Data Communication
(DC)

Lecture 8b

Overview of the contents

- **Transport-Layer Services**
- **Connectionless and Connection-Oriented Protocols**

Transport-Layer Services

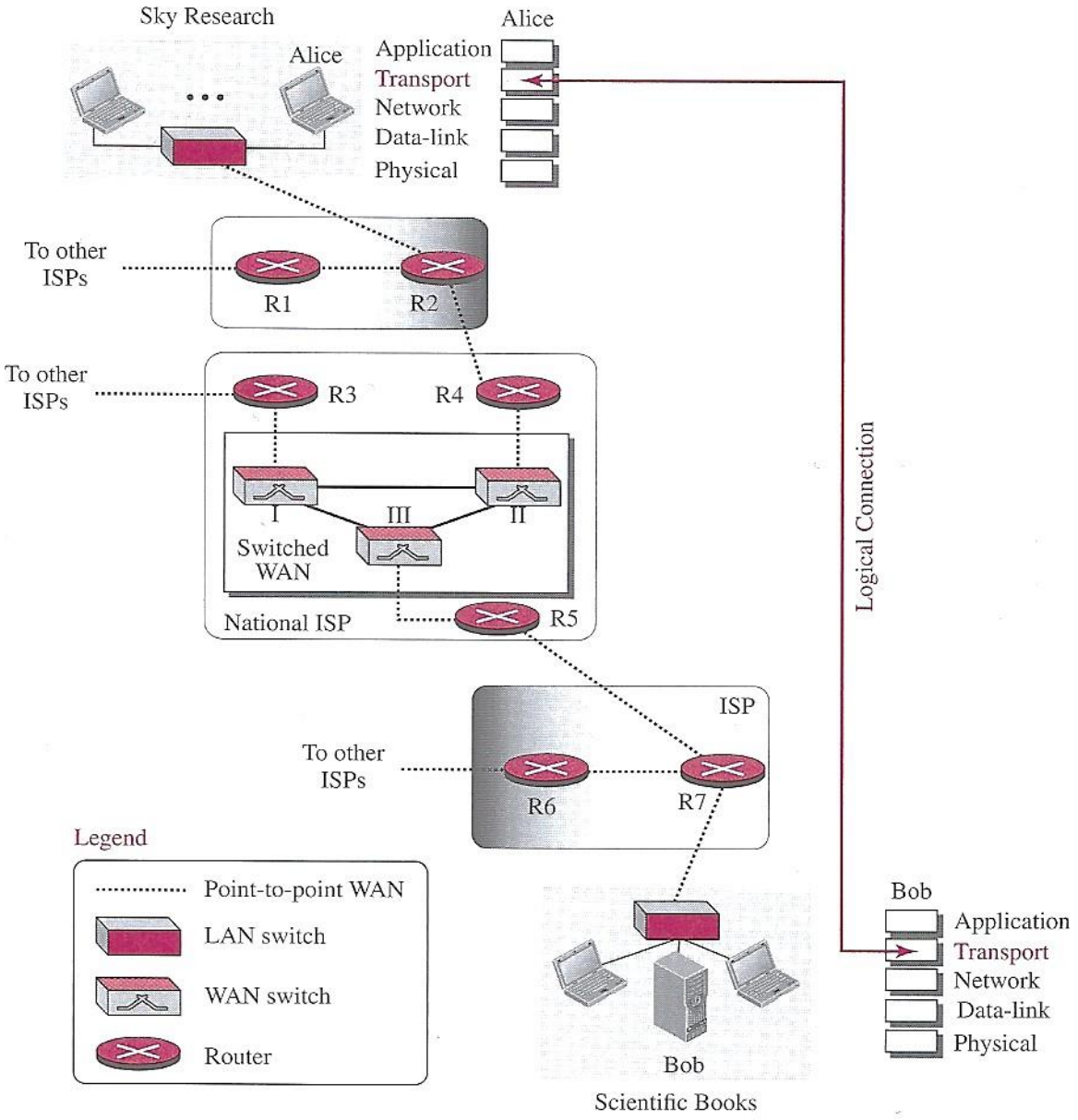
Transport Layer

Introduction

Transport Layer is located between the Application Layer and the Network Layer.

It provides **process-to-process communication** between two Application Layers, one on each host.

The communication takes place via a logical connection between the two hosts, each of which can be in different sites of the globe.

