Institute of Electrical and Electronic Engineering, University M'Hamed BOUGARA of Boumerdes

# Chapter 7
# Transport Layer

by Hadjira BELAIDI

**Objectives**

After completing this chapter, you will be able to:

- ✓understand principles behind transport layer services:
  - ✓multiplexing/demultiplexing
  - ✓reliable data transfer
  - ✓flow control
  - ✓congestion control

- ✓Instantiation and implementation in the Internet

# Chap7 Outlines

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP and
  - TCP

# Types of data deliveries

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
    - UDP
    - TCP

*The transport layer is responsible for process-to-process delivery*

*—The delivery of a packet, part of a message, from one process to another.*

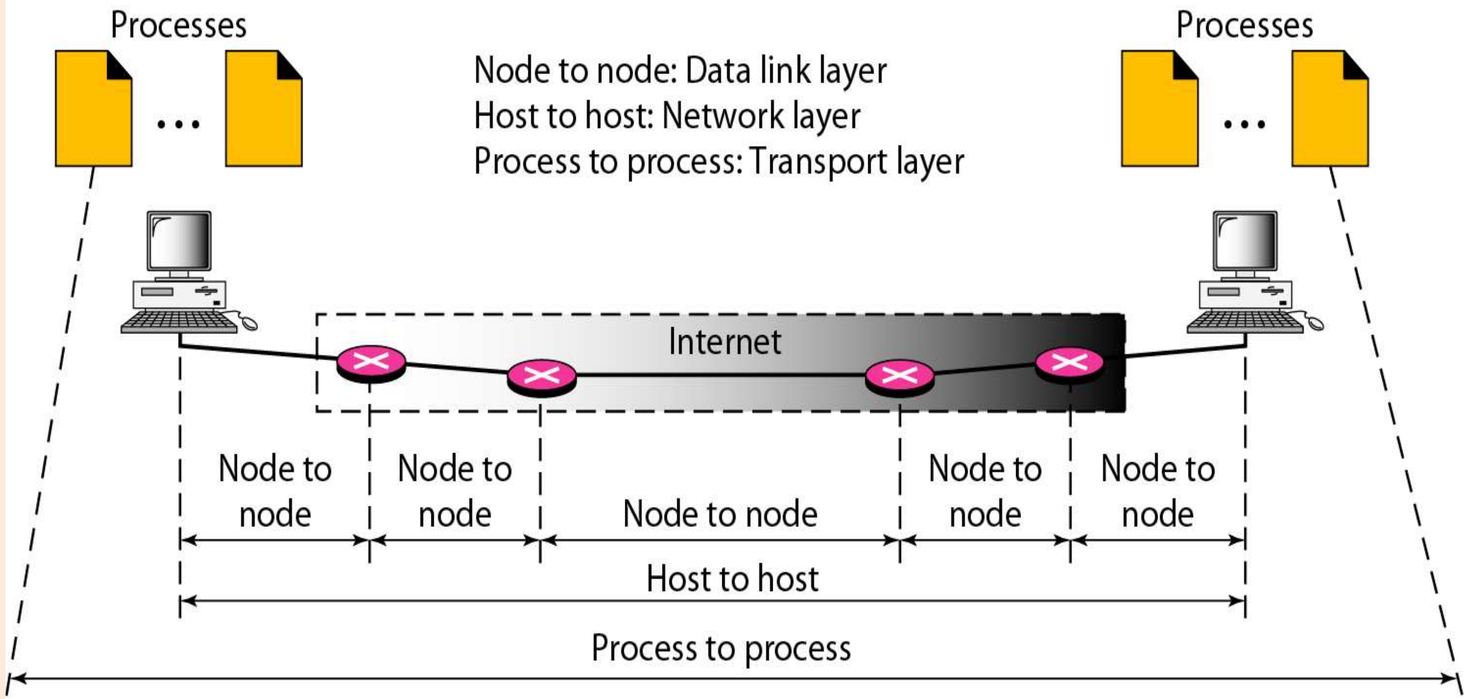
*—Two processes communicate in a client/server relationship,*

In Client/Server communication, four entities must be defined:

- Sending Node
    - Local Host IP
    - Local Process Port number
- Receiving Node
    - Remote host IP
    - Remote Process ID Port number

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
    - UDP
    - TCP



# Transport layer vs Network layer

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

**Transport Layer**

❑ logical communication between **processes**

❑ moves messages from application process to the network layer and vice-versa: Sending & Receiving sides

❑ computer network can make multiple transport layer protocols available
- TCP
- UDP

