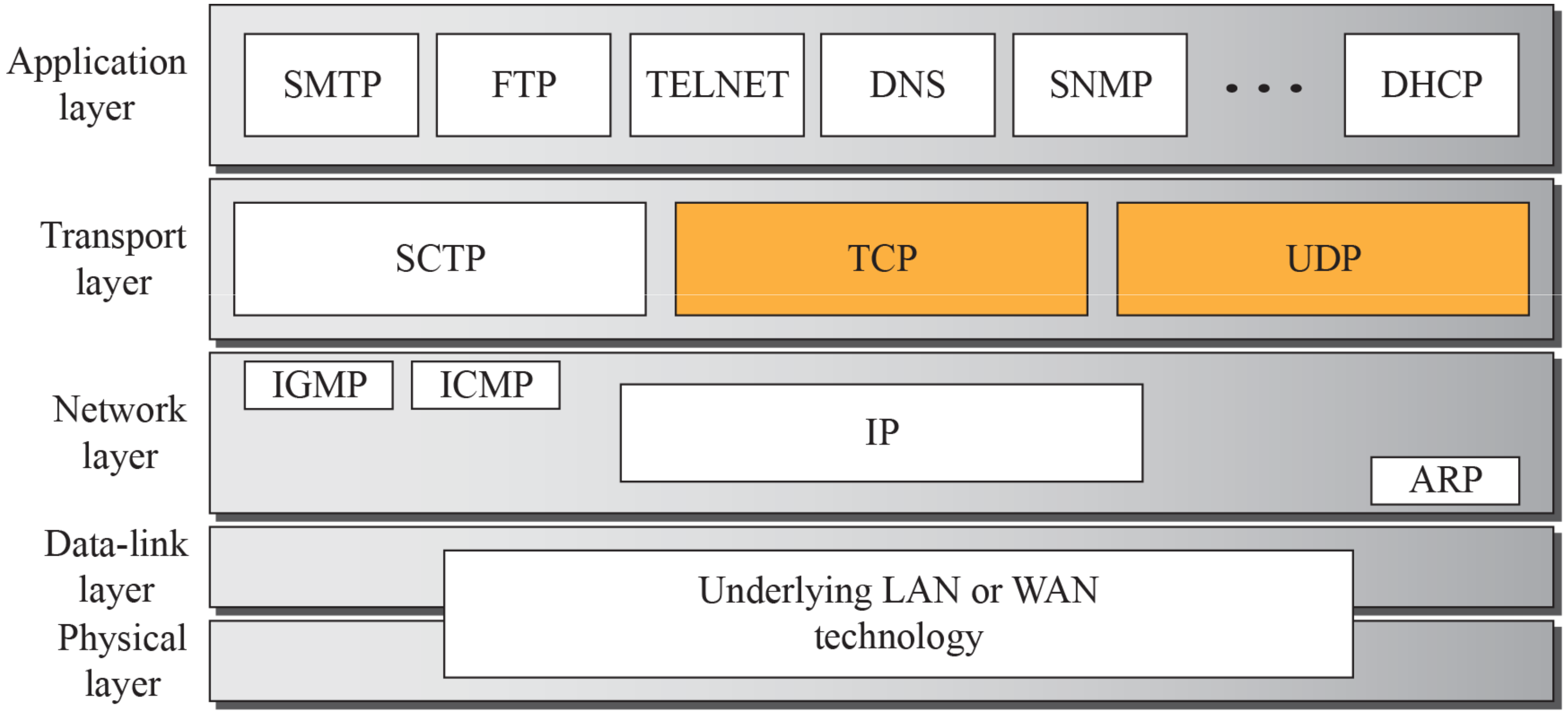
- Provides *logical communication* between app processes running on different hosts.
- Transport protocols run in end systems
  - **Sender side:** breaks application messages into segments, passes to network layer.
  - **Receiver side:** reassembles segments into messages, passes to application layer.
- *More than one transport protocol available to applications*.
  - Internet: TCP and UDP

Overview (2/2)

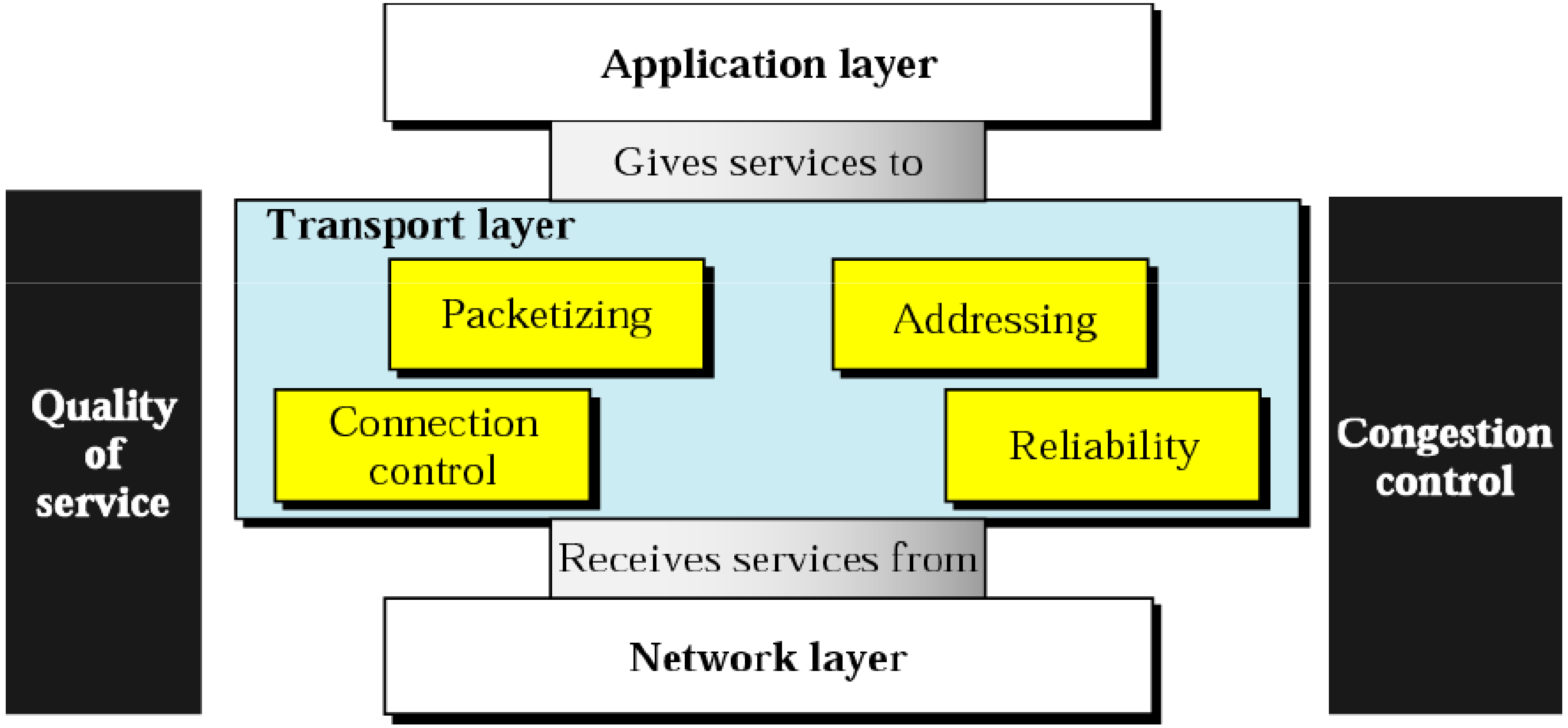
The Transport layer prepares application data for transport over the network and processes network data for use by applications.



Position of Transport Layer Protocols in the TCP/IP Protocol Suite



Position of Transport Layer Protocols in the TCP/IP Protocol Suite Contd…



Transport Layer Vs Network Layer

- *Network Layer:*
  - *Logical communication between hosts.*
- *Transport Layer:*
  - *Logical communication between processes.*
  - Relies on, enhances, network layer services.

Relationship between Transport and Network Layers

Household Analogy (1/6)

- Consider two houses, one on the East Coast and the other on the West Coast, with each house being home to a dozen kids.
- The kids in the East Coast household are cousins with the kids in the West Coast households.
- The kids in the two households love to write each other – each kid writes each cousin every week, with each letter delivered by the traditional postal service in a separate envelope.
- Thus, each household sends 144 letters to the other household every week. (These kids would save a lot of money if they had e-mail!).
- In each of the households there is one kid – Alice in the West Coast house and Bob in the East Coast house – responsible for mail collection and mail distribution.
- Each week Alice visits all her brothers and sisters, collects the mail, and gives the mail to a postal-service mail person who makes daily visits to the house.
- When letters arrive to the West Coast house, Alice also has the job of distributing the mail to her brothers and sisters. Bob has a similar job on the East coast.

Household Analogy (2/6)

- In this example,
  - The postal service provides logical communication between the two houses
    - the postal service moves mail from house to house, not from person to person.
  - On the other hand, Alice and Bob provide logical communication between the cousins
    - – Alice and Bob pick up mail from and deliver mail to, their brothers and sisters.
  - Note that, from the cousins' perspective, Alice and Bob *are* the mail service, even though Alice and Bob are only a part (the end system part) of the end-to-end delivery process.

Household Analogy (3/6)

- *Hosts (also called end systems)* = *Houses*
- *Processes* = *Cousins*
- *Application messages* = *Letters in envelope*
- *Network layer protocol* = *Postal service* (including mail persons)
- *Transport layer protocol* = *Alice and Bob*

Household Analogy (4/6)
Transport Layer Protocols Live in the End Systems

- Continuing with this analogy, observe that Alice and Bob do all their work within their respective homes; they are not involved, for example, in sorting mail in any intermediate mail center or in moving mail from one mail center to another.
- Similarly, **transport layer protocols live in the end systems**.
- Within an end system, a transport protocol moves messages from application processes to the network edge (i.e., the network layer) and vice versa; but it doesn't have any say about how the messages are moved within the network core.
- *Intermediate routers neither act on, nor recognize, any information that the transport layer may have appended to the application messages*.

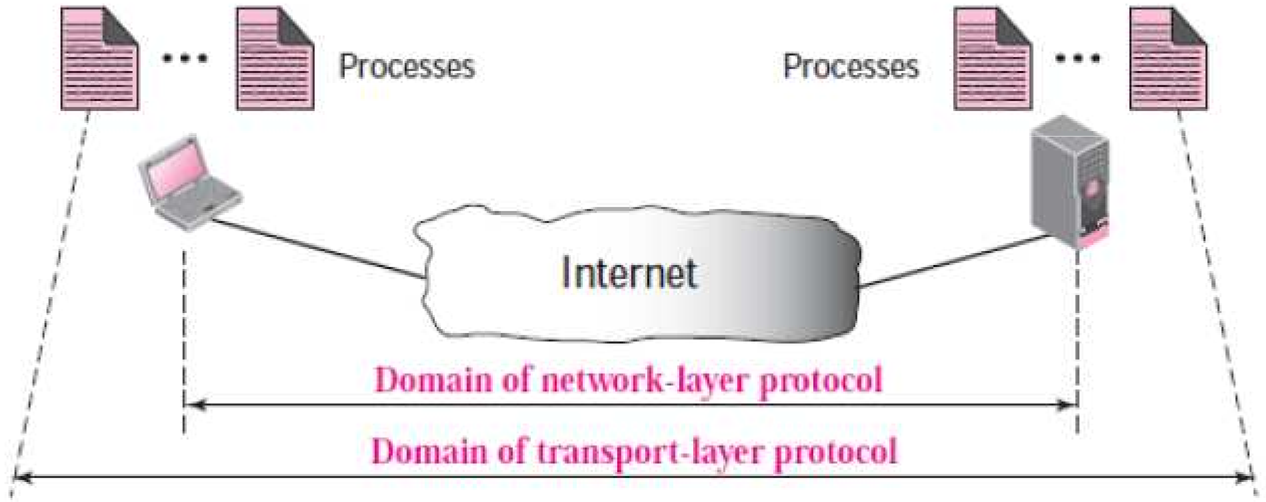
Household Analogy (5/6)

- Suppose now that when Alice and Bob go on vacation, another cousin pair – say, Susan and Harvey – substitute for them and provide the household-internal collection and delivery of mail.
- Unfortunately for the two families, Susan and Harvey do not do the collection and delivery in exactly the same way as Alice and Bob.
- Being younger kids, Susan and Harvey pick up and drop off the mail less frequently and occasionally lose letters (which are sometimes chewed up by the family dog).
- Thus, the cousin-pair Susan and Harvey do not provide the same set of services (i.e., the same service model) as Alice and Bob.