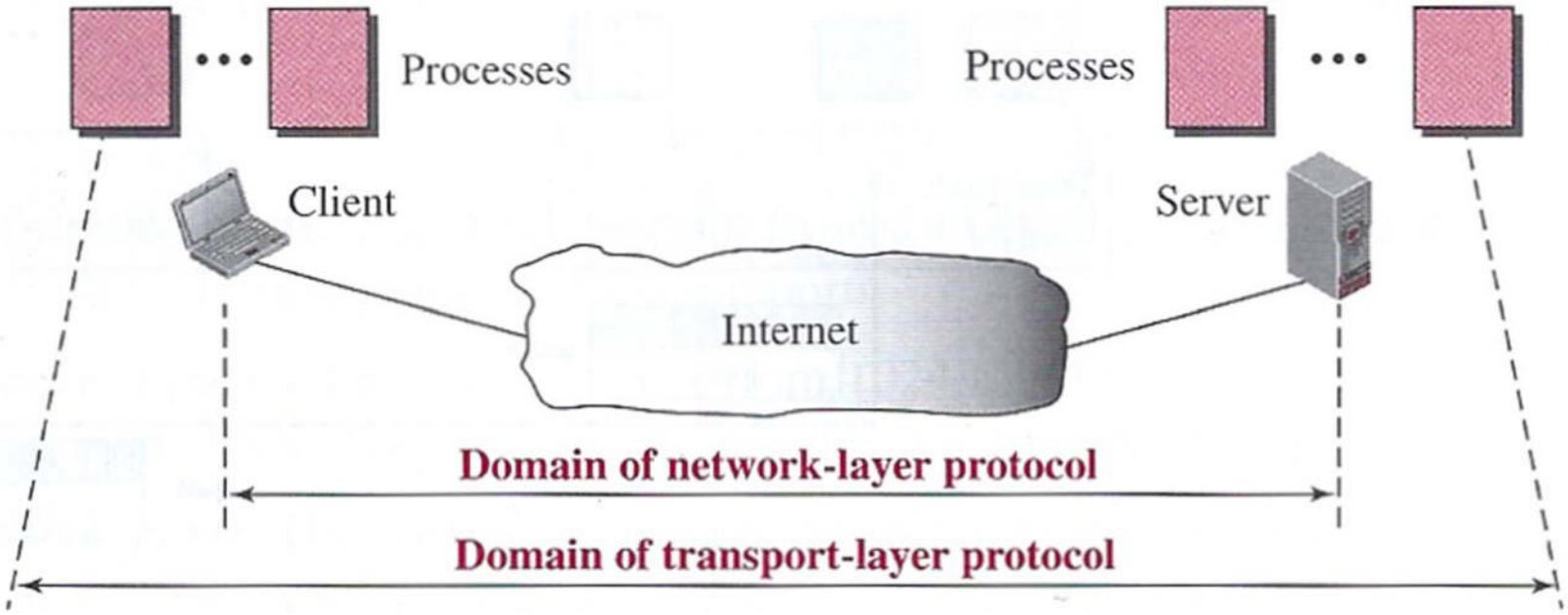


Transport Layer

Service

The transport layer ensures that data is delivered SAFELY from one process (running program) on one computer (host) to another process (running program) on another computer (host). **(by safe flow and error control)**

The Data Link layer is responsible for ***node-to-node communication*** **(Frames are sent between neighboring nodes on a LAN)**
The Network layer is responsible for ***host-to-host communication*** **(Datagrams are sent between hosts on the Internet.)**
The Transport layer is responsible for ***process-to-process communication***.



Transport Layer

Addressing

We need to have an addressing system so that we can distinguish different processes on a host. **The address in this transport layer is called a port number**.

This address has 16 bits, giving 65,536 different addresses (0 - 65,535)

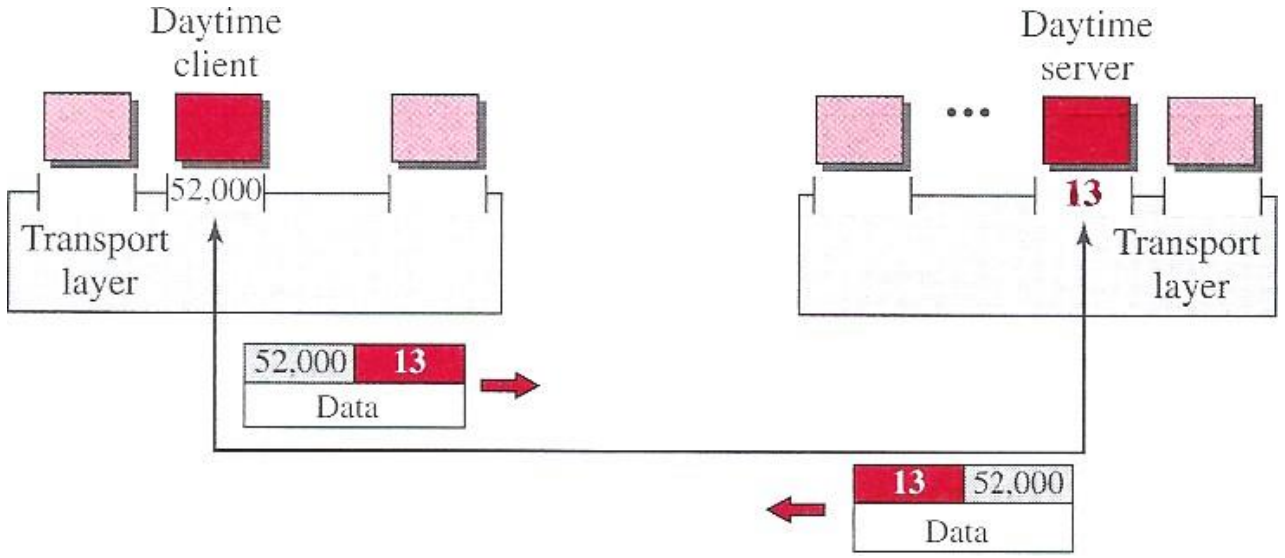
The client program defines itself by a port number (ephemeral port number). This (random) number is selected by the transport layer software running on the client's host.

The server process must also define itself by a port number. This port number, however, cannot be chosen randomly (otherwise the client could not find the server).

It has been decided by TCP/IP to use universal numbers for servers, that is, **well-known port numbers.**