❑ **process-to-process communication**

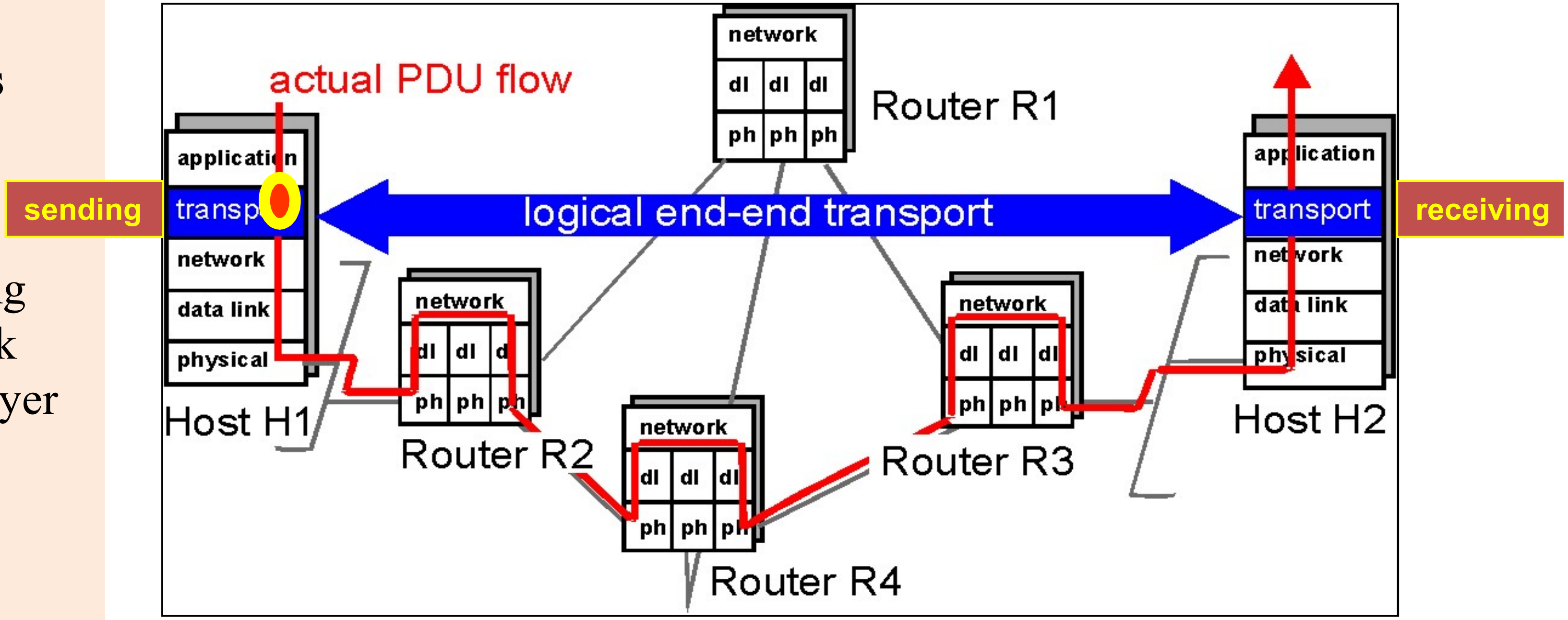
**Network Layer**

❑ logical communication between **end systems**

❑ **host-to-host communication**

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP



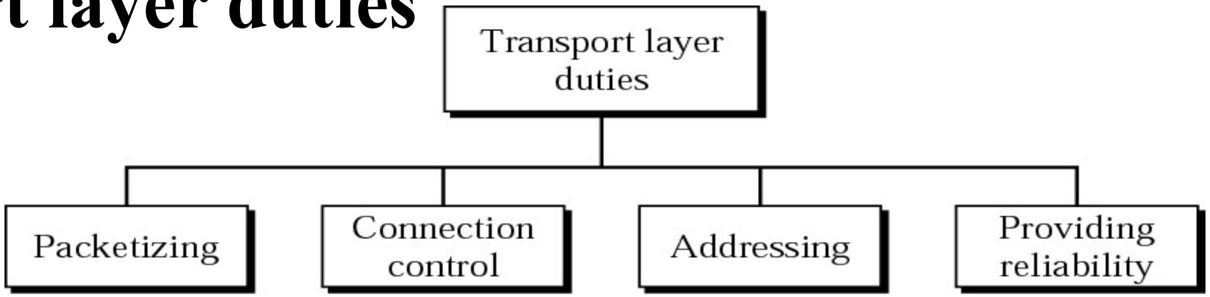

- **converts messages to 4-PDUs**

  **Breaks down application messages into smaller chunks + transport layer header = 4-PDUs**

- **Network Layer: Each 4-PDU encapsulated into a 3-PDU**

- **receives 4-PDUs**
- **removes transport header**
- **reassembles the messages**
- **passes to receiving application process**

# Transport layer duties




**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

- Packetizing
  - Sender side: breaks application messages into segments, passes them to network layer
  - Transport layer at the receiving host deliver data to the receiving process
- Connection control
  - Connection-oriented
  - Connectionless
- Addressing
  - Port numbers to identify which network application
- Reliability
  - Flow control
  - Error Control

# 1. Processes communicating across network

## 1.1. Sockets

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
    - UDP
    - TCP

- Process is an instance of a program in execution.
- Processes on two hosts communicate with each other by sending and receiving messages
- The process receives messages from, and sends messages into the network through its socket
- A socket is the interface between the application layer and the transport layer within a host.
- Sockets are the **programming interface** used to build network applications over the internet.
- Programmers can select which transport layer protocol (UDP or TCP) to be used by the application and select few transport-layer parameters (maximum buffer size, Maximum segment size, starting sequence number of segment).

**Outlines**

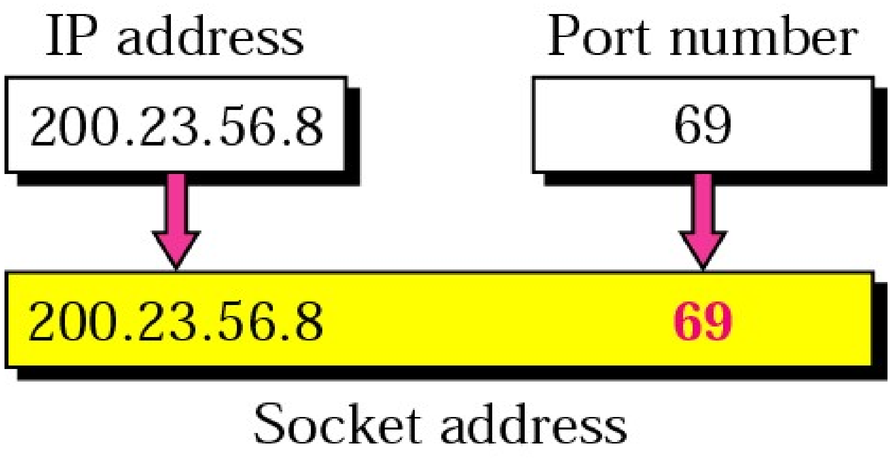
- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

Transport layer at the receiving host delivers data to the socket

There should be a unique identifier for each socket.

Socket identifier is called socket address

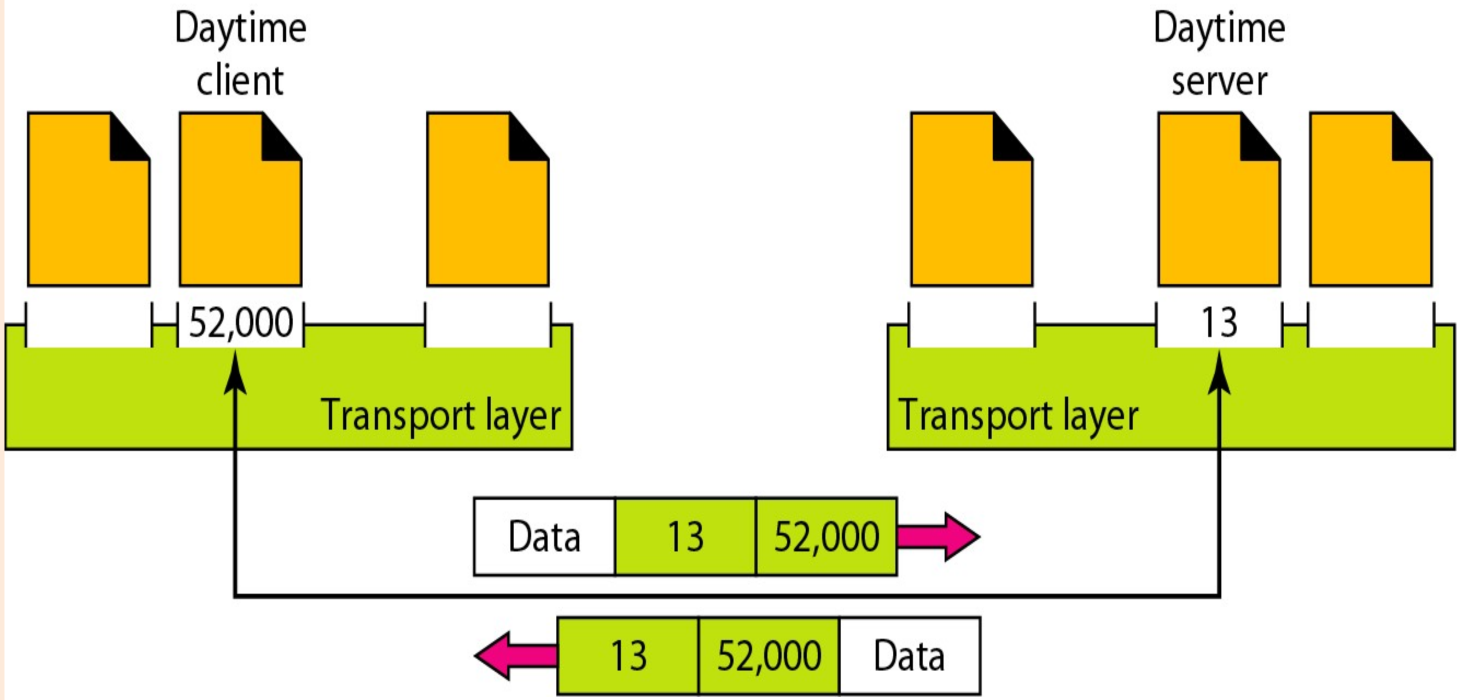
Socket address = IP address & Port number