Household Analogy (6/6)

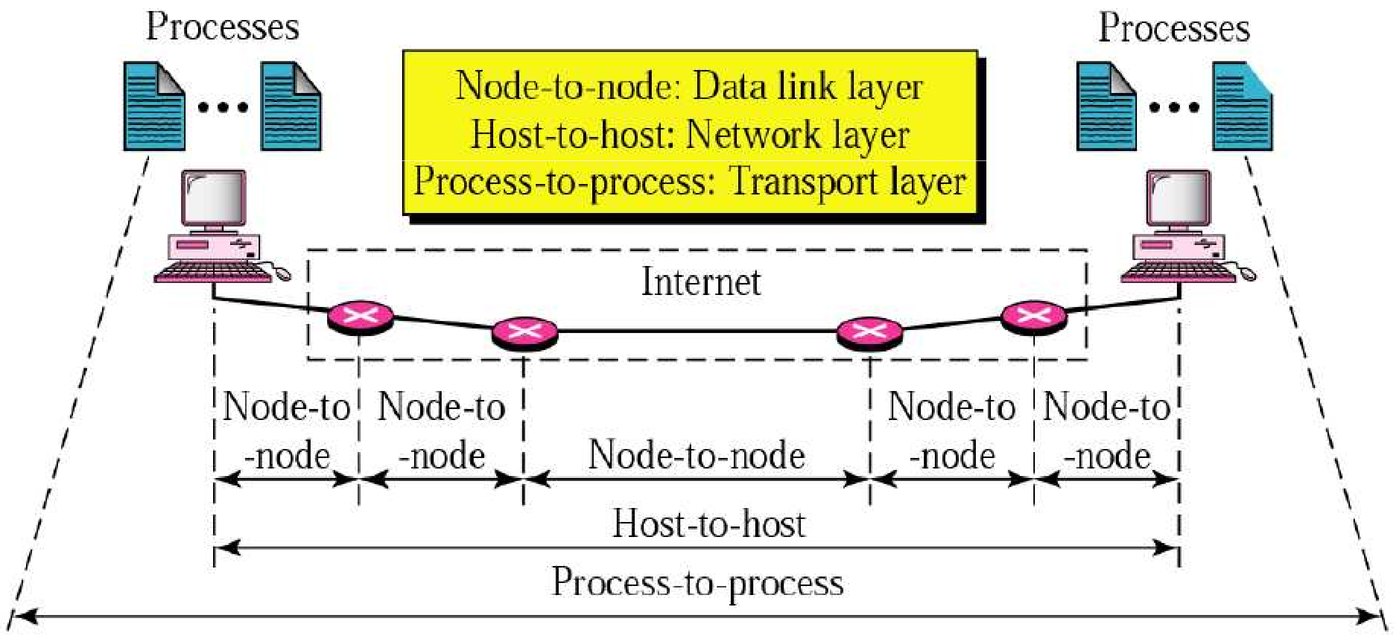
- In an analogous manner, a computer network may make available multiple transport protocols, with each protocol offering a different service model to applications.
- The possible services that Alice and Bob can provide are clearly constrained by the possible services that the postal service provides.
- For example, if the postal service doesn't provide a maximum bound on how long it can take to deliver mail between the two houses (e.g., three days), then there is no way that Alice and Bob can guarantee a maximum delay for mail delivery between any of the cousin pairs.
- In a similar manner, the services that a transport protocol can provide are often constrained by the service model of the underlying network-layer protocol.
- *If the network layer protocol cannot provide delay or bandwidth guarantees for 4-PDUs sent between hosts, then the transport layer protocol can not provide delay or bandwidth guarantees for the messages sent between processes*.

Host-to-Host Communication Vs Process-to-Process Communication


Processes
Processes
Internet
Domain of network-layer protocol
Domain of transport-layer protocol

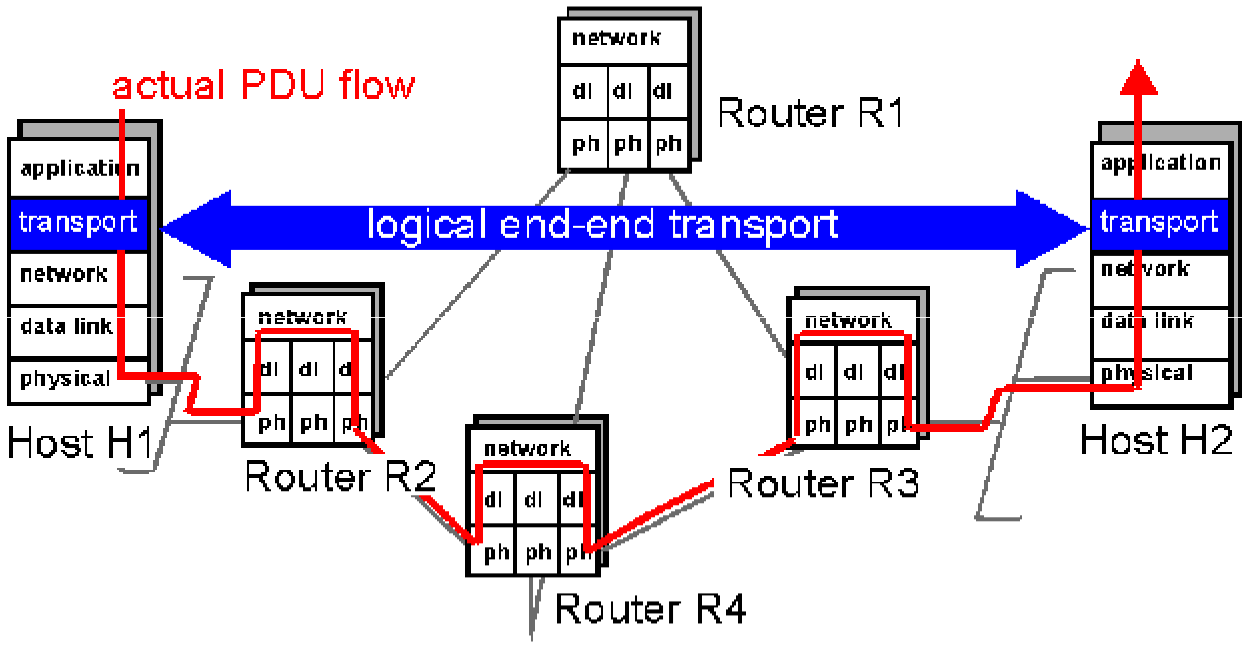
Node-to-Node, Host-to-Host and Process-to-Process Delivery

Transport Layer



Process-to-process delivery

The Notion of Logical Communication (1/4)



The Notion of Logical Communication (2/4)

- A transport layer protocol provides for **logical communication** between application processes running on different hosts.
- By "logical" communication, we mean that although the communicating application processes are not *physically* connected to each other (indeed, they may be on different sides of the planet, connected via numerous routers and a wide range of link types), from the applications' viewpoint, it is as if they were physically connected.
- *Application processes use the logical communication provided by the transport layer to send messages to each other, free for the worry of the details of the physical infrastructure used to carry these messages*.

The Notion of Logical Communication (3/4)

- *Transport layer protocols are implemented in the end systems but not in network routers.*
- Network routers only act on
  - the network layer fields of the layer-3 Protocol Data Units (PDUs);
  - they do not act on the transport layer fields.

The Notion of Logical Communication (4/4)

- At the sending side,
  - the transport layer converts the messages it receives from a sending application process into layer 4-PDUs (that is, **Transport-Layer Protocol Data Units (TPDU)**).
  - This is done by (possibly) breaking the application messages into smaller chunks and adding a transport-layer header to each chunk to create TPDUs.
  - The transport layer then passes the TPDUs to the network layer, where each TPDU is encapsulated into a Layer 3-PDU.
- At the receiving side,
  - the transport layer receives the TPDUs from the network layer, removes the transport header from the TPDUs, reassembles the messages and passes them to a receiving application process.
- A computer network can make more than one transport layer protocol available to network applications.
  - For example, the Internet has two protocols – TCP and UDP.
  - ***Each of these protocols provides a different set of transport layer services to the invoking application***.
- All transport layer protocols provide an application multiplexing/demultiplexing service.

Inference

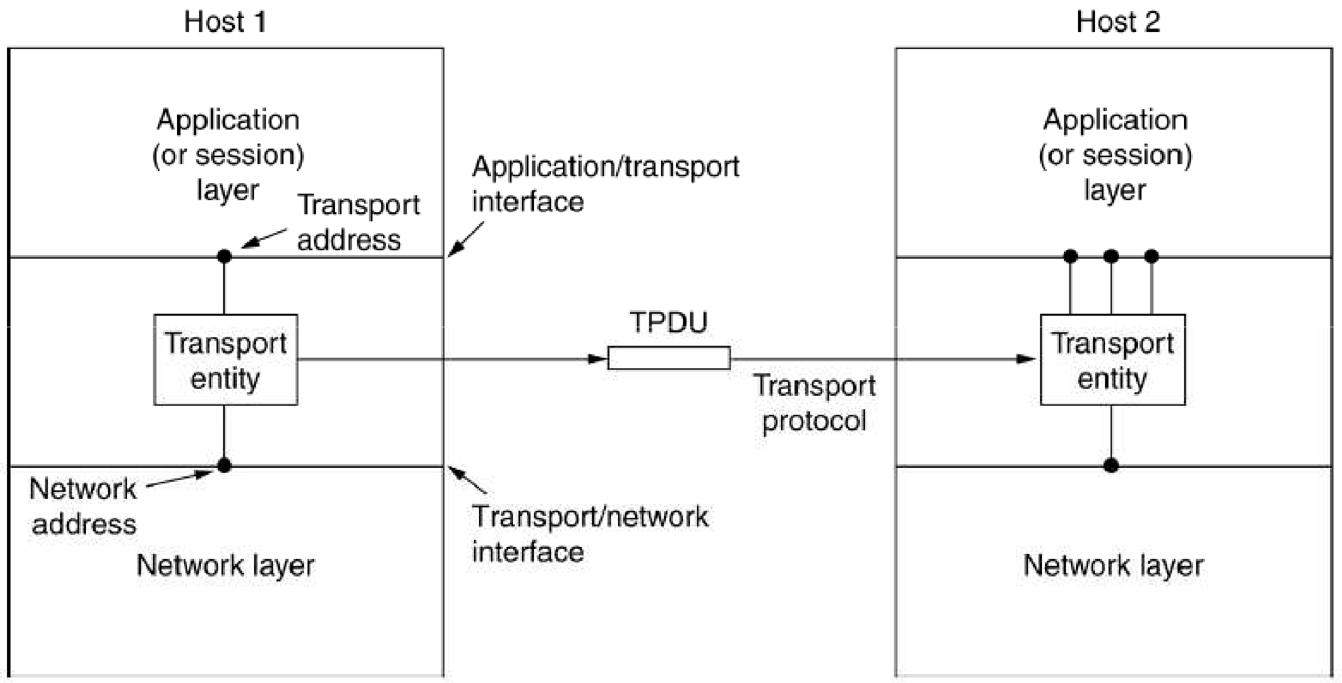
- Certain services can be offered by a transport protocol even when the underlying network protocol doesn't offer the corresponding service at the network layer.
- For example, *a transport protocol can offer reliable data transfer service to an application even when the underlying network protocol is unreliable*, that is, even when the network protocol loses, garbles and duplicates packets.
- As another example, *a transport protocol can use encryption to guarantee that application messages are not read by intruders*, even when the network layer cannot guarantee the secrecy of layer 4-PDUs.