Transport Layer

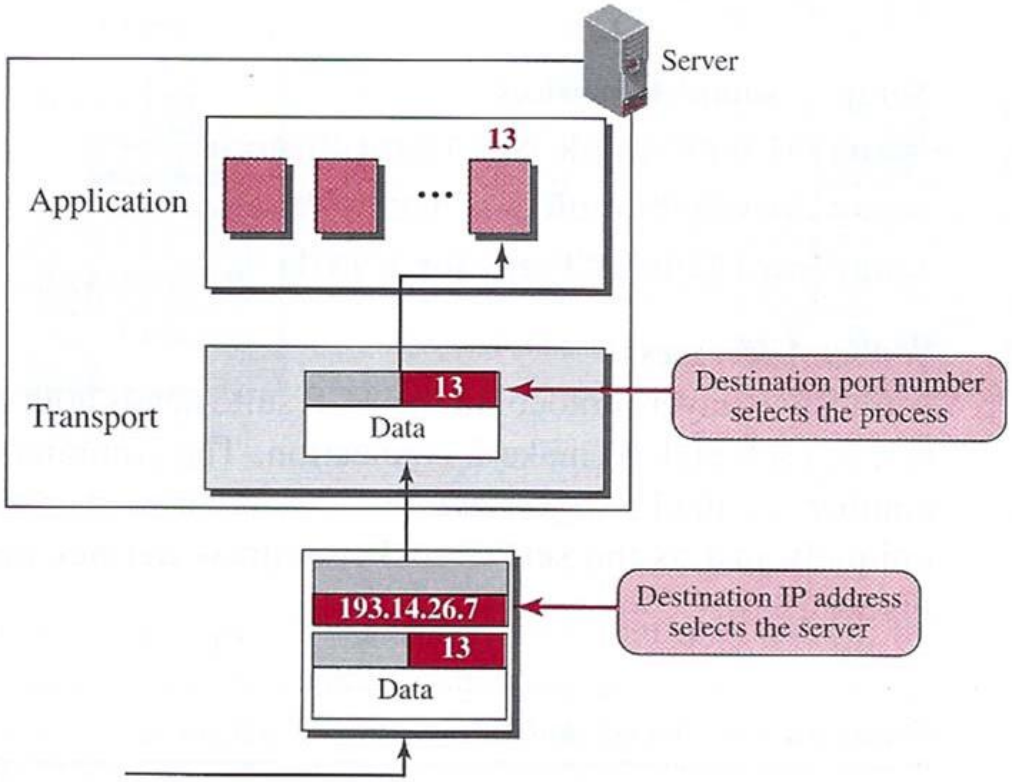
Addressing: example



As we can see, the daytime client has been given a random port number (52,000), while the daytime server has a well-known port number (13)

Transport Layer

Addressing: example



It should be clear that the IP address and port number have their different roles to play when finding the final receiver of some data.

IP address *selects the host among the different hosts in the world.*

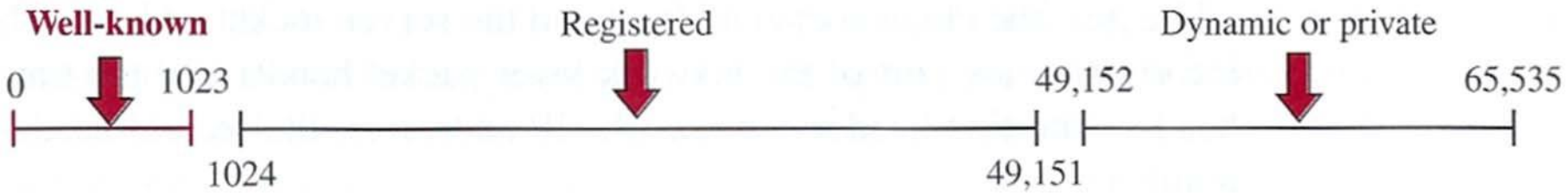
Port number *selects a specific process on this selected host.*

Transport Layer

ICANN Range

Internet **C**orporation for **A**ssigned **N**ames and **N**umbers (**ICANN**) has divided port addresses into 3 ranges:

- **Well-known ports**: these are numbers in the range 0 – 1,023 and are assigned and controlled by ICANN.
- **Registered ports**: these are numbers in the range 1,024 – 49,151 and are not assigned or controlled by ICANN. They can be only registered with ICANN to avoid duplication.
- **Dynamic ports**: these are numbers in the range 49,152 - 65,535 and are neither registered nor controlled. They can be assigned to any process, and these are the temporary ports.



Transport Layer

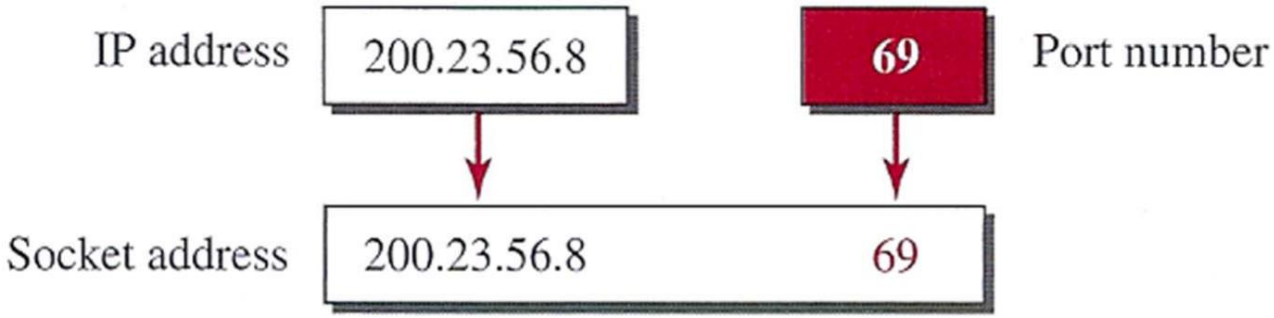
Socket addresses

Process-to-process communication requires two identifiers:

- IP address
- Port number

The combination of an IP address and a port number is called a Socket address.

- The client socket address uniquely identifies the client process!
- The server socket address uniquely identifies the server process!