# Example

**Outlines**

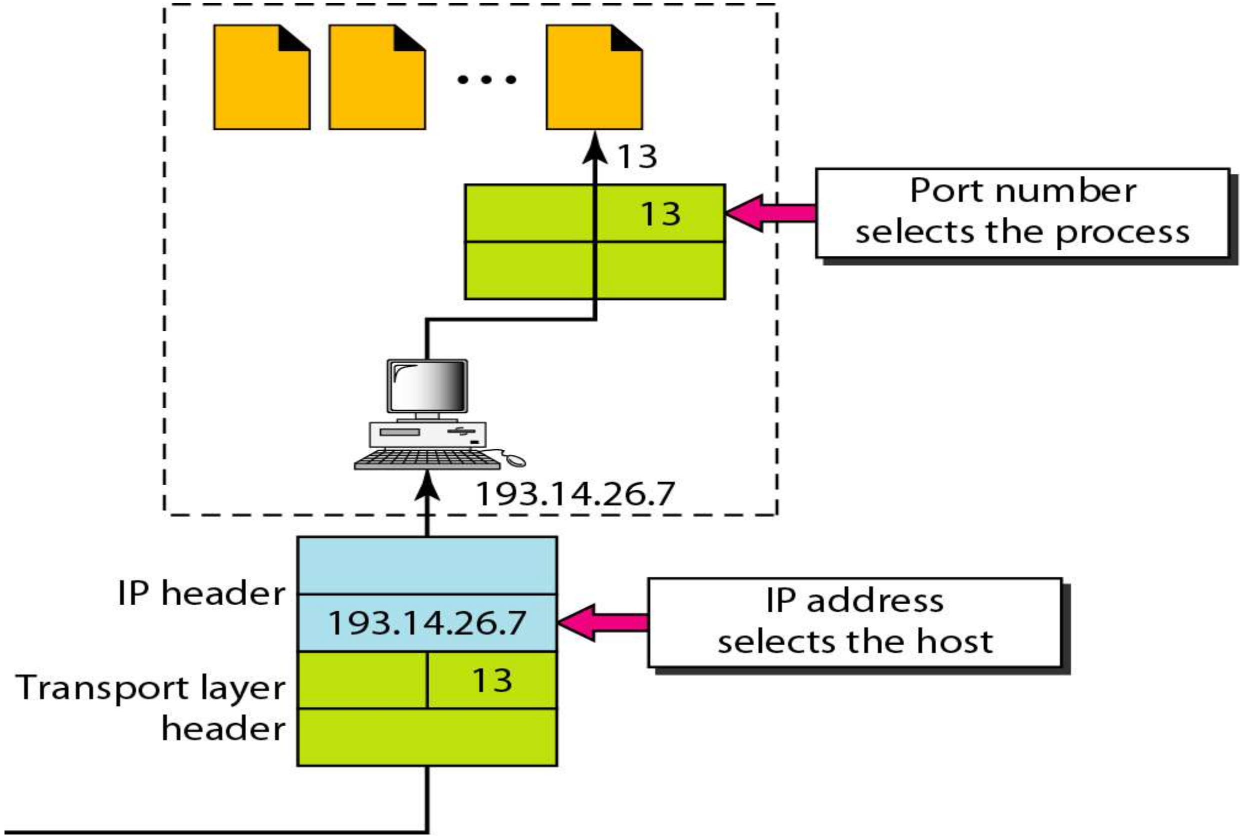
- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP



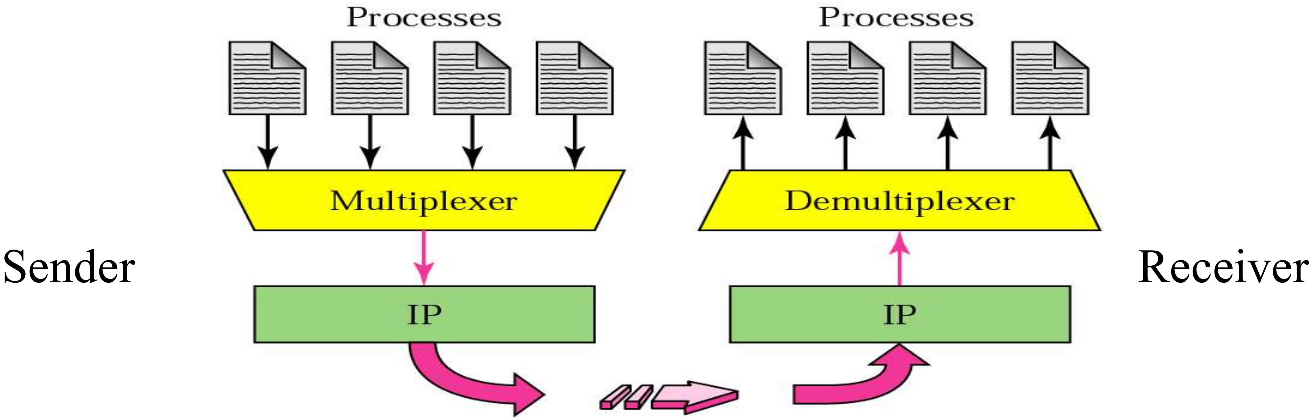
**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

# Process-to-Process delivery needs IP address and Port number



# 1.2. Multiplexing and demultiplexing




Multiplexing: (at the sending node) The process of encapsulating data messages from different applications sockets with the header information and pass the segments to the network layer.

DeMultiplexing: (at the receiving node) The process of delivering the received data segment to the correct application.

- Example:
- Suppose that the following is running on the same computer:
  - Downloading a web page while transferring data through FTP
  - Two telnet sessions are also running
  - Transport layer receives TPDUs from network layer for all four processes