Transport Service

Transport Entity

- The ultimate goal of the transport layer is to provide efficient, reliable, and cost-effective service to its users, normally processes in the application layer.
- To achieve this goal, the transport layer makes use of the services provided by the network layer.
- The *hardware and/or software within the transport layer that does the work is called the* ***transport entity***.

Services Provided to the Upper Layers



The network, transport, and application layers.

Services Provided to the Upper Layers (1/2)

- The bottom four layers can be seen as the transport service provider, whereas the upper layer(s) are the transport service user.
- To allow users to access the transport service, the transport layer must provide some operations to application programs, that is, a transport service interface.
- **Each transport service has its own interface.**
- This transport interface is truly bare bones, but it gives the essential flavor of what a connection- oriented transport interface has to do.
  - It allows application programs to establish, use, and then release connections, which is sufficient for many applications.

Services Provided to the Upper Layers (2/2)

- The transport service is similar to the network service, but there are also some important differences.
- The main difference is that the network service is intended to model the service offered by real networks.
- Real networks can lose packets, so the network service is generally unreliable.
- The (connection-oriented) transport service, in contrast, is reliable.
- A second difference between the network service and transport service is whom the services are intended for.
- The network service is used only by the transport entities. Few users write their own transport entities, and thus few users or programs ever see the bare network service.
- In contrast, many programs see the transport primitives. Consequently, the transport service must be convenient and easy to use.

Transport Service Primitives (1/5)

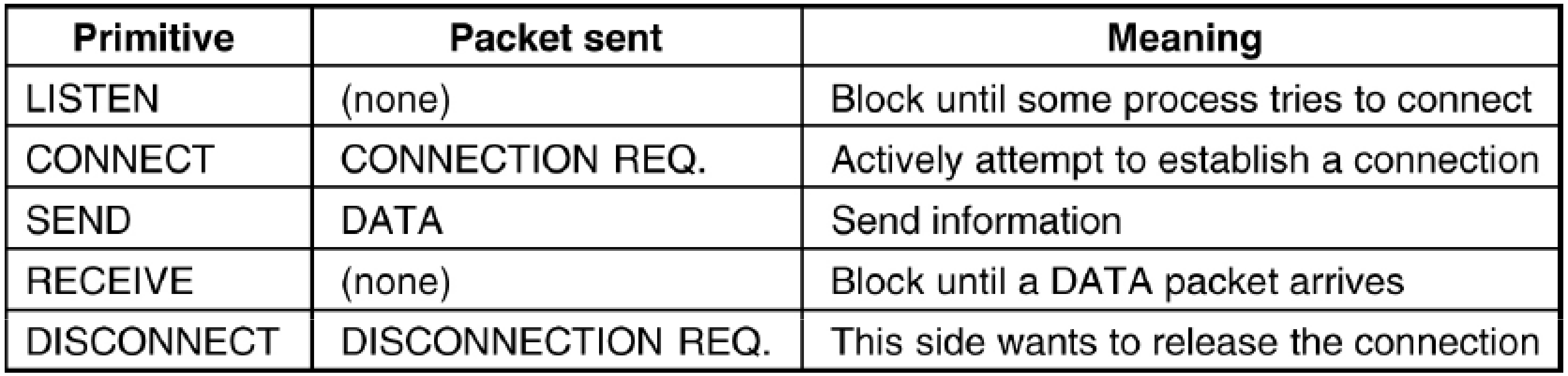
- A service is specified by a set of primitives.
- A primitive means operation.
- To access the service a user process can access these primitives.
- These primitives are different for connection oriented service and connectionless service.
- There are five types of service primitives:
  - **LISTEN**: When a server is ready to accept an incoming connection it executes the LISTEN primitive. It blocks waiting for an incoming connection.
  - **CONNECT**: It connects the server by establishing a connection. Response is awaited.
  - **RECIEVE**: Then the RECIEVE call blocks the server.
  - **SEND**: Then the client executes SEND primitive to transmit its request followed by the execution of RECIEVE to get the reply. Send the message.
  - **DISCONNECT**: This primitive is used for terminating the connection. After this primitive one can’t send any message. When the client sends DISCONNECT packet then the server also sends the DISCONNECT packet to acknowledge the client. When the server package is received by client then the process is terminated.

Transport Service Primitives (2/5)

- The purpose of the transport layer—to provide a reliable service on top of an unreliable network.
- To get an idea of what a transport service might be like, consider the five primitives.
- To start with, the server executes a LISTEN primitive, typically by calling a library procedure that makes a system call that blocks the server until a client turns up. When a client wants to talk to the server, it executes a CONNECT primitive.
- The transport entity carries out this primitive by blocking the caller and sending a packet to the server.

Transport Service Primitives (3/5)

The socket primitives for TCP.



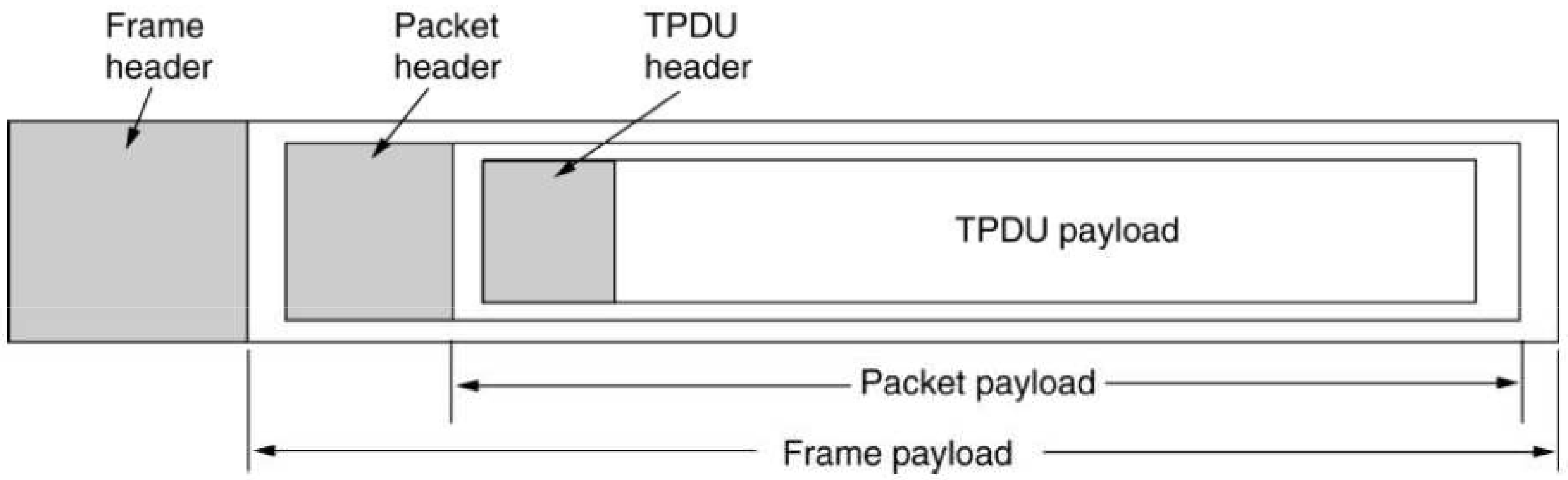
Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

The primitives for a simple transport service.

Transport Service Primitives (4/5)

- The term segment for messages sent from transport entity to transport entity. TCP, UDP and other Internet protocols use this term.
- Thus, segments (exchanged by the transport layer) are contained in packets (exchanged by the network layer).
- In turn, these packets are contained in frames (exchanged by the data link layer).
- When a frame arrives, the data link layer processes the frame header and, if the destination address matches for local delivery, passes the contents of the frame payload field up to the network entity.
- The network entity similarly processes the packet header and then passes the contents of the packet payload up to the transport entity.

Transport Service Primitives (5/5)

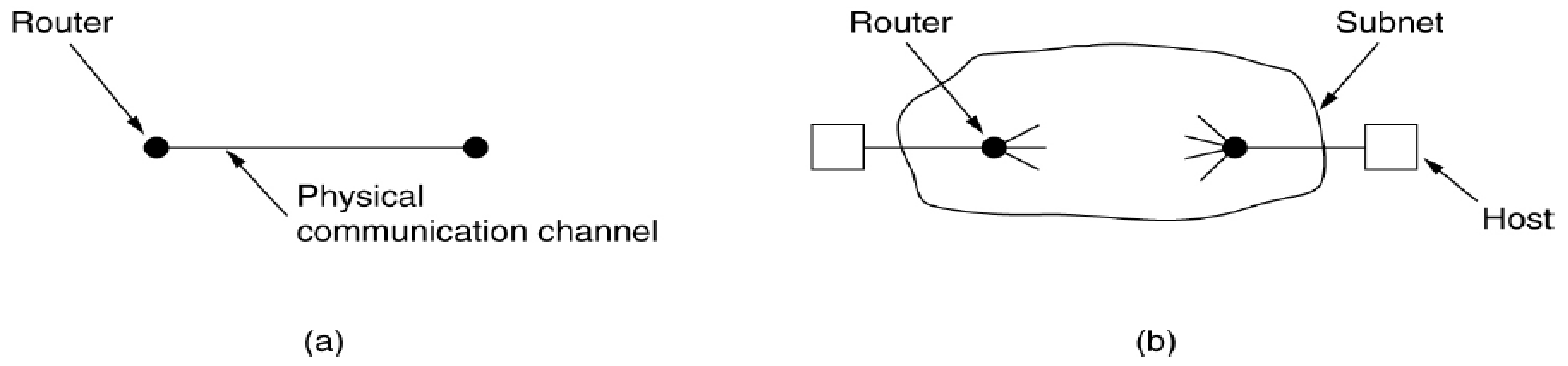


The nesting of TPDUs, packets, and frames.

Transport Protocols Vs Data Link Protocols

- The transport service is implemented by a **transport protocol used between the** two transport entities.
- Though the transport protocols resemble the Data Link Protocols, significant differences are present due to the major dissimilarities between the environments in which the two protocols operate.
- A physical channel exists in DLL, where as it is replaced by the entire subnet for Transport Layer.
- No explicit addressing of destinations is required in DLL, where it is required for Transport layer.
- A final difference between the data link and transport layers is one of amount rather than of kind.
  - Buffering and flow control are needed in both layers, but the presence of a large and dynamically varying number of connections in the transport layer may require a different approach than we used in the data link layer.

Environment of the Data Link Layer and Transport Layer



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

Primitives for a Simple Transport Service

- The sequence starts with a server application like FTP or TELNET telling the transport entity (a piece of hardware or software) through a library callable routine that it is ready to accept clients. This is LISTEN.
- A client can then request a connection by sending its transport entity a CONNECT request. The transport entity will respond by sending out a connection request TPDU. The server would send back a connection accepted TPDU to establish the connection.
- Once the connection is established, data can be sent back and forth using the SEND and RECEIVE primitives. [Note: This is STOP-AND-WAIT--the simplest transmission procedure.]
- Disconnects can be symmetric or assymetric. Assymetric: Either party can send a disconnect releasing the connection (like the phone system). Symmetric: When one party sends a disconnect, that means that it has no more data to send. The connection isn't released until both parties send a disconnect.