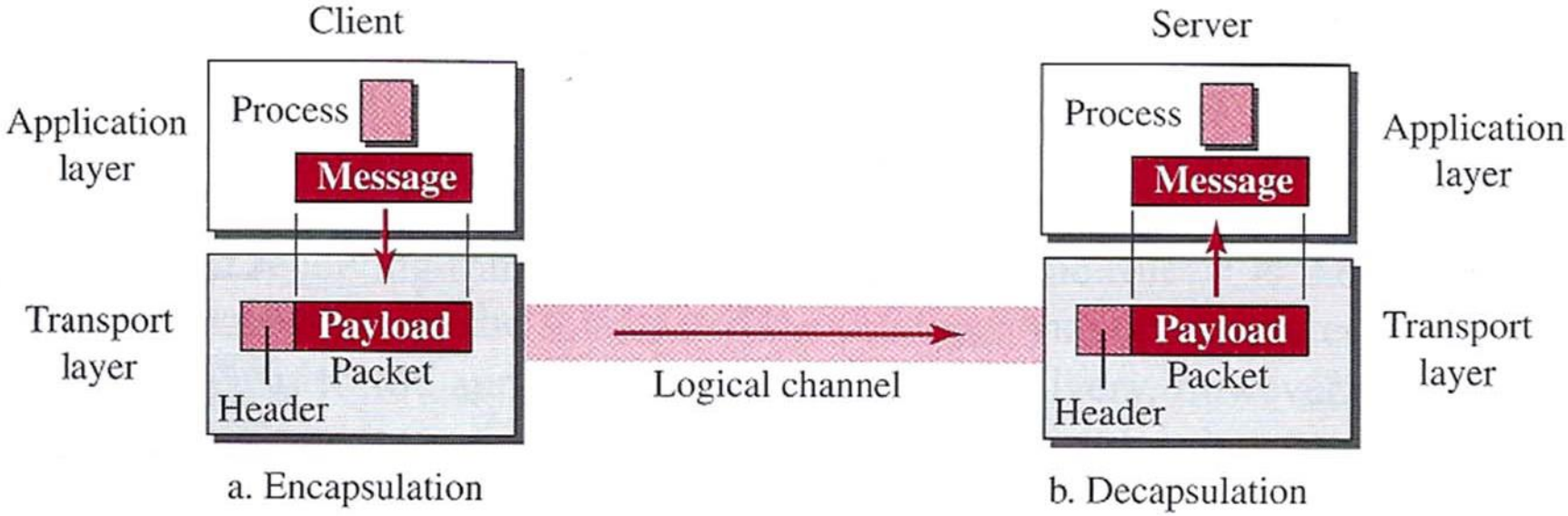


Can also be written as: 200.23.56.8:69

Transport Layer

Encapsulation and decapsulation

In order to be able to send a message from one process to another, the transport layer protocol has to encapsulate and decapsulate the messages.



- Encapsulation happens at the sender site. When a process wants to send a message, it is delivered to the transport layer with a pair of socket addresses (sender/receiver) and some other information depending on the transport layer's protocol.
- Decapsulation happens at the receiver site. When the packet arrives at the receiver. The header of the packet is removed, and data is delivered to the process specified by the port address. The sender socket address is also handed over to the process in case it needs to respond to the received message.

Transport Layer

Multiplexing and demultiplexing

Multiplexing

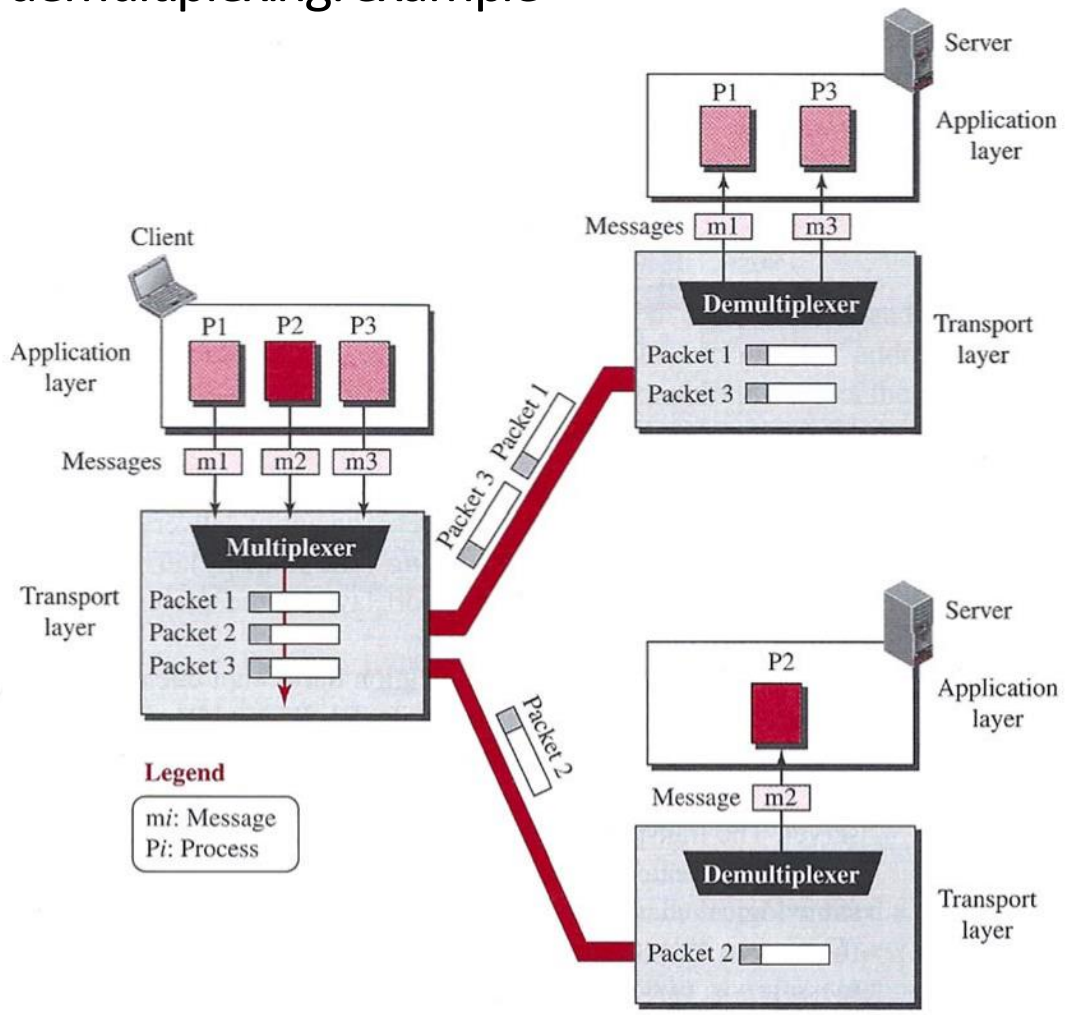
On the sender side, there can be many processes that need to send packets. However, only one transport layer protocol is available. This is called a ***many-to-one*** relationship and requires **multiplexing**.