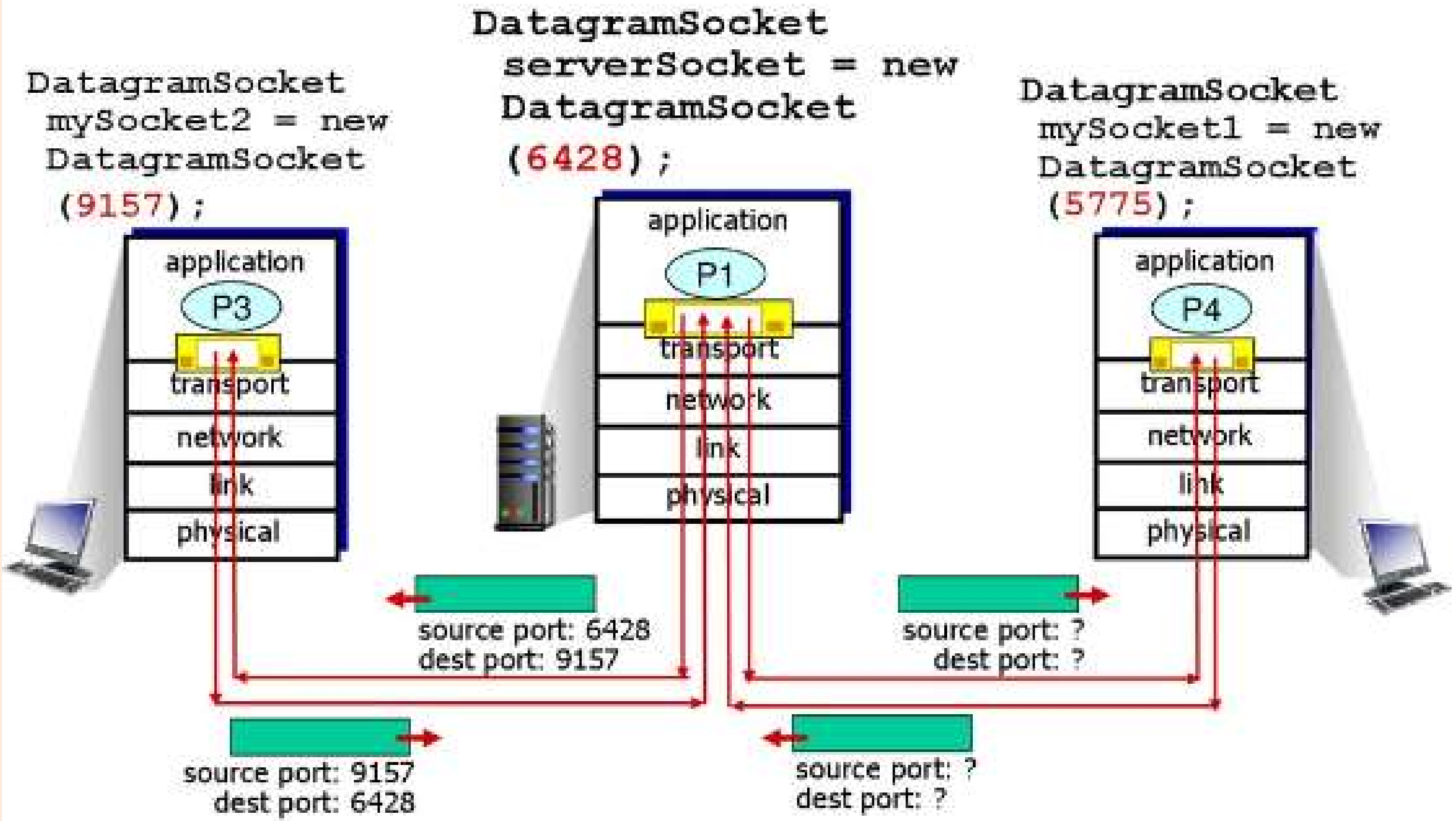
**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

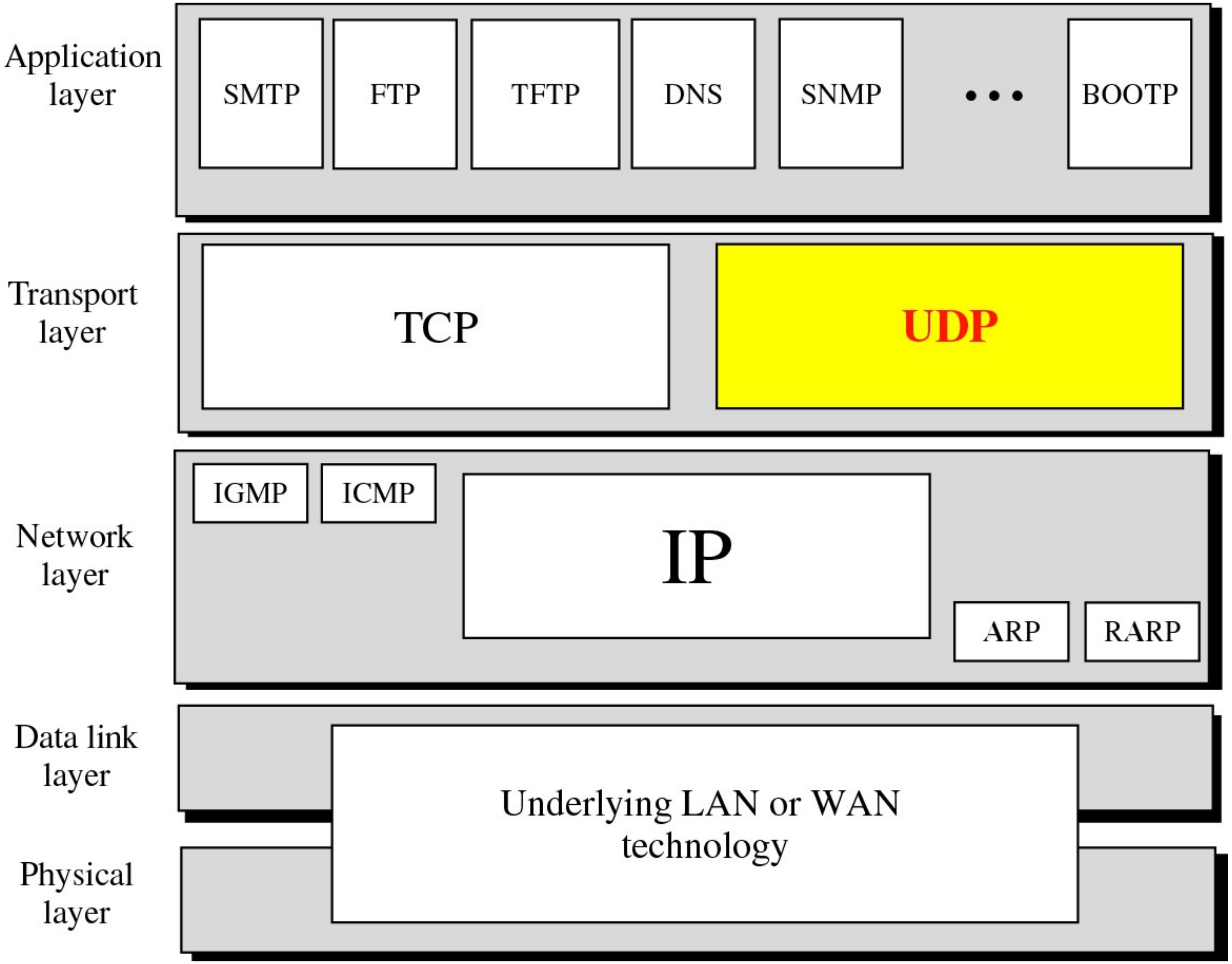
# *Example*



**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - TCP and
  - UDP

# 2. Transport layer protocols



**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
    - UDP
    - TCP

# 2.1. User Datagram Protocol (UDP).

The User Datagram Protocol (UDP) is a transport layer protocol defined for use with the IP network layer protocol.

It provides a best-effort datagram service to an End System (IP host).

The service provided by UDP is an unreliable service that provides no guarantees for delivery and no protection from duplication (e.g. if this arises due to software errors within an Intermediate System (IS)).

The simplicity of UDP reduces the overhead from using the protocol and the services may be adequate in many cases.