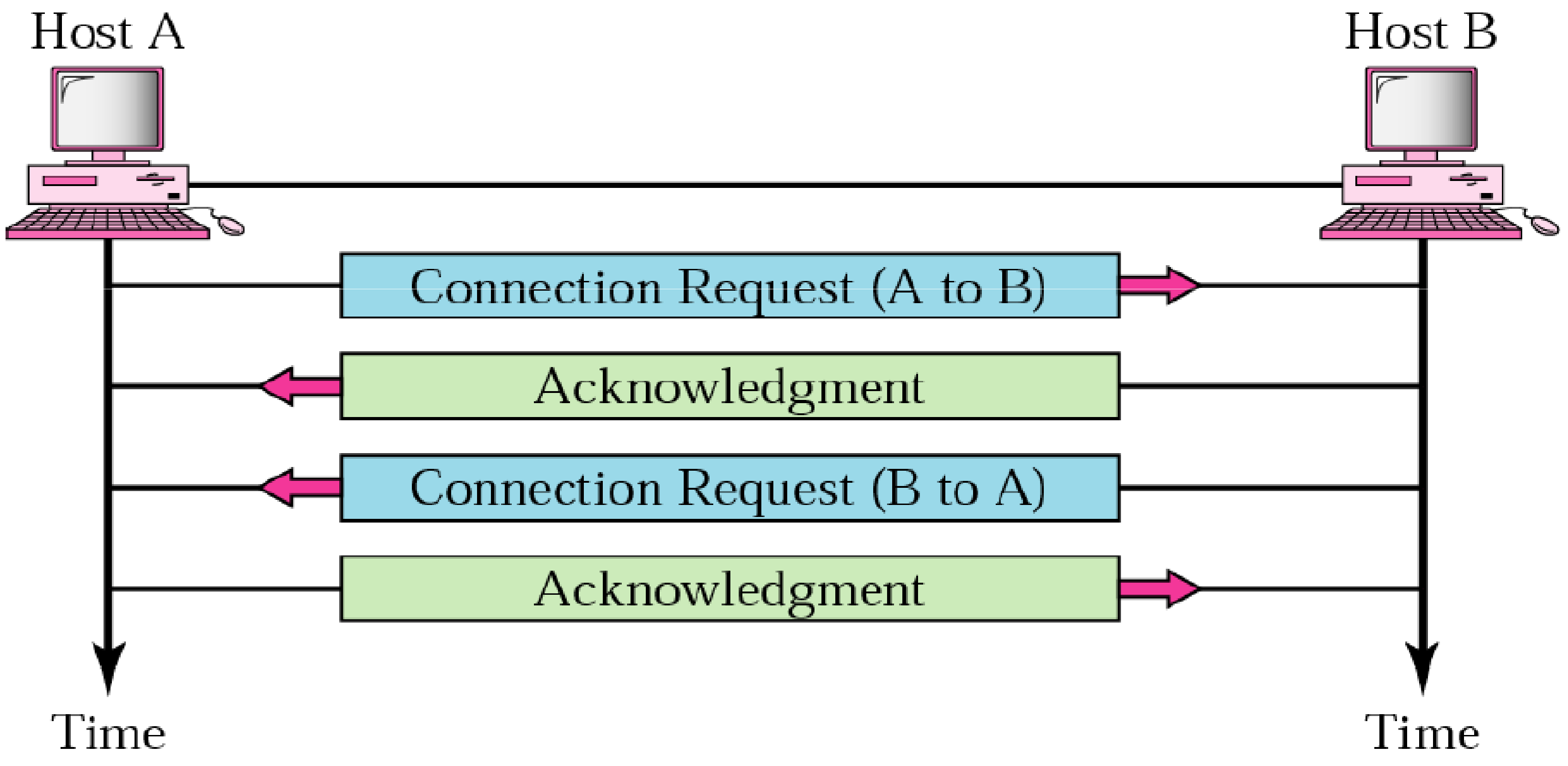
Elements of Transport Protocols

- Addressing
- Connection Establishment
- Connection Release
- Flow Control and Buffering
- Multiplexing
- Crash Recovery

Two way Handshake Connection Establishment

- A sends a CR, B replies with an ACK.
- Lost CR’s are handled by re-transmission
  - – Can lead to duplicate CRs
  - – Ignore duplicate CRs once connected

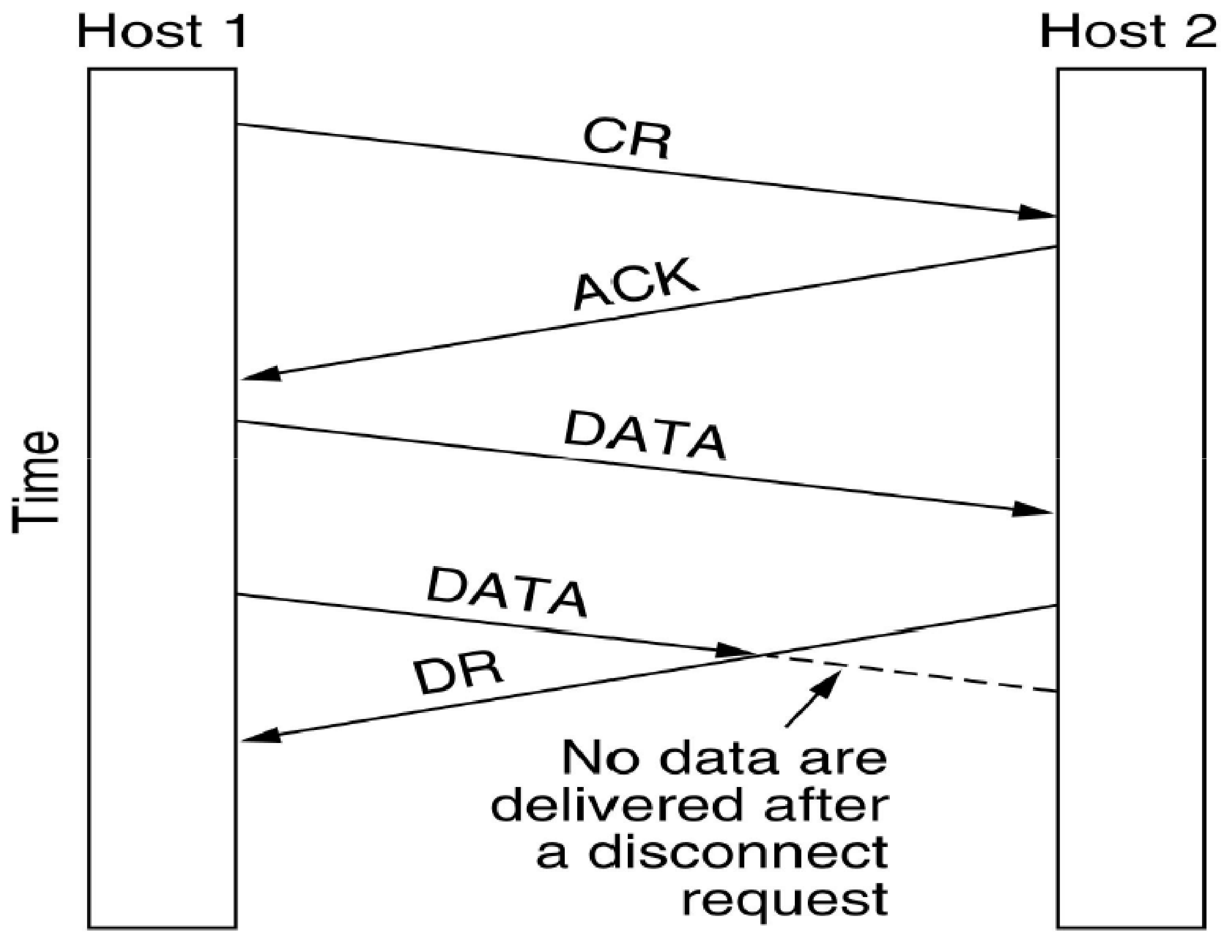
Connection Establishment



Connection Release

- *Connection release is easy but needs care.*
- 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
- Symmetric release treats the connection as two separate unidirectional connections and requires each one to be released separately.
- *Asymmetric release may result in loss of data.*