Demultiplexing

On the receiver side, the transport layer protocol accepts messages from different source processes and distributes them to the correct destination processes according to their port numbers. This is a ***one-to-many*** relationship and requires **demultiplexing**.

Transport Layer

Multiplexing and demultiplexing: example



Three client processes run on the client host: P1, P2 and P3

- P1 and P3 send requests to their corresponding server processes running on a server
- P2 sends a request to its corresponding server processes running on another server

Transport Layer

Flow control

Whenever an entity produces items and another entity consumes them, there should be a balance between production and consumption rates.

If the items are produced faster than they can be consumed, the consumer can be overwhelmed and may need to discard some items.

Flow control is needed to prevent losing the data items at the consumer site.

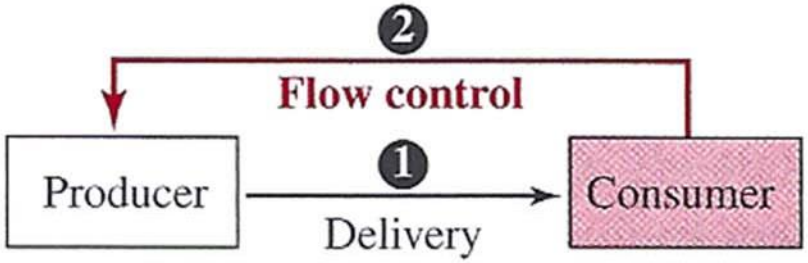
Transport Layer

Flow control: Pushing or pulling

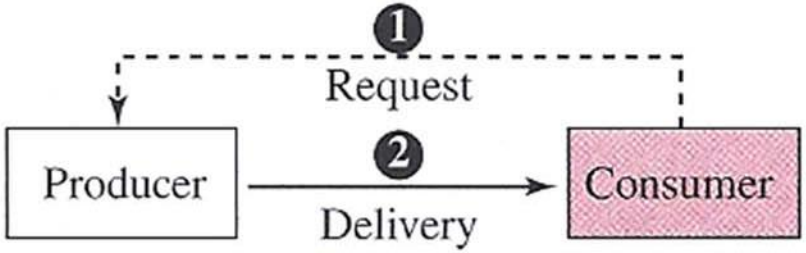
Delivery of data from producer to consumer can take place in two ways:

Pushing: If the producer/sender delivers data whenever it is produced, without a prior request from the consumer/receiver.

Pulling: If the producer delivers data only when the receiver first requests them.



a. Pushing



b. Pulling

When the producer pushes the items, the consumer may be overwhelmed and there is a need for flow control, in the opposite direction, to prevent discarding of the items.

However, there is no need for flow control when the consumer pulls the items (since it requests when it is ready).

Transport Layer

Flow control

When we deal with communication at the transport layer, we are dealing with four entities:

- Sender process
- Sender transport layer
- Receiver process
- Receiver transport layer

The sender process at the application layer is only a producer. It produces messages and passes them on to the transport layer.

The sender transport layer has a dual role:

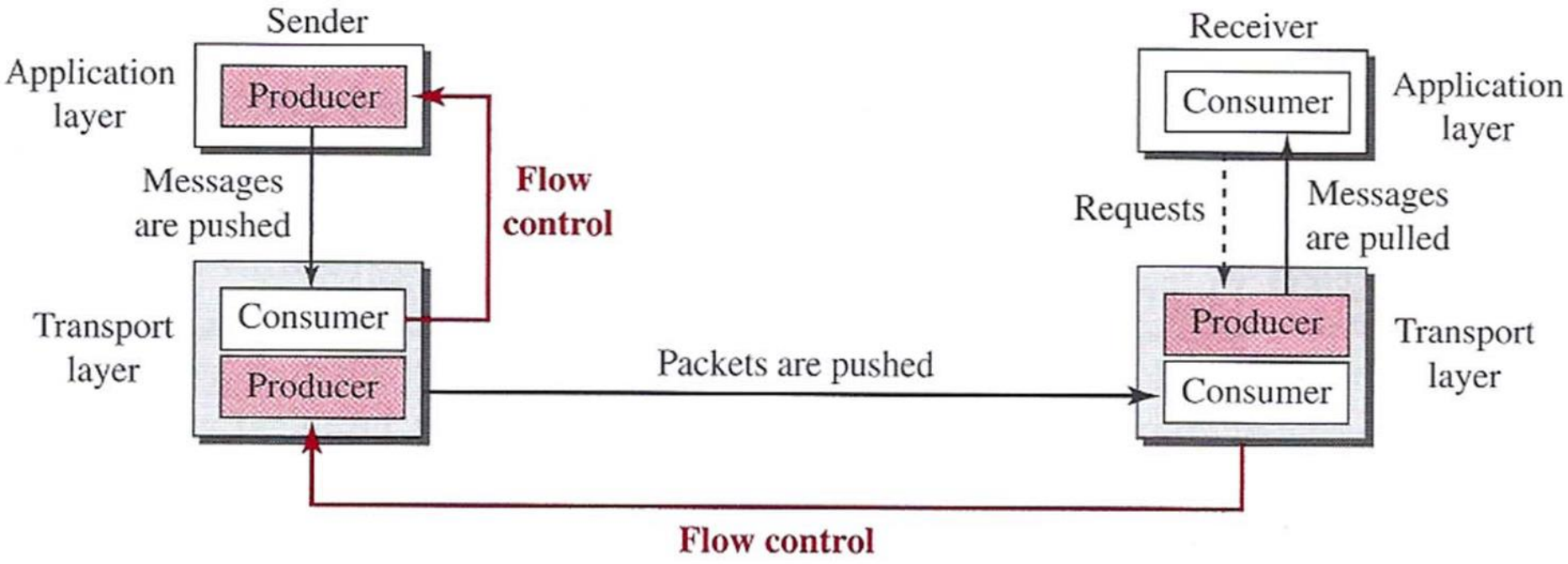
- It is consumer - packets are received from sender application layer through ***pushing***.
- It is producer – packets are encapsulated and sent to the receiver transport layer through ***pushing***.