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

- **Connectionless**
  - **No handshaking** between UDP sender, receiver
  - Each UDP segment handled **independently** of others
- A **server application** that uses UDP serves only **ONE request** at a time. All other requests are stored in a **queue** waiting for service.
- **Unreliable protocol has no flow and error control**
  - A UDP segment can be **lost**, **arrive out of order**, **duplicated**, or **corrupted**
  - **Checksum field checks error in the entire UDP segment. It is Optional**
  - UDP **does not do anything to recover** from an error it simply **discard** the segment ➔ Application accepts full responsibility for errors

**Outlines**

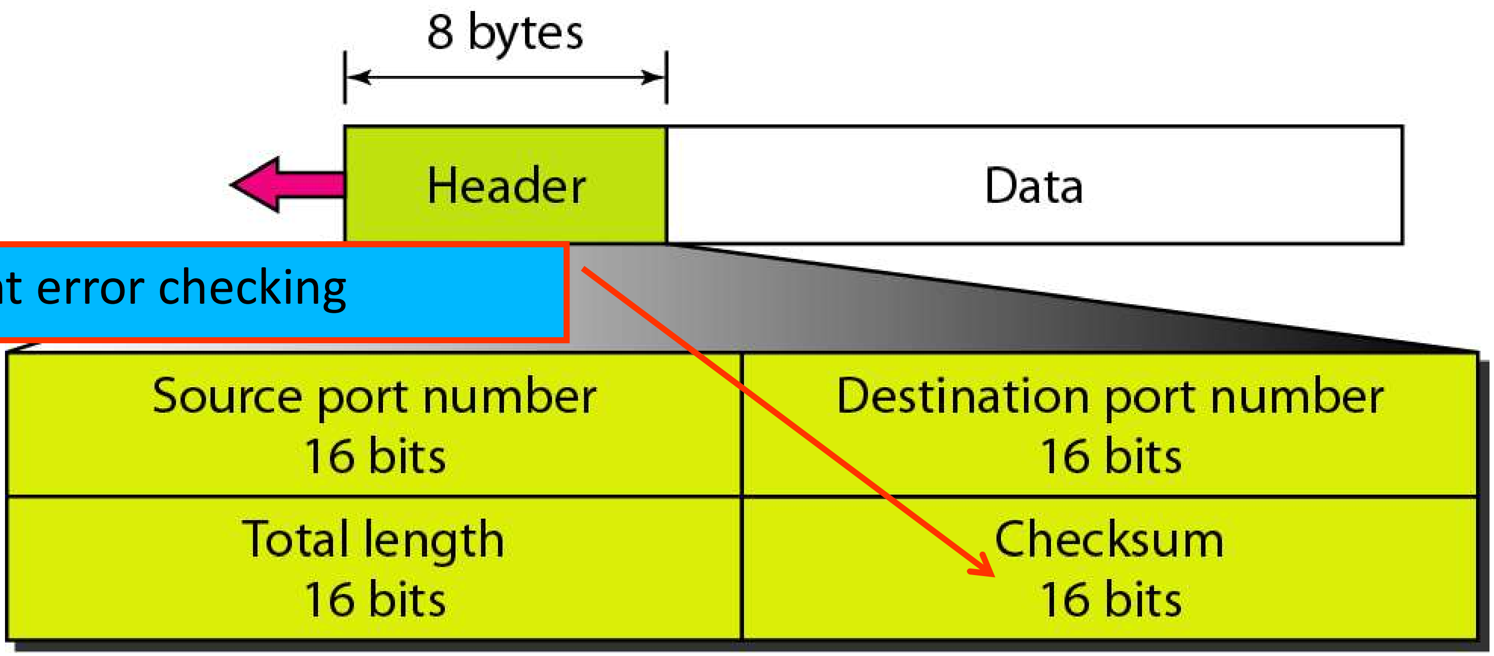


- It **uses port numbers** to multiplex/demultiplex data from/to the application layer.
- Advantages: Simple, **minimum overhead**, **no connection delay**
- **Services provided by UDP:**
  - Process-to-Process delivery
  - Error checking (however, if there is an error UDP does NOT do anything to recover from **error. It will just** discard the message.

**Outlines**

- Introduction
- Proce[...]
comm[...]
across network
- Transport Layer Protocols
    - UDP
    - TCP

# 2.1.1. User datagram format




Header size = 8 bytes

Minimum UDP process data size = 0 bytes

Maximum UDP process data size=

65535 – 20 (network layer headers) - 8 (UDP headres)= 65507 bytes

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

*Note*

UDP length

= IP length – IP header’s length

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

# UPD Checksum

Goal: detect "**errors**" (e.g., flipped bits) in transmitted segment

Sender:

- treat segment contents as sequence of **16-bit integers**
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?* More later ….

# UPD Checksum example

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

- Three packets of 16 bits each
  - 0110011001100110
  - 0101010101010101
  - 0000111100001111
- adding the three, calling it 'r':
  - 1100101011001010
- Send the four packets, the original three and 1's complement of 'r' to destination

- The 1's complement of 'r' is:
  - 0011010100110101
- at destination, the sum of four packets should be:
  - 1111111111111111
- **If the packet is damaged:**
  - 1111**0**11111111111 (**zeros!!**)

**Why provide for error checking?** ***No guarantee that it is provided in all of the links between source and destination***

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

# 2.1.2. UDP Applications

- Used for applications that can tolerate small amount of packet loss:
  - Multimedia applications,
  - Internet telephony,
  - real-time-video conferencing
  - Domain Name System messages
  - Audio
  - Routing Protocols

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

## 2.2. Transmission Control Protocol (TCP).

TCP (Transmission Control Protocol) is a standard that defines how to establish and maintain a network conversation via which application programs can exchange data. TCP works with the Internet Protocol (IP), which defines how computers send packets of data to each other. Together, TCP and IP are the basic rules defining the Internet.