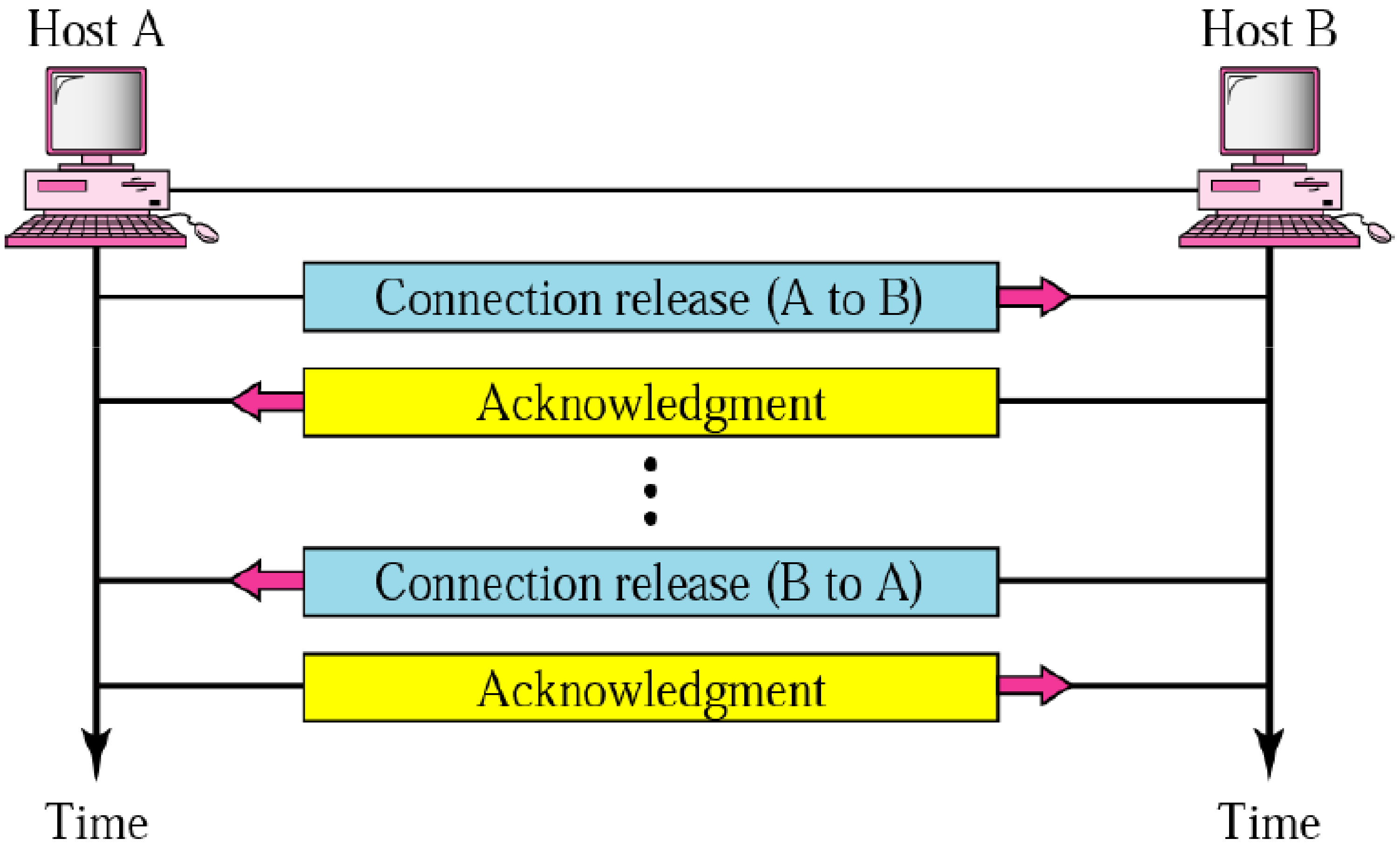
Asymmetric Release



CR: connection request
DR: disconnect request

Abrupt disconnection with loss of data → needs better protocol

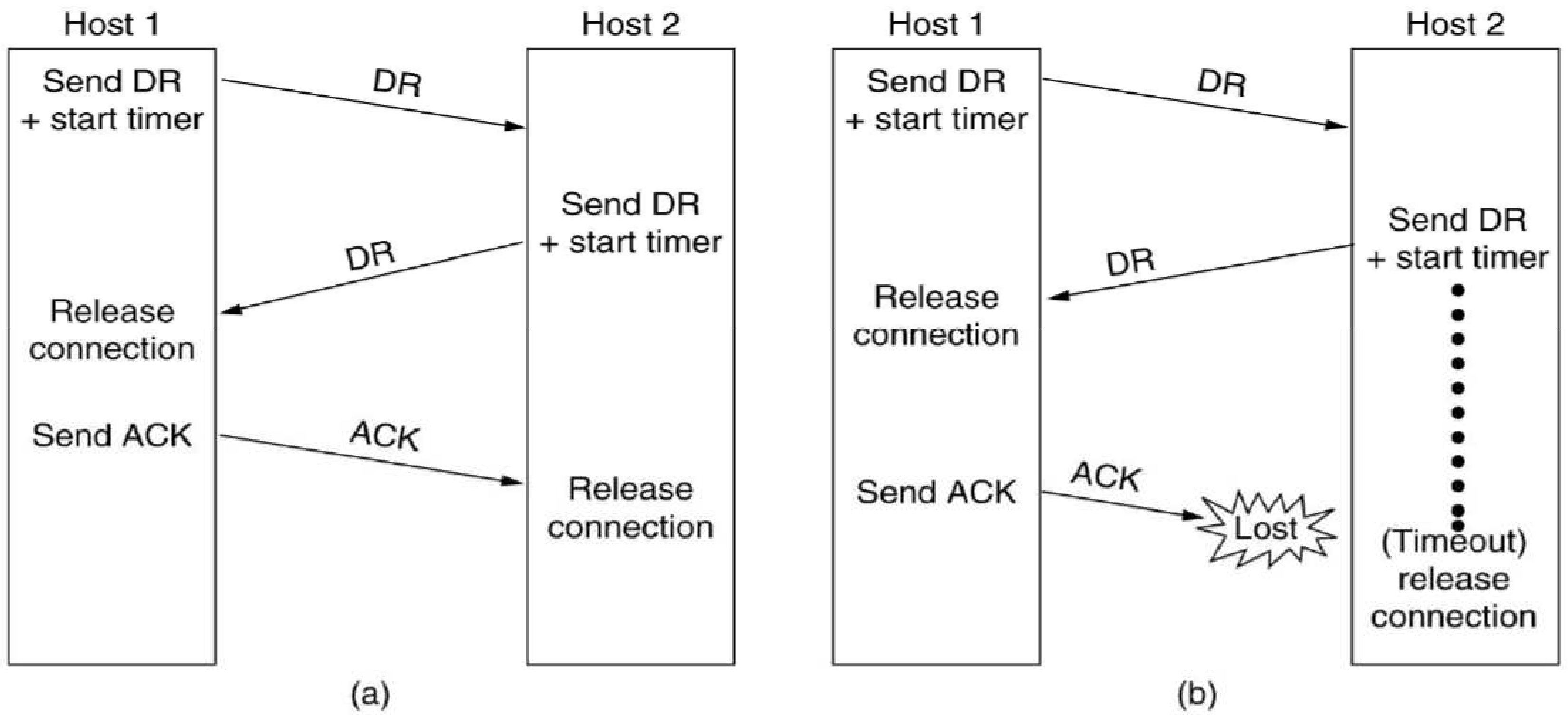
Symmetric Release (1/2)



Symmetric Release (2/2)

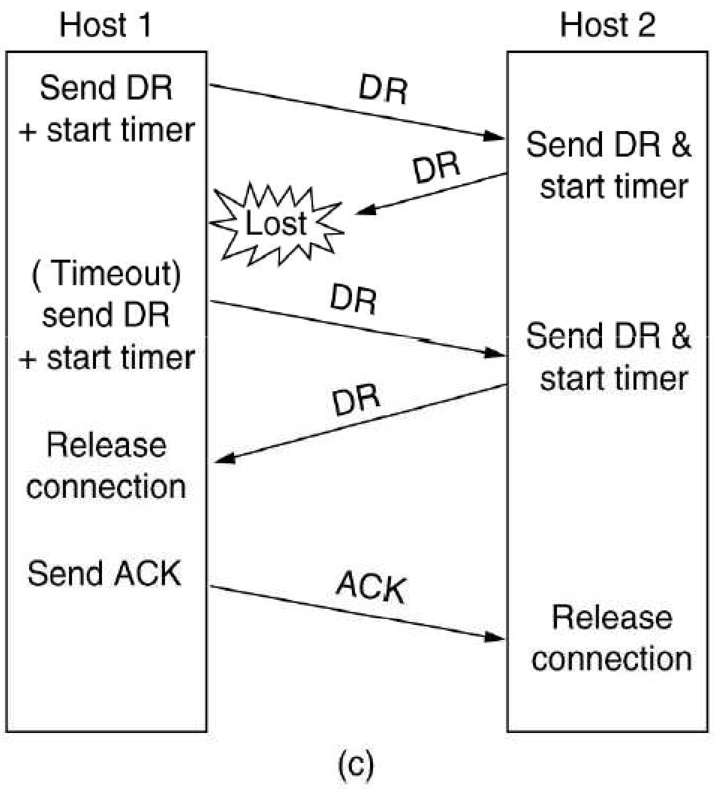
- Symmetric release does the job when each process has a fixed amount of data to send and clearly knows when it has sent it.
- One can envision a protocol in which Host 1 says “I am done. Are you done too?”. If Host 2 responds: “I am done too. Good Bye, the connection can be safely released”.
- Unfortunately, this protocol does not always work: i.e., **Two-Army Problem**.

Connection Release (1/2)

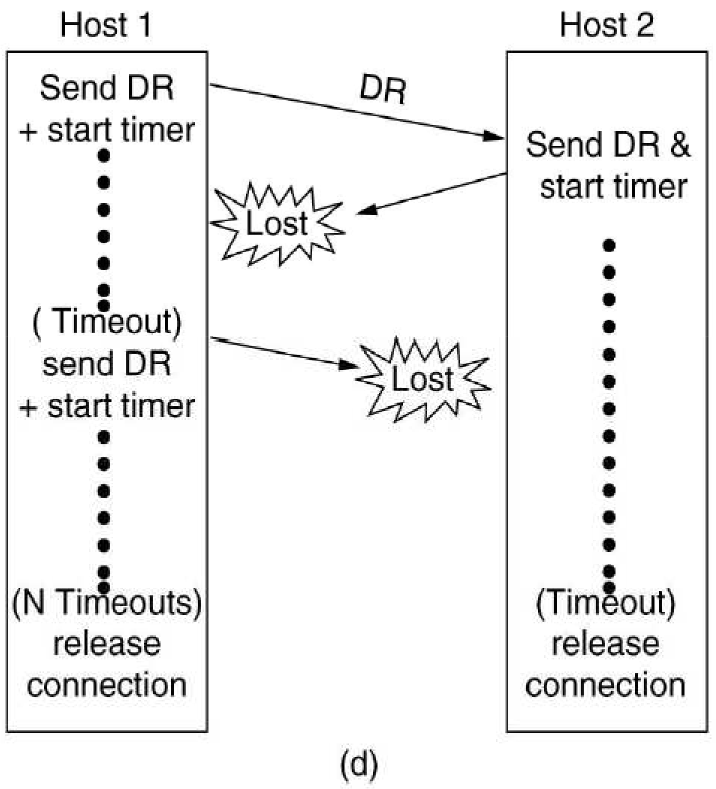


(a) Normal case of a three-way handshake. (b) final ACK lost.

Connection Release (2/2)



(c) Response DR lost subsequent DRs lost.

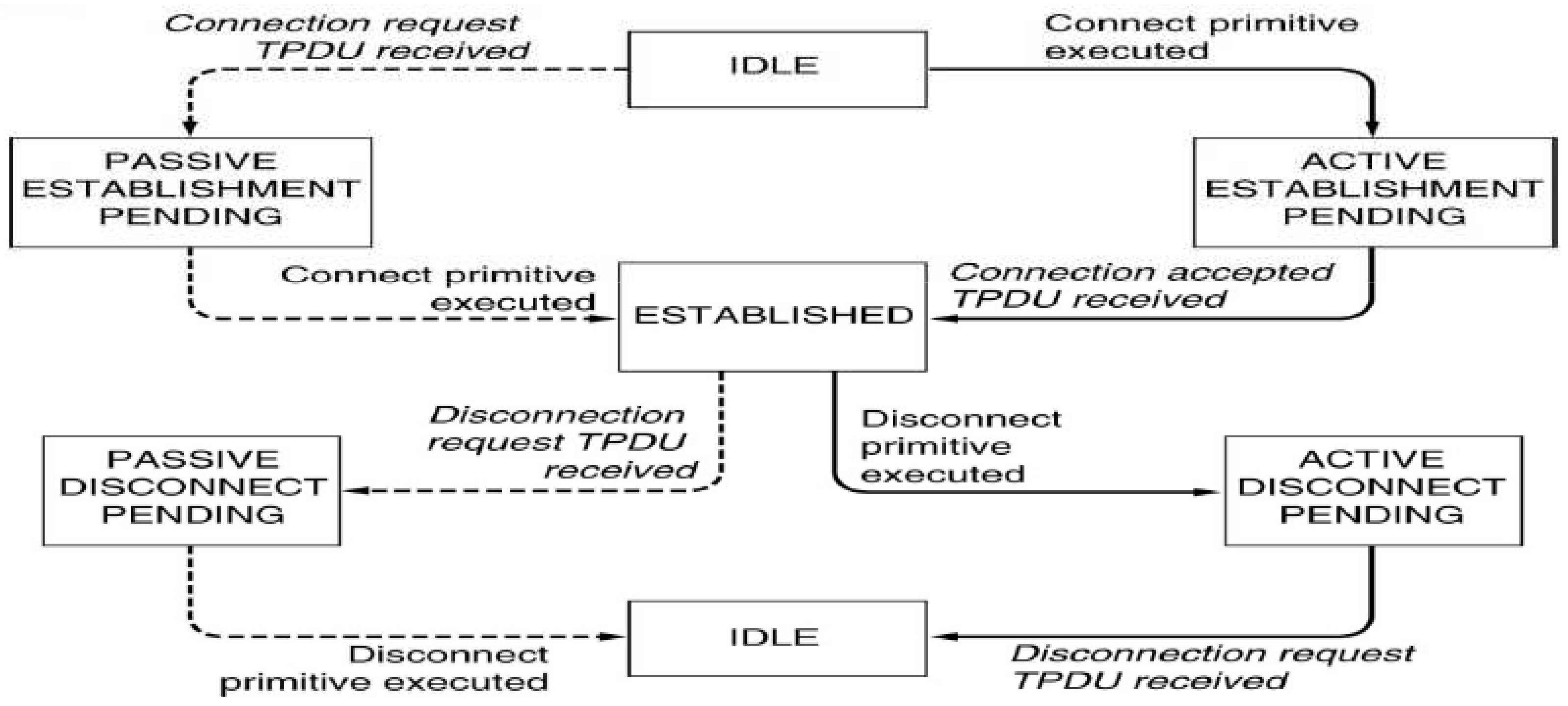

Host 1
Host 2
Send DR + start timer
DR
Send DR & start timer
Lost
(Timeout) send DR + start timer
Lost
(N Timeouts) release connection
(Timeout) release connection
(d)

(d) Response lost and

State Diagram for a Simple Connection Management Scheme (1/3)

- Transport layer protocols can be very complex and *state diagrams are used to understand them*.
- A state diagram for connection establishment and release for the simple primitives.
- *Each transition is triggered by some event, either a primitive executed by the local transport user or an incoming packet (labeled in italics).*
- *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.*

State Diagram for a Simple Connection Management Scheme (2/3)