The receiver transport layer also has a dual role:

- It is consumer – packets received from the sender transport Layer through ***pushing***.
- It is producer – packets are decapsulated and delivered to the receiver application layer through ***pulling***.

Transport Layer

Flow control



So, we need the flow control in at least the first two phases of this procedure:

- from the sending transport layer to the sending application layer.
- from the receiving transport layer to the sending transport layer.

Transport Layer

Flow control: Buffers

Although flow control can be implemented in several ways, two buffers will often be used:

- One at the sender transport layer.
- One at the receiver transport layer.
- When the buffer at the sender transport layer is full, the application layer is informed to stop delivering messages.
- When the buffer has some vacancies, the application layer is informed that it can continue the delivery of messages.
- When the buffer at the receiver transport layer is full, the sender transport layer is informed to stop delivering messages.
- When the buffer has some vacancies, the sender transport Layer is informed that it can continue the delivery of messages.

One could implement ***one ring-buffer*** (on the sender and receiver sides) for this flow control.

Transport Layer

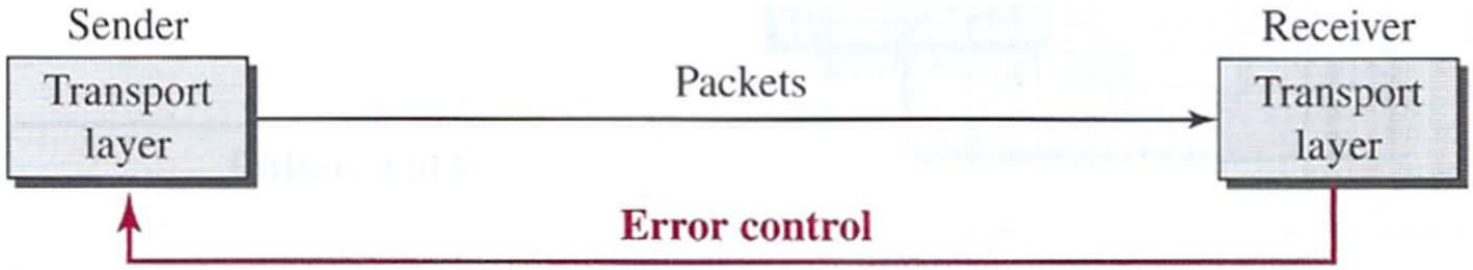
Error Control

Since the network layer is unreliable (IP protocol), we need to make the transport layer reliable if an application requires reliability.

This reliability can be e.g., achieved by adding an ***error control service*** to the transport layer. Error control at transport layer's level is responsible for:

1. Detecting and discard corrupted packets.
2. Keeping track of lost and discarded packets and resending them.
3. Recognizing duplicate packets and discarding them.
4. Buffering out-of-order packets until the missing packets arrive.

Error control only involve the sender and receiver transport layers
(as opposed to the flow control we just looked at)



As with the case of flow control, the receiving transport layer manages error control, most of the time, by informing the sending transport layer about the problems.

Transport Layer

Error Control: Sequence numbers

Error control requires that:

- The sender transport layer knows which packets is to be resent.
- The receiver transport layer knows which packet is a duplicate, or which packet has arrived out of order.

This can be done if the packets are numbered!

We add a ***sequence number*** in a field in the transport layer packet.

With this sequence number, the above error control can be performed.

- The packet numbering takes place sequentially (i.e., consecutively).
- The sequence number field sets a limit on the size of the sequence numbers.
- When the maximum number is reached, we can wrap around the sequence. (the sequence numbers are modulo 2^m and m is the number of the bits in the field.)

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...

Transport Layer

Error Control: Acknowledgment

- The receiver can send receipts (ACK) back to the sender when error-free packets are received.
- The sender starts a timer when a packet is sent. If the sender does not receive this receipt before the timer expires, the packet will be resent.
- If the receiver had sent a receipt that did not arrive in time, the receiver will receive the resent duplicate package (which can be just discarded silently based on the sequence number by the receiver)

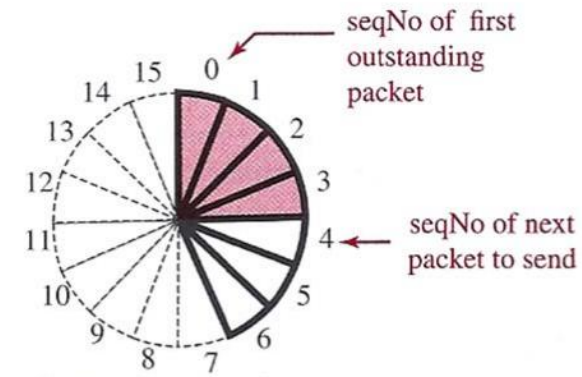
Combination of Flow and Error Control

Sequence numbering and acknowledgement can be combined if we use two numbered buffers on the sender and receiver sides, which place packets in the buffer in relation to the sequence number.

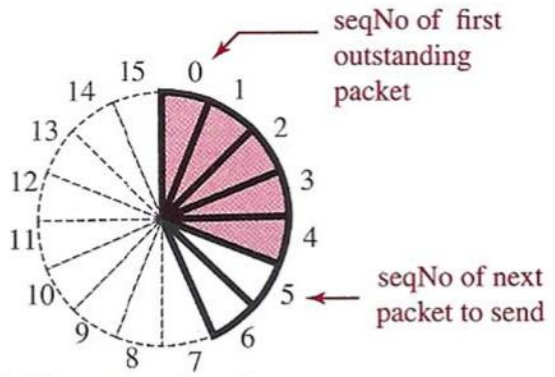
Transport Layer

Error Control: Sliding Window

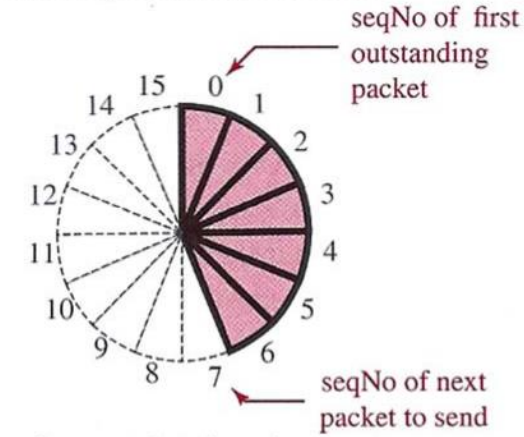
Since the sequence numbers use modulo 2^m, a circle can represent sequence numbers from 0 to $2^m - 1$. The buffer is represented as a set of slices, called **sliding window**, that occupies part of the circle at any time.