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
    - UDP
    - TCP

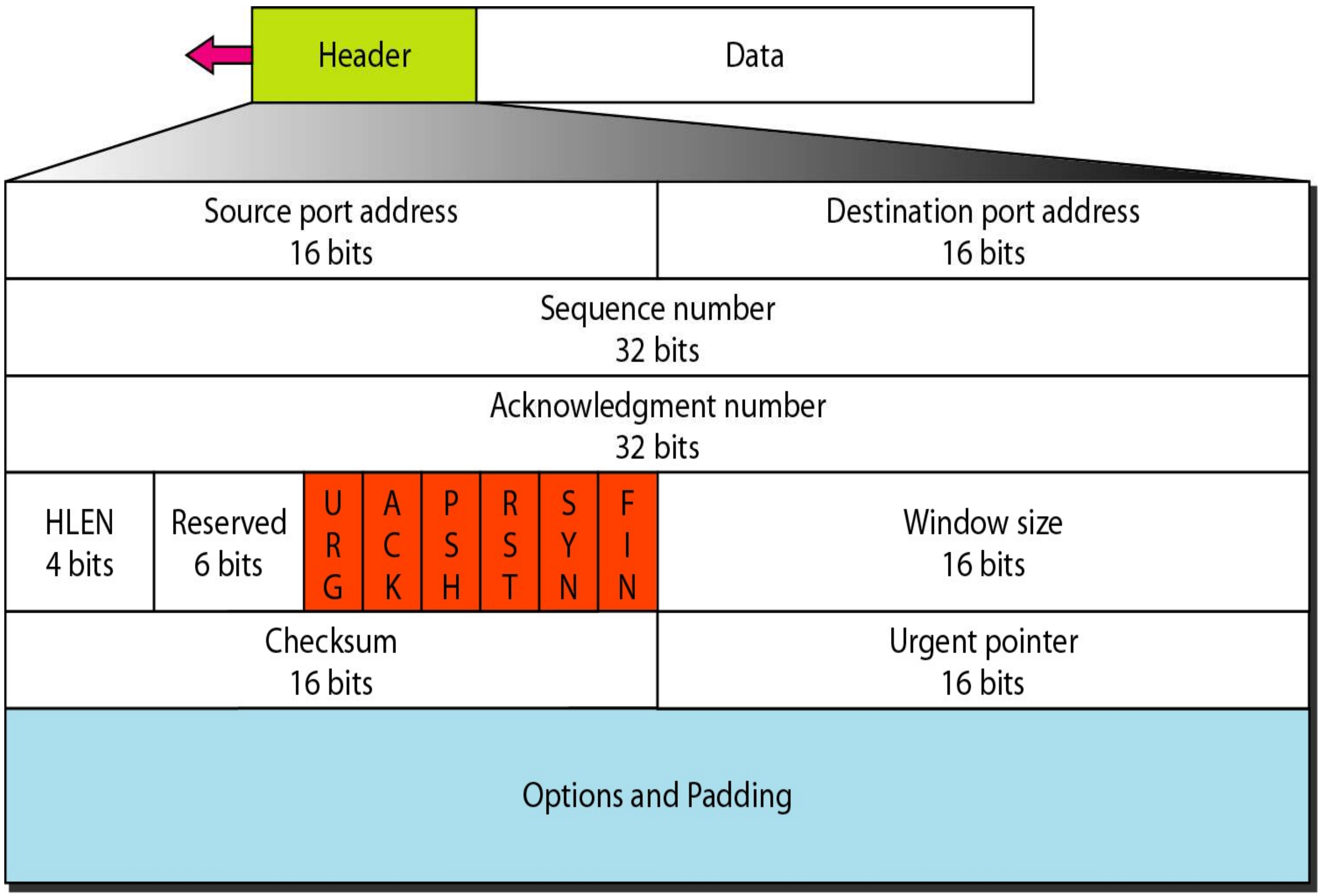
- TCP is a connection-oriented protocol, which means a connection is established and maintained until the application programs at each end have finished exchanging messages.

- It determines how to break application data into packets that networks can deliver,

- Sends packets to and accepts packets from the network layer,
- Manages flow control,
- And—because it is meant to provide error-free data transmission— handles retransmission of dropped or garbled packets as well as acknowledgement of all packets that arrive.

- In the Open Systems Interconnection (OSI) communication model, TCP covers parts of Layer 4, the Transport Layer, and parts of Layer 5, the Session Layer.

## *TCP segment format*

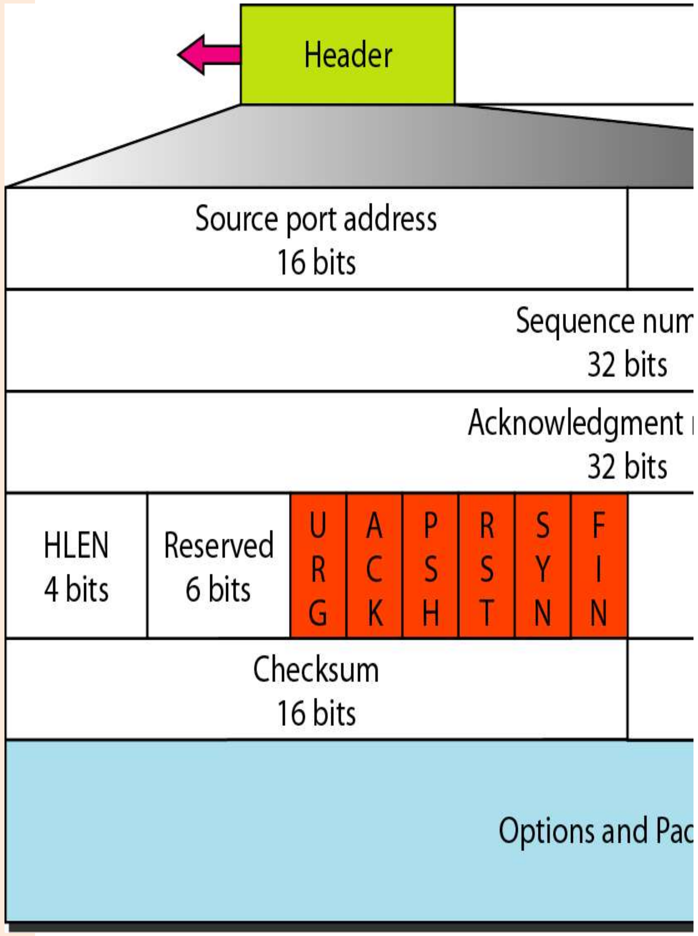



Minimum header length is 20 bytes and the maximum is 60 bytes when there are options

## Outlines

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

### *TCP segment format*




HLEN: Header Length

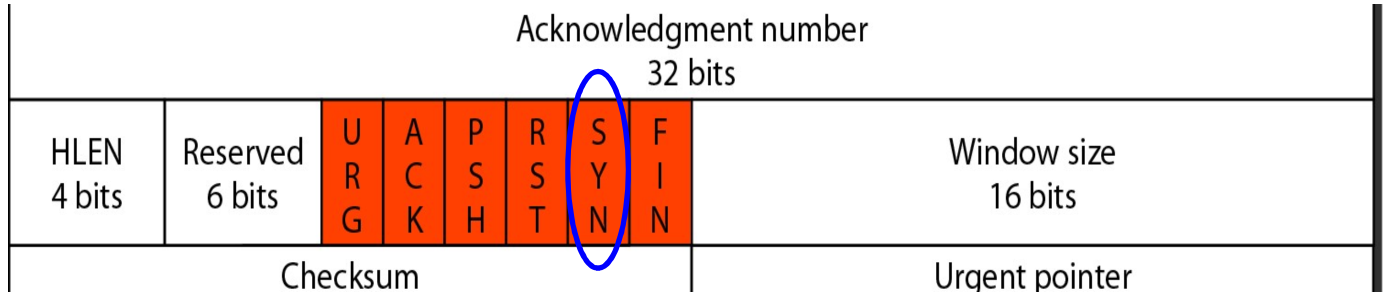
***Flags***

URG: urgent pointer field significant.
ACK: acknowledgment field significant.
PSH: push function.
RST: reset the connection.
SYN: synchronize the sequence numbers.
FIN: no more data from sender.