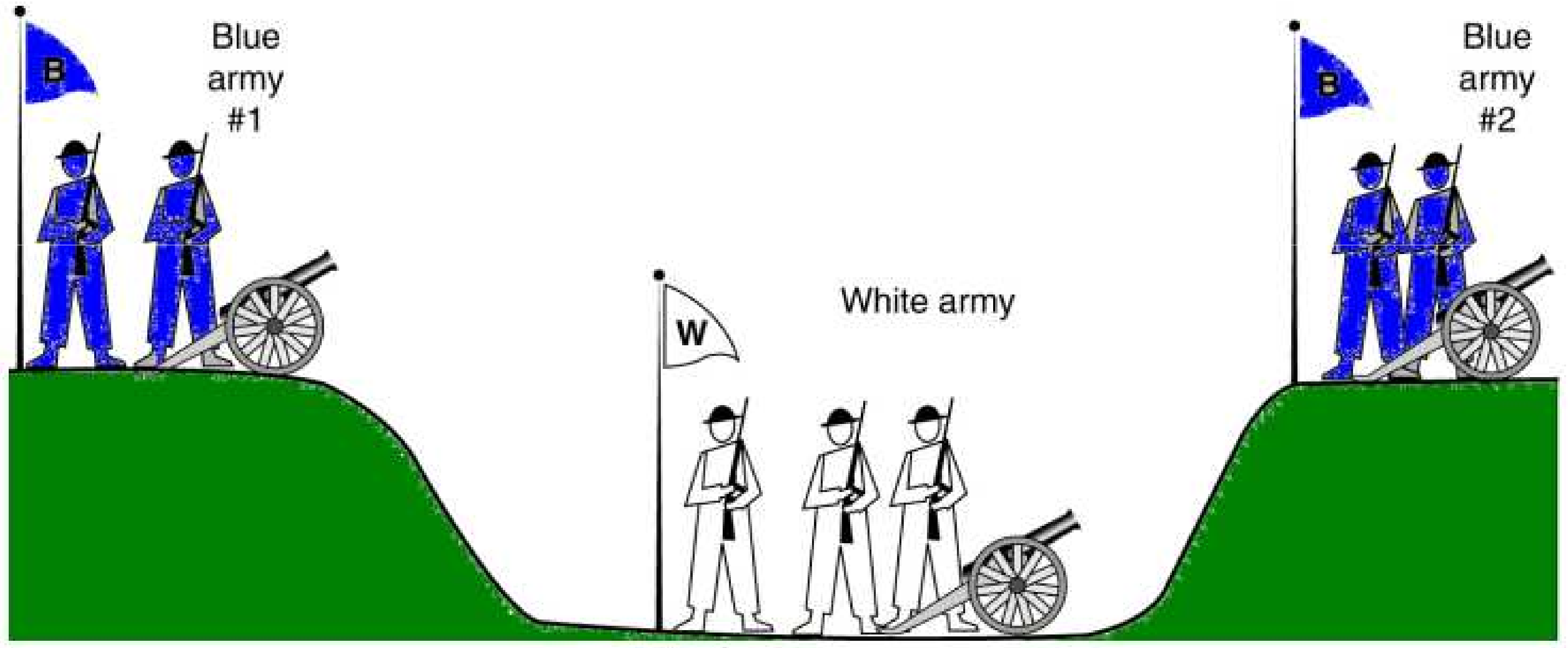


Connection request TPDU received
IDLE
Connect primitive executed
PASSIVE ESTABLISHMENT PENDING
ACTIVE ESTABLISHMENT PENDING
Connect primitive executed
ESTABLISHED
Connection accepted TPDU received
Disconnection request TPDU received
Disconnect primitive executed
PASSIVE DISCONNECT PENDING
ACTIVE DISCONNECT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

State Diagram for a Simple Connection Management Scheme (3/3)

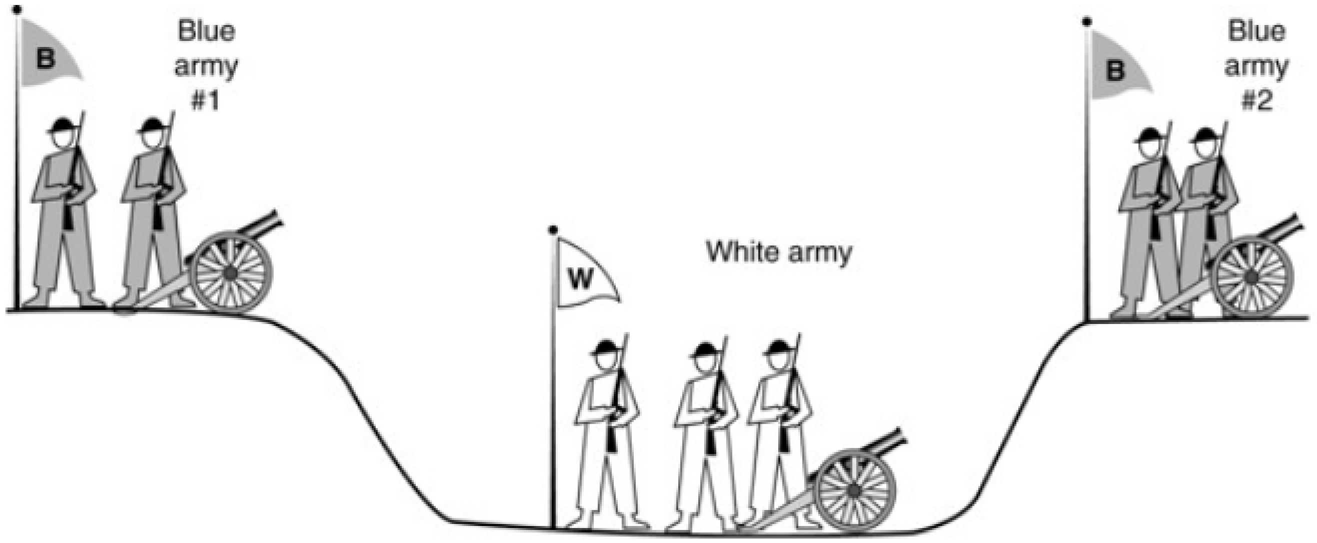
- For simplicity, it is assumed that each TPDU is separately acknowledged.
- Assume that a symmetric disconnection model is used, with the client going first.
- This model is quite unsophisticated.
- There are more realistic models on that.

Two-Army Problem (or) Two General Problem



Two-Army Problem (or) Two General Problem

- A white army is encamped in a valley.
- On both 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.



Two-Army Problem (or) Two General Problem –Dilemma 1

- If the two Blue Army platoons attack the White Army together and at exactly the right time, they will prevail.
- The longer they wait, the more surprised the White Army will be by the attack.
- But if they wait too long, they will run out of supplies, grow weak, and starve.
- The timing is critical.
- But if they are not coordinated, they will surely lose.
- They must attack at exactly the same time.
- When the time is right Blue Army 1 (B1) will send a messenger to Blue Army 2 (B2) that says, "Attack at dawn!"
- But B1 may become concerned that B2 did not get the message and consequently not attack.
- If that happens, B1 will be defeated. Alternately, B2 may become concerned that B1 will become concerned, so they may not attack, leaving B1 to be defeated in solitude.

Two-Army Problem (or) Two General Problem –Dilemma 2

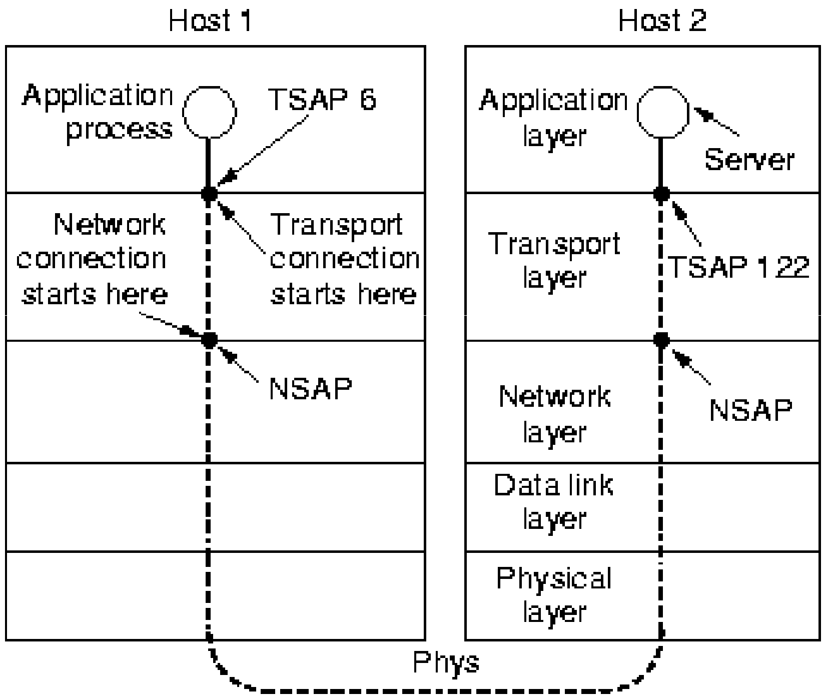
- So what if they agree that the recipient of the message, B2 , will return an ACK to the sender, B1?
- This is just one level of indirection.
- The B2 may become concerned that the ACK was lost and not attack.
- Or B1 may become concerned that B2 became concerned and did not attack.
- This problem can't be solved by ACK-ACK, or even ACK-ACK-ACK-ACK -- more ACKs just add more levels of indirection, but the same problem remains.

Two-Army Problem (or) Two General Problem –Dilemma 3

- Another issue might be fake messages.
- What if the White Army sent an impostor to deliver a message to B2 telling them to attack too early.
- They would be defeated if they followed it.
- But if they did not obey messages for fear that they were fraudulent, B1 would be defeated when they did attack, even after advising B2.
- This fear might also prompt an army from acting upon a perfectly valid message.

The Moral of the Story

- The moral of this story is that *there is no solution to this problem if the communications medium is unreliable*.
- Please note that it is the medium, not the protocol. This is an important distinction.
- *A reliable protocol above an unreliable medium can guarantee that a message will eventually be sent*, provided of course that the recipient eventually becomes ready and accessible.
- ***But no protocol can guarantee that a message will be delivered within a finite amount of time*** -- error conditions may persist for long and indeterminate amounts of time.



Host 1
Host 2
Application process
TSAP 6
Network connection starts here
Transport connection starts here
NSAP
Application layer
Server
Transport layer
TSAP 122
Network layer
NSAP
Data link layer
Physical layer
Phys

The need of Port and Socket (1/2)

- *To determine which local process at a given host actually communicates with which process, at which remote host, using which protocol.*

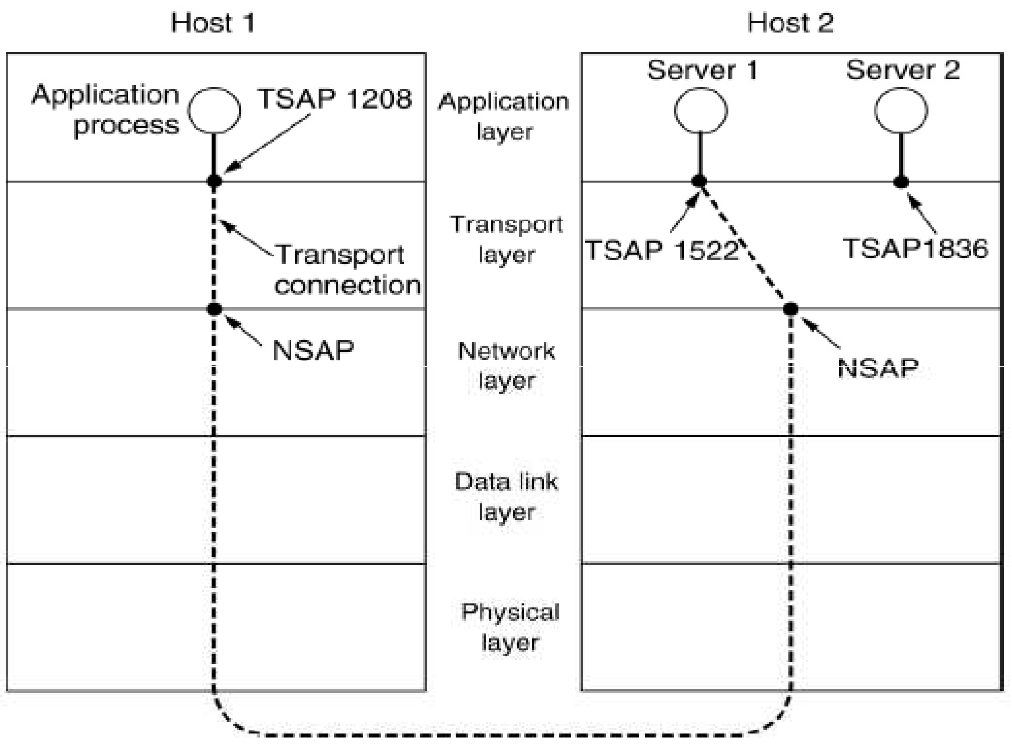
The need of Port and Socket (2/2)

- *An application process is assigned a process identifier number (process ID), which is likely to be different each time that process is started*.
- Process IDs differ between operating system platforms, hence they are not uniform.
- A server process can have multiple connections to multiple clients at a time, hence simple connection identifiers would not be unique.
- *The concept of ports and sockets provides a way to uniformly and uniquely identify connections and the programs and hosts that are engaged in them, irrespective of specific process IDs*.