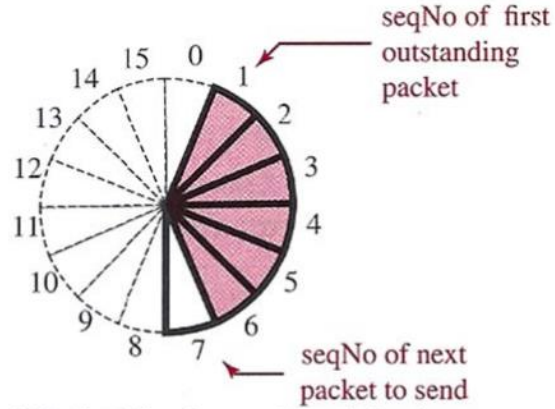
a. Four packets have been sent.



b. Five packets have been sent.



c. Seven packets have been sent; window is full.

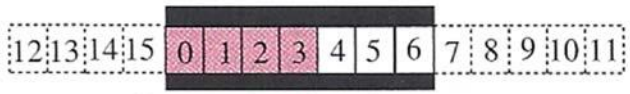


d. Packet 0 has been acknowledged; window slides.

The sequence numbers are in modulo 16 and the size of the window is 7

Transport Layer

Error Control: Sliding Window (in linear representation)



a. Four packets have been sent.

b. Five packets have been sent.

c. Seven packets have been sent; window is full.

d. Packet 0 has been acknowledged; window slides.

- The leftmost packets are acknowledged (12 - 15), the sender no longer has copies of them. They are finished.
- The packets in the shaded field of the window, the sender needs to have copies of them, as they have been sent, but their fate is not known yet, we do not know if they should be resent or not.
- The white packets in the window are ready to be sent, but data has not yet been handed over to the network layer.

As receipts received, the window slides to the right.

The size of the window thus depends on how many packets of which the sender can keep copies.

Transport Layer

Congestion control

- An important issue in a packet-switched network, such as Internet, is ***Congestion***.
- This occurs when the number of packets (the load) sent to the network is greater than the capacity of the network.
- Congestion control refers to a mechanism/technique that controls congestion and keeps the load below capacity.

Congestion in a network occurs because routers and switches have queues--buffers that hold the packets before and after processing. **Congestion at the transport layer is actually the result of congestion at the network layer, which manifests itself at the transport layer.**

Later we will see how the TCP protocol (assuming that there is no congestion control at the network layer) makes its own congestion control mechanism.

Connectionless and Connection-Oriented Protocols

Transport Layer

Connectionless and Connection-oriented service

In a **Connectionless service**, packets are sent without the need of establishing a logical connection at the beginning and closing it in the end.

- Packets are not numbered,
- or can be lost,
- or arrive at the receiver ”out-of-order”,
- There is no receipt for received packets,
- **UDP** protocol is an example of such a "Connectionless service".

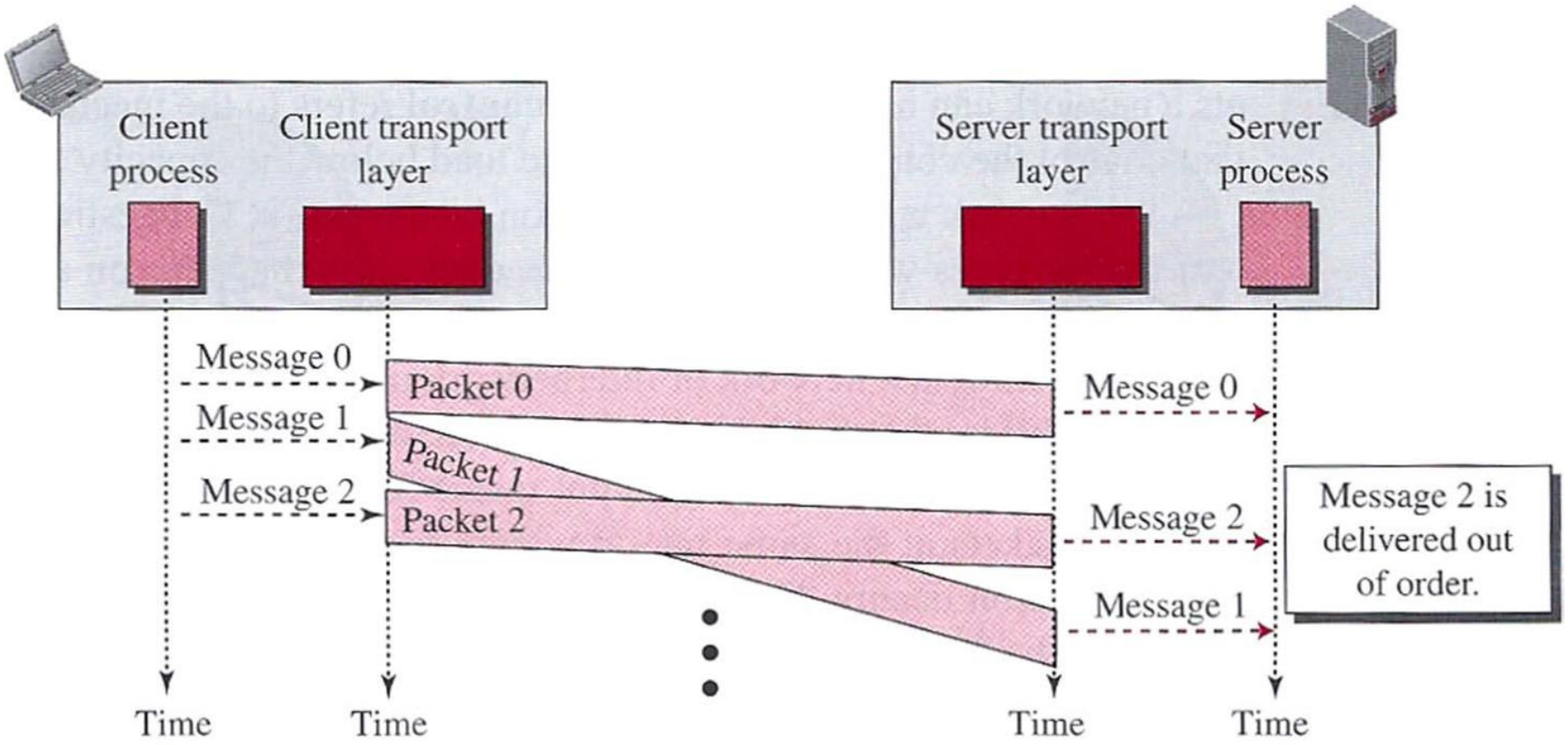
In a **Connection-oriented service**, there must be:

- first, a logical connection is established between sender and receiver,
- Then data is sent,
- and the logical connection is closed in the end.
- **TCP** and **SCTP** are examples of ”Connection-oriented service”.

Transmission Control Protocol (**TCP**)
User Datagram Protocol (**UDP**)
Stream Control Transmission Protocol (**SCTP**)

Transport Layer

Connectionless service: example

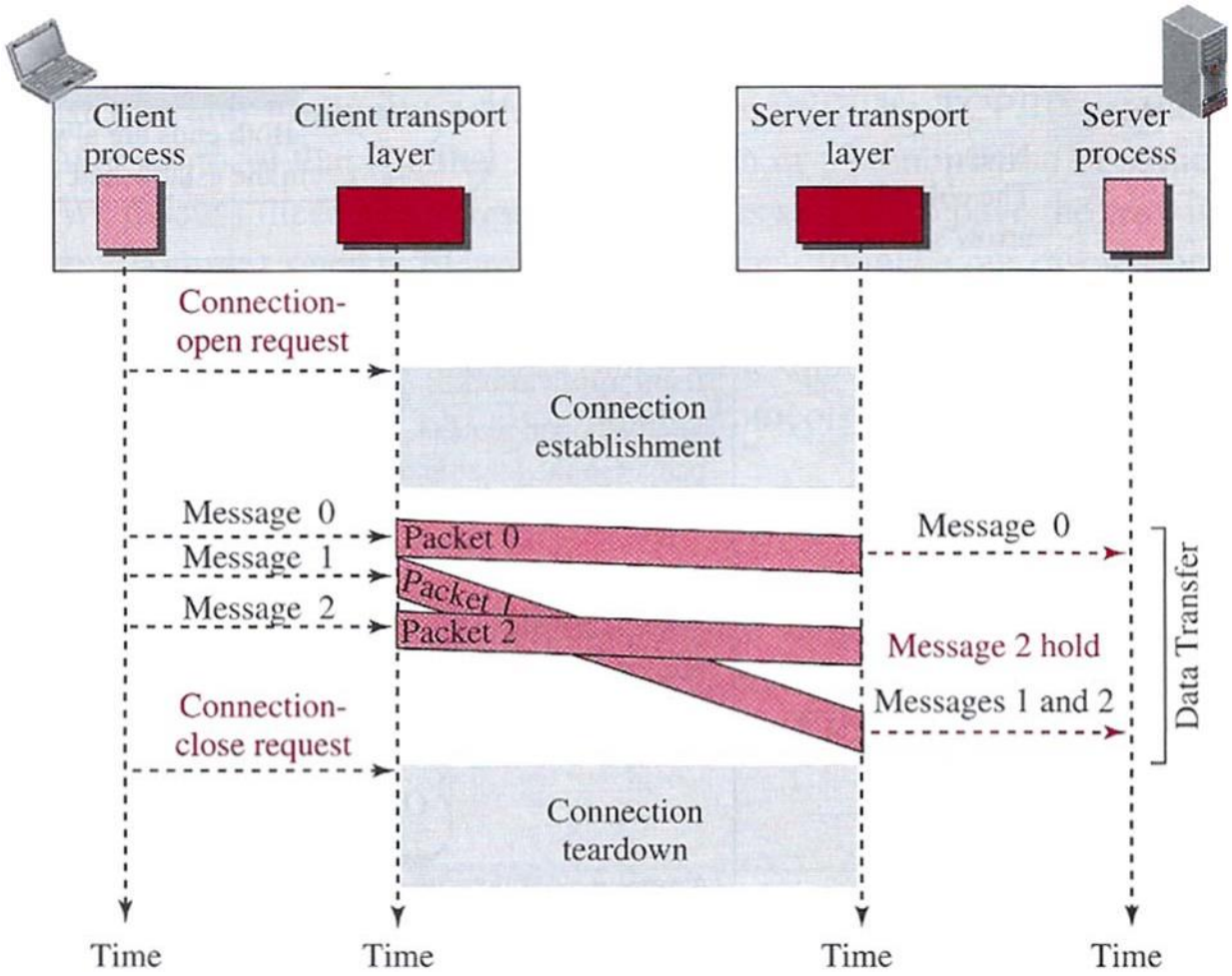


Here we can e.g., see that the three packets do not arrive at the server host in order because of the extra delay in transportation of the second packet.

No flow control, error control, or congestion control can be effectively implemented in a connectionless service.

Transport Layer

Connection-oriented service: example



A connection is established first and packets are delivered to the receiver process in the right order thanks to the coordination of the two host at the transport layer.

Flow control, error control, and congestion control can be implemented in a connection-oriented protocol.