## Outlines

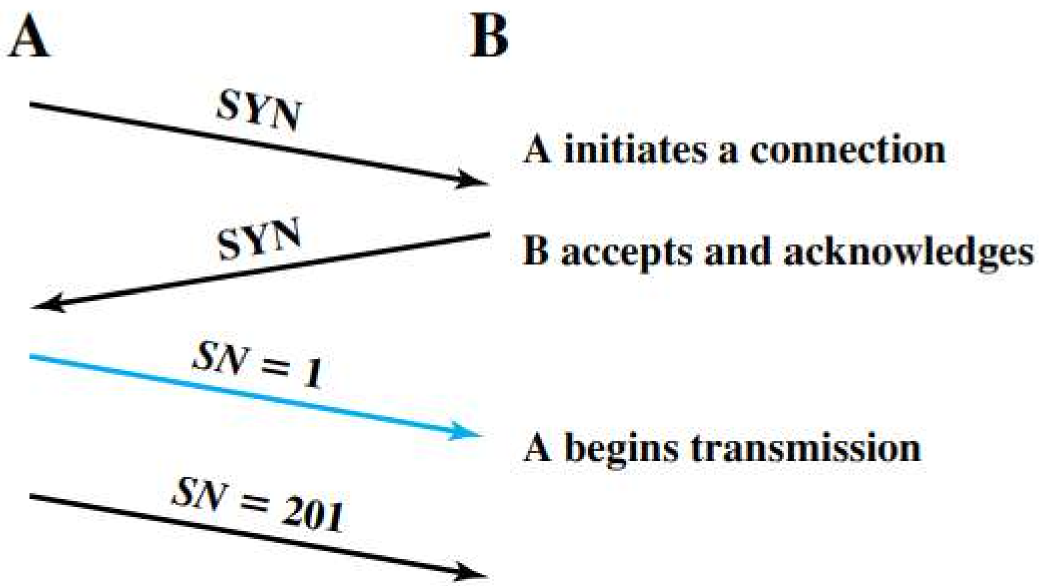
- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

## *Flags*




SYN: Synchronize

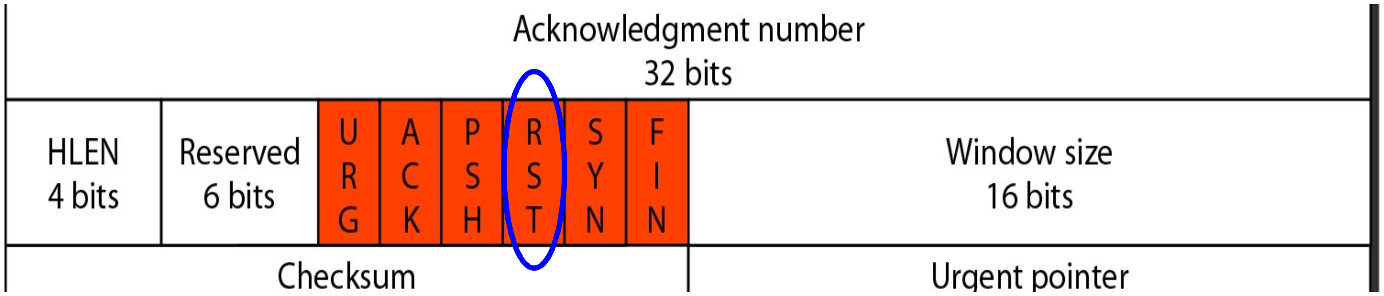
SYN=1 → request for connection



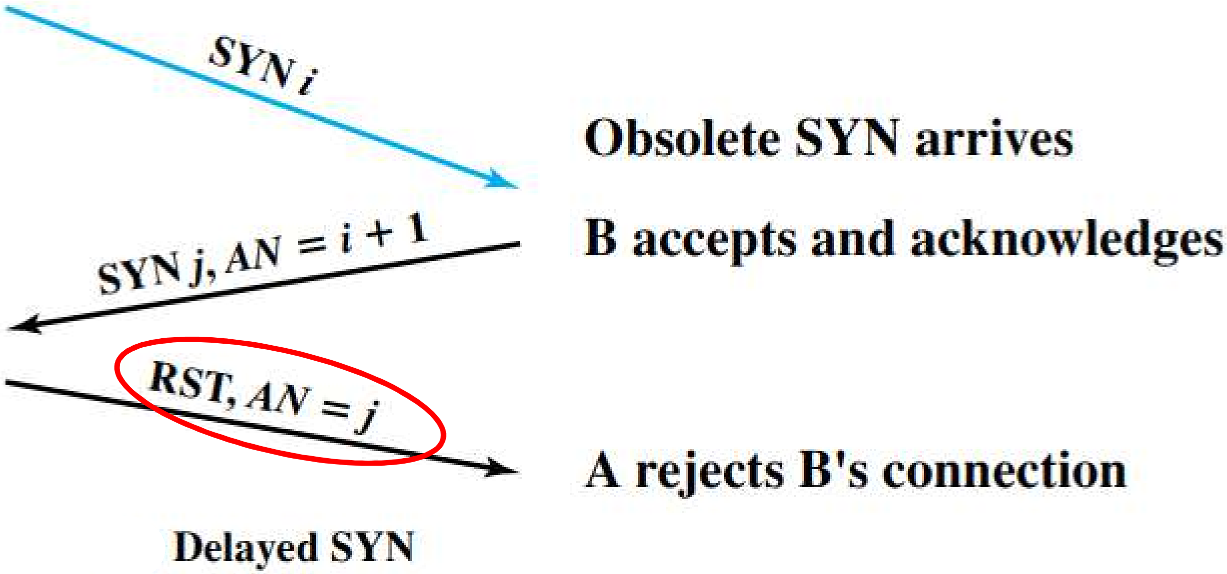
**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

## *Flags*



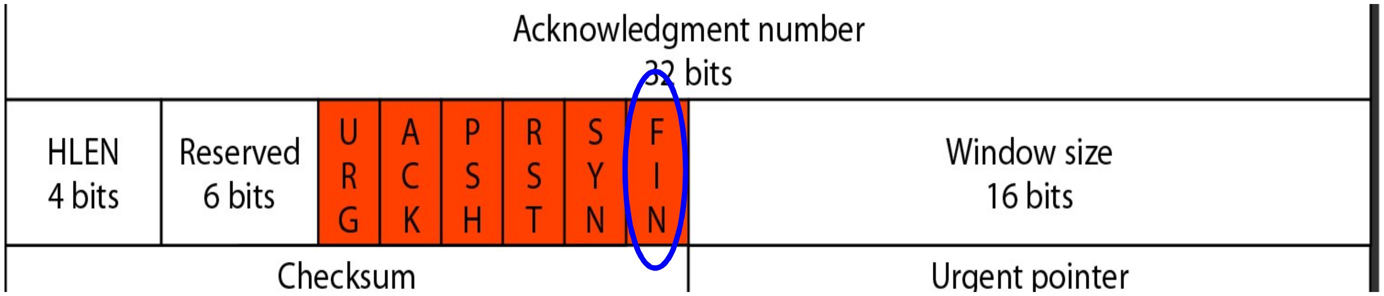

RST: Reset
RST=1 → Send an RST if the connection state is not yet OPEN and/or an invalid ACK



**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

***Flags***




FIN: Connection Termination

FIN=1 → close the connection

Each side must explicitly acknowledge the FIN of the other, using an ACK with the sequence number of the FIN to be acknowledged. For a graceful close, a transport entity requires the following:

- It must send a FIN $i$ and receive $AN = i + 1$
- It must receive a FIN $j$ and send $AN = j + 1$
- It must wait an interval equal to twice the maximum expected segment lifetime.