Solution?

- "Attack" is equivalent to "disconnect".
- *No protocol exists that works.*
- *In reality, one is usually prepared to take more risks when releasing connections.*

Addressing

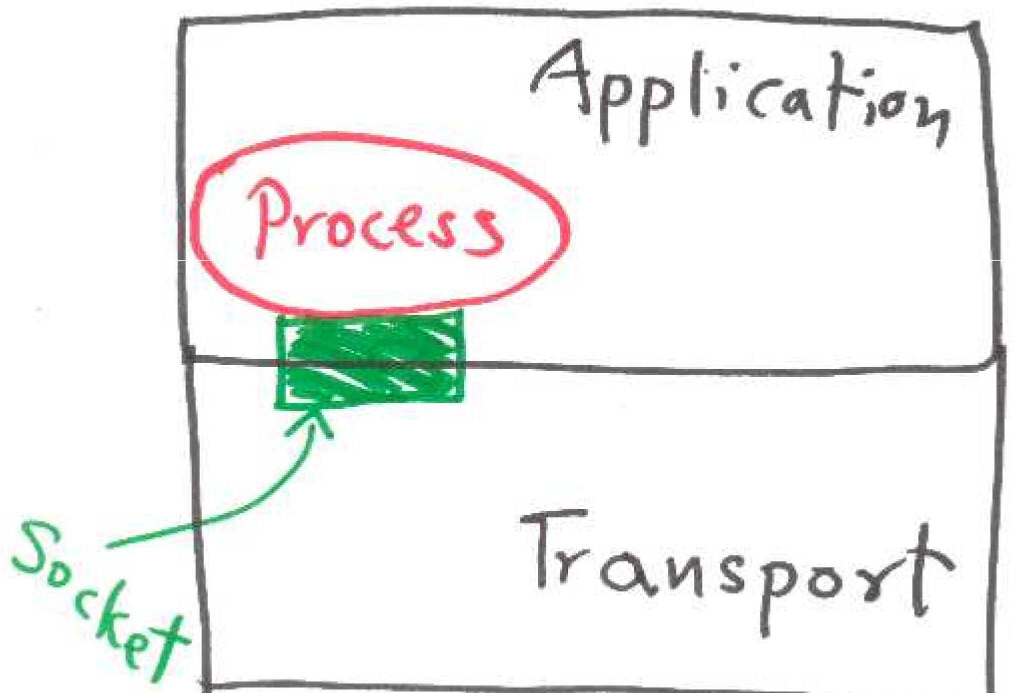


TSAPs, NSAPs and transport connections.

Sockets (1/2)

- Transport layer actually does not directly deliver messages to processes.
- So, who does that? Well, there's an intermediate software interface called a socket layer that does the transmission of messages between the underline network and the processes.
- Now remember that <u>*a process can have one or more sockets through which it can pass data to and from network and each of these sockets have a unique identifier*</u>.

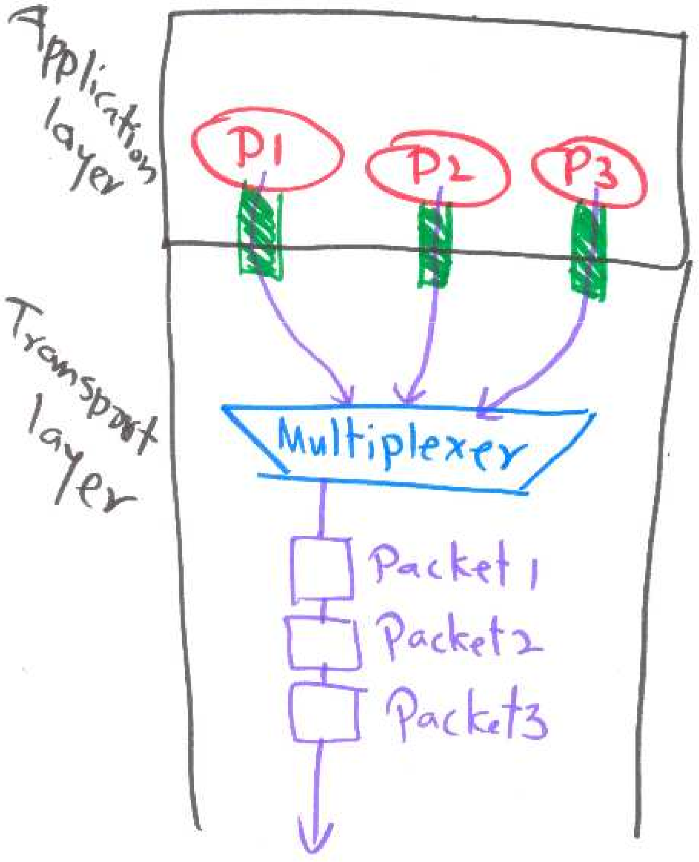
Sockets (2/2)



Multiplexing (1/2)

- Transport layer gathers chunks of data it receives from different sockets and encapsulate them with transport headers. Passing these resulting segments to the network layer is called multiplexing.

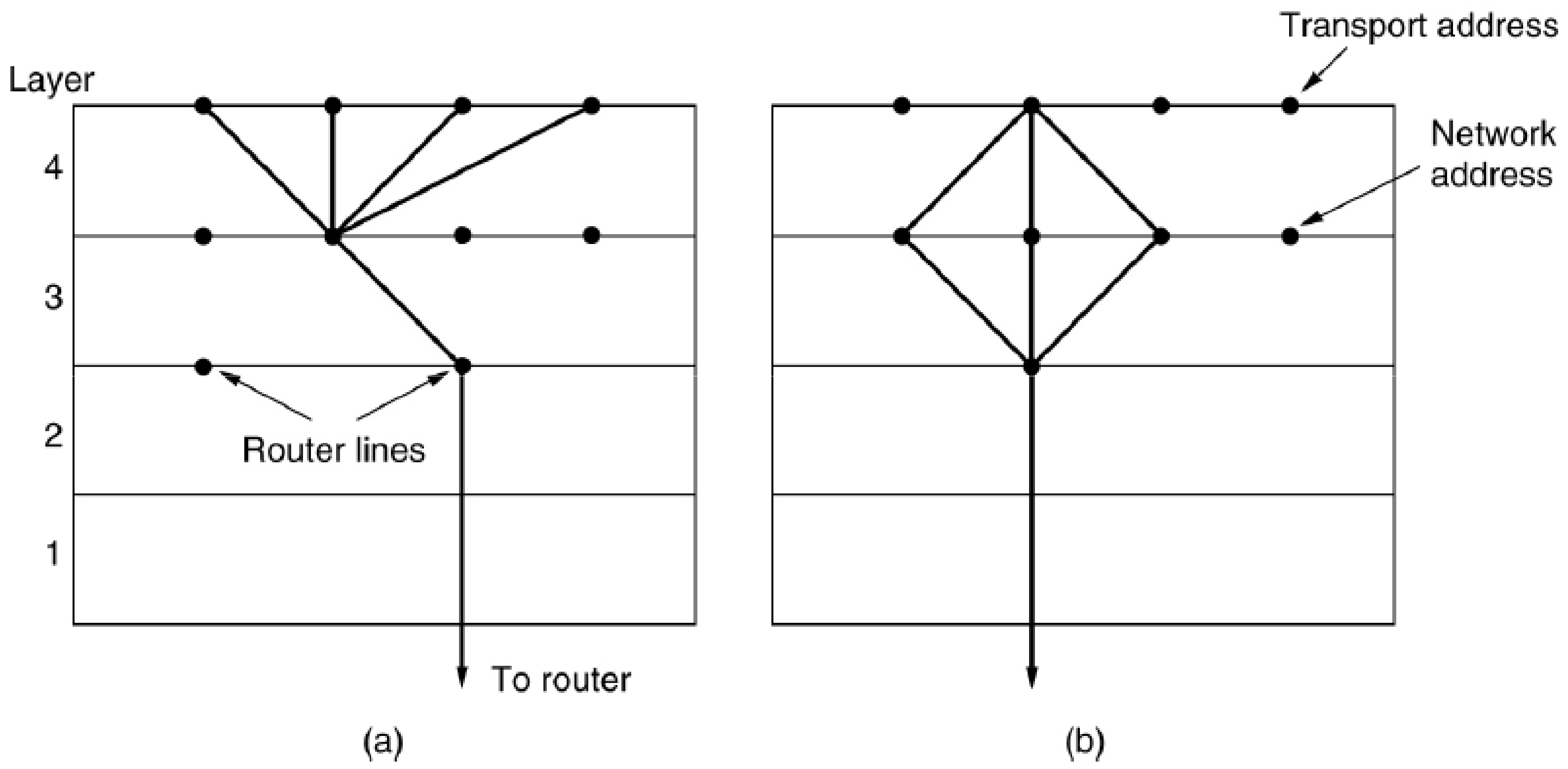
Multiplexing (2/2)



Upward and Downward Multiplexing (1/2)

- Upward multiplexing is used when multiple transport processes are connected to the same host. This would be useful if a virtual circuit (or worse yet, a physical circuit) is being underutilized.
- Downward multiplexing is the inverse scenario where a single network connection cannot handle the traffic from the transport layer process. By dividing the traffic among different network connections (e.g., different satellite connections), it is possible to get better throughput.

Upward and Downward Multiplexing (2/2)



(a) Upward multiplexing. (b) Downward multiplexing.

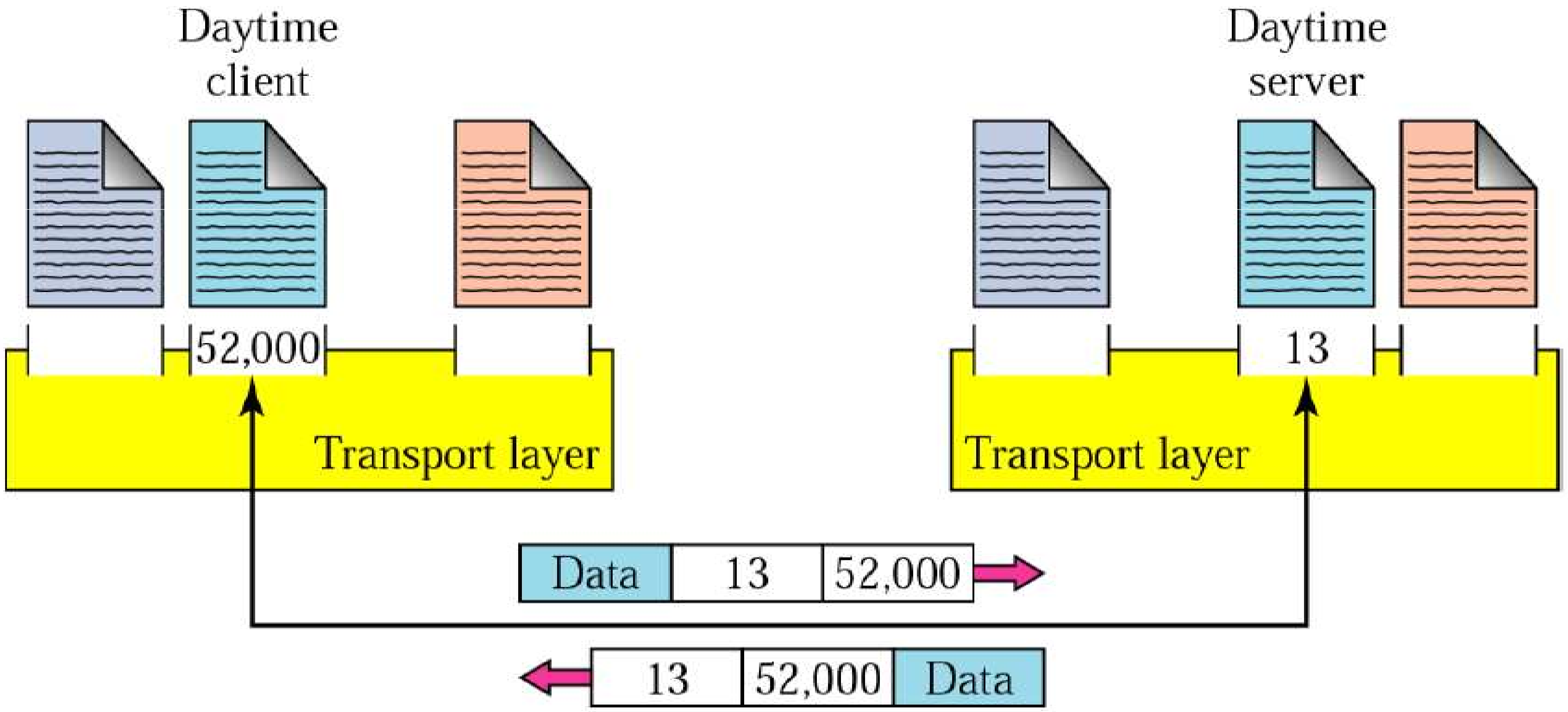
Demultiplexing

- The reverse process which is delivering data to the correct socket by the transport layer is called demultiplexing.
- But this still doesn’t explain how the transport layer identifies the correct socket.
- Port numbers are the ones that do the trick.

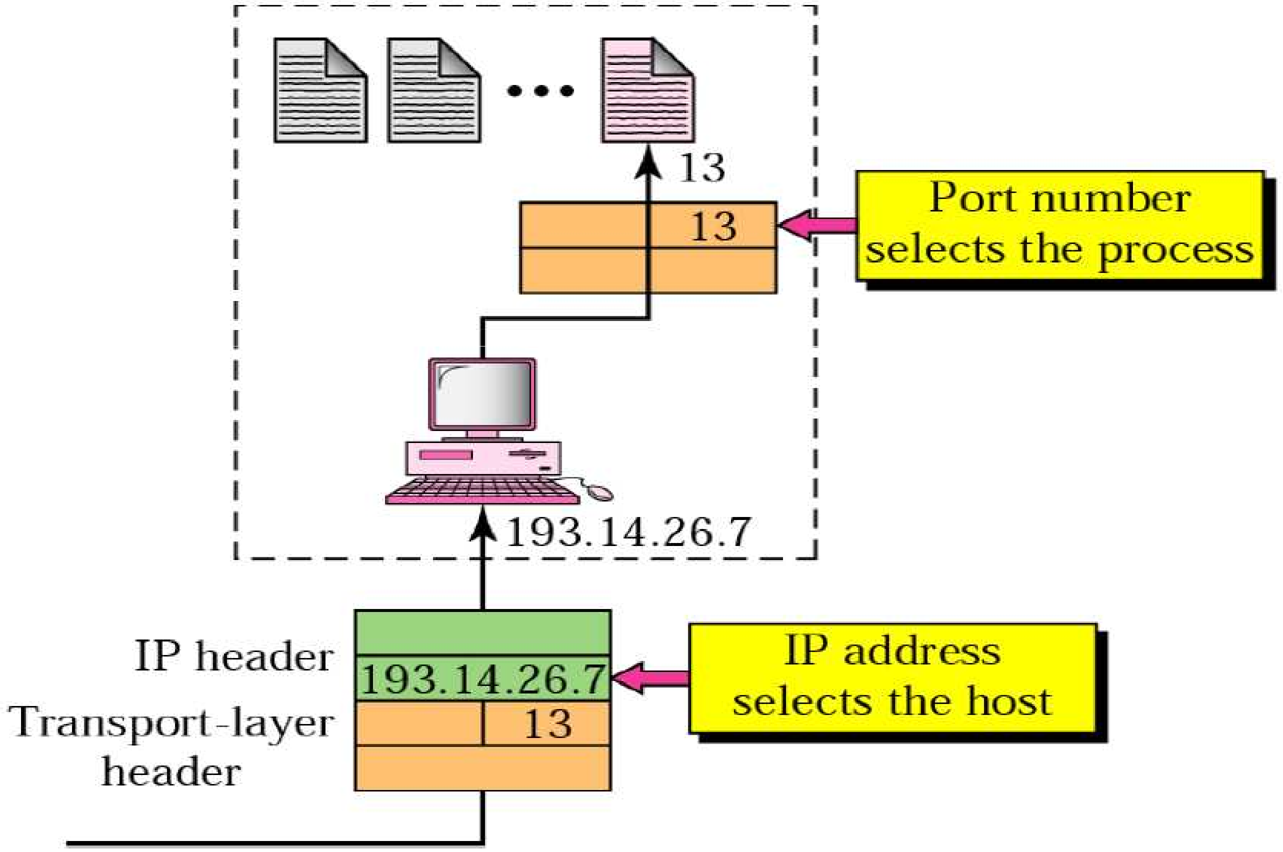
Ports

- Each process that wants to communicate with another process identifies itself to the TCP/IP protocol suite by one or more ports.
- A port is a 16-bit number, used by the host-to-host protocol to identify to which higher level protocol or application program (process) it must deliver incoming messages.

Port Numbers



IP Addresses Vs Port Numbers

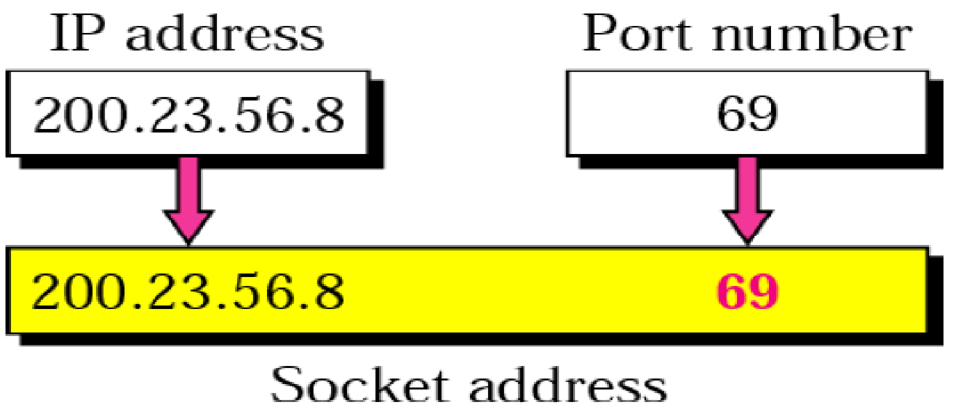


Socket and Socket Address

- A machine provides a variety of services and to differentiate between these services, each service is assigned with a unique port number.
- The port numbers less than 1024 are considered as **well known ports** and are reserved for standard services.
- In transport layer, two processes communicate with each other via sockets.
- *A socket acts as an end point of the communication path between the processes*.
- *The IP address and Port address put together defines the socket address*.

Socket Address

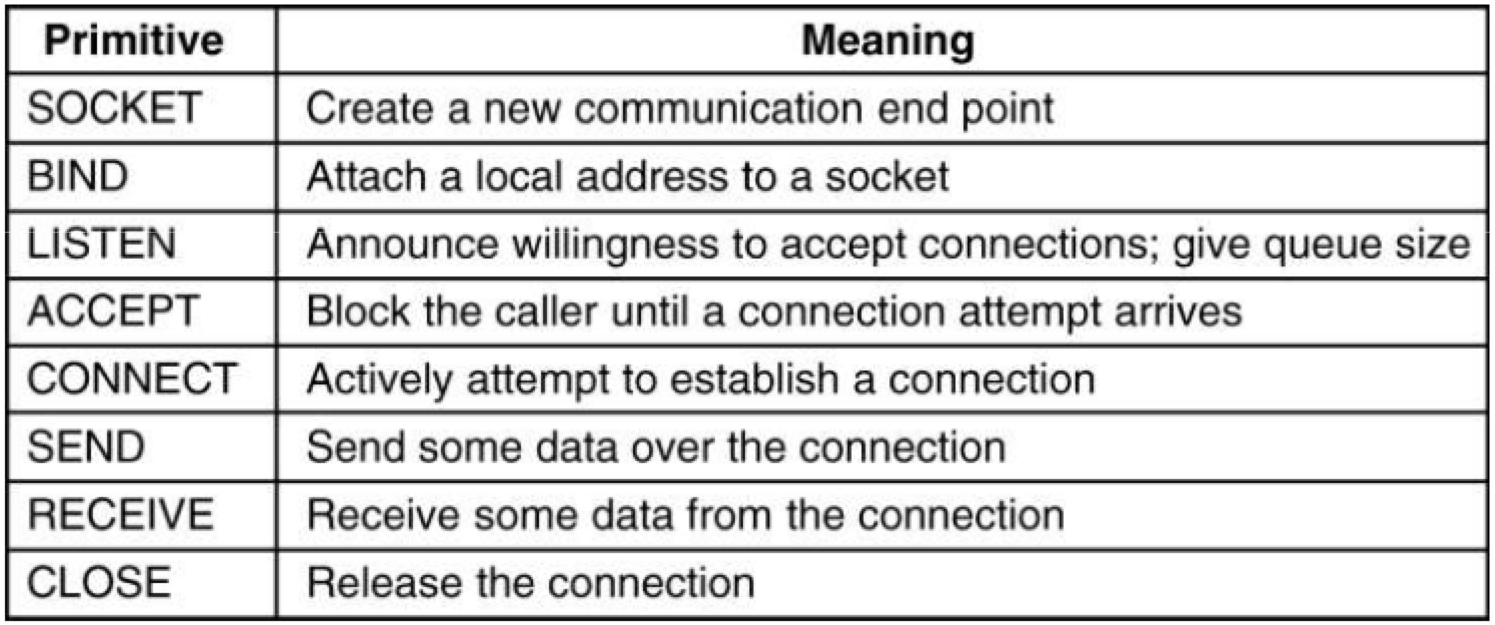
Socket Sample



Berkeley Sockets

- Sockets were first released as part of the Berkeley UNIX 4.2BSD software distribution in 1983.

The socket primitives for TCP



Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
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

- The primitives used in Berkeley UNIX for TCP are a bit more complex.
- Primitives:
  - <u>Socket</u>: This opens a connection between the application (server or client) and its transport entity.
  - <u>Bind</u>: This gives the socket an address (a port number). Some services like FTP and TELNET have "well-known numbers" (specified in the Internet RFC's) and do not require BIND.
  - <u>Listen</u>: Sets aside space for multiple incoming calls, but does not ready the server to accept them (block).
  - That is done by <u>ACCEPT</u>.