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

- Transmission Control Protocol properties:
  - Connection-oriented (establishment & termination)
  - Reliable
  - Full-duplex

## 2.2.1. Connection-Oriented

**Outlines**

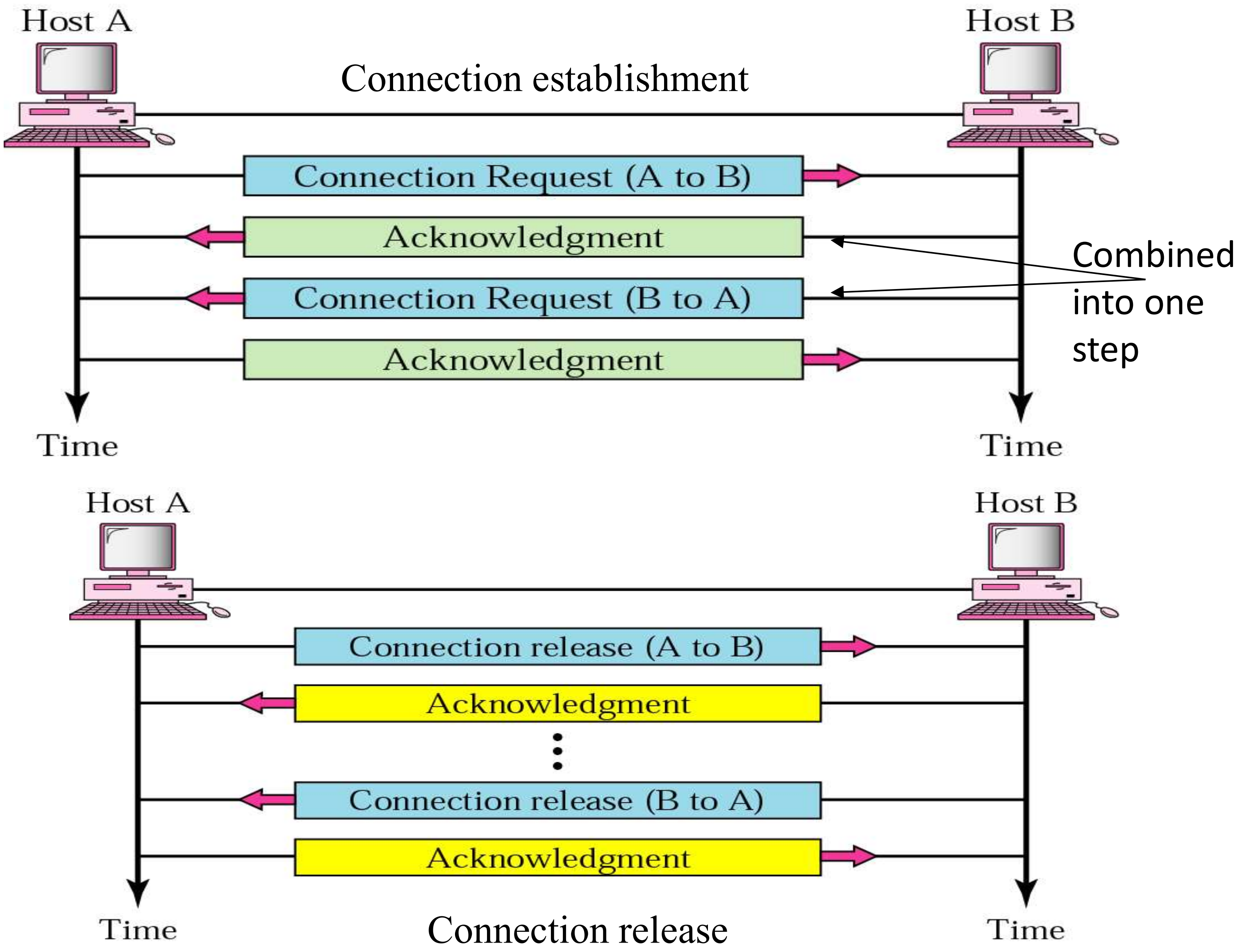
- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

- *Connection oriented* means that a virtual connection is established before any data is transferred.
- Connection ensures that the receiving process is available and ready **before** the data is sent.
- **Three-way handshaking connection** establishment procedure because TCP is full-duplex both side must initialize communication and get approval from the other side before any data transfer,
- Virtual connection since TCP protocol will make sure that segments are given to the receiver application in the same order as they were sent by the sender even if they travel through different physical paths.
- A server application that uses TCP can handle **many client** requests at the **same time** each has **its own connection**.

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

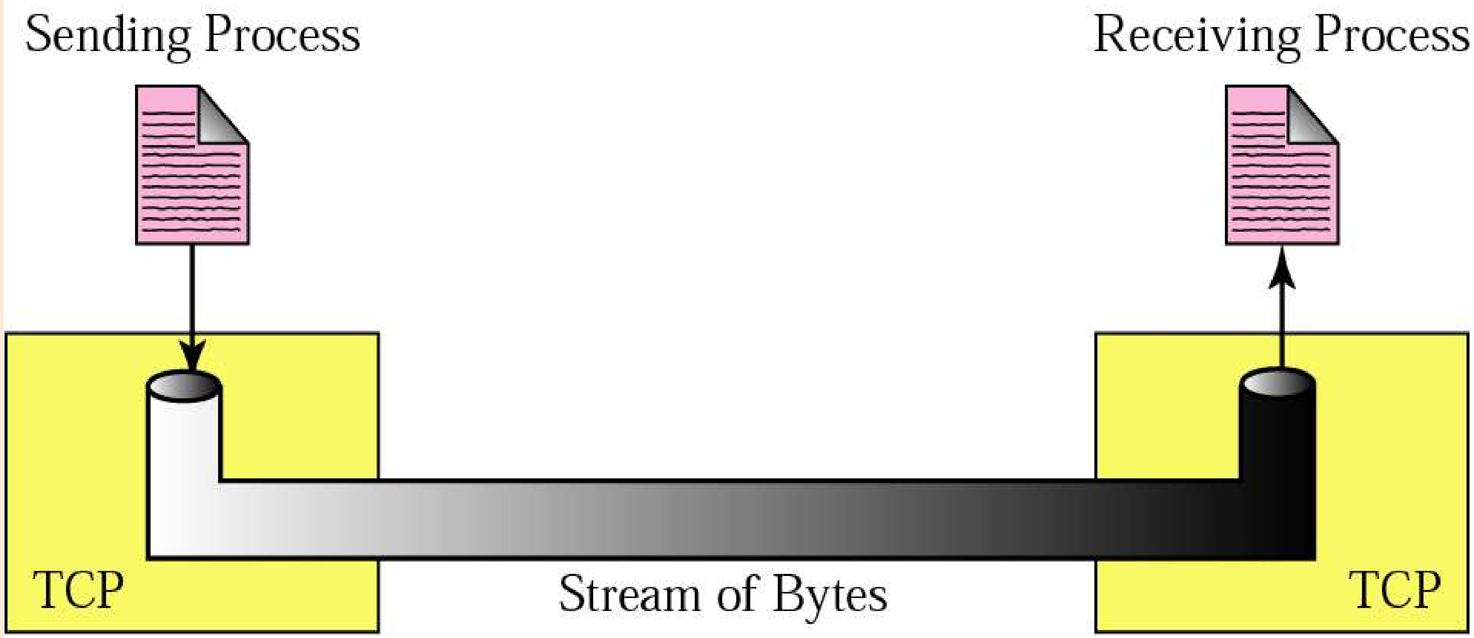
## Connection establishment and termination



**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
    - UDP
    - TCP

## TCP establishes a virtual connection




TCP will deliver segments to the applications in order and without error, lost, or duplicates

## Outlines



# 2.2.2. Full Duplex

- Data segments can flow in both directions at the same time.
- Each TCP connection has its own sending and receiving buffers.

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

## 2.2.3. Flow control and Reliability

- **Flow control** (process-to-process): TCP makes sure that the sender does not cause the receiver buffer to overflow
  - By defining the amount of data that can be sent before receiving an acknowledgement from the receiver **(sliding – window protocols)**

**Outlines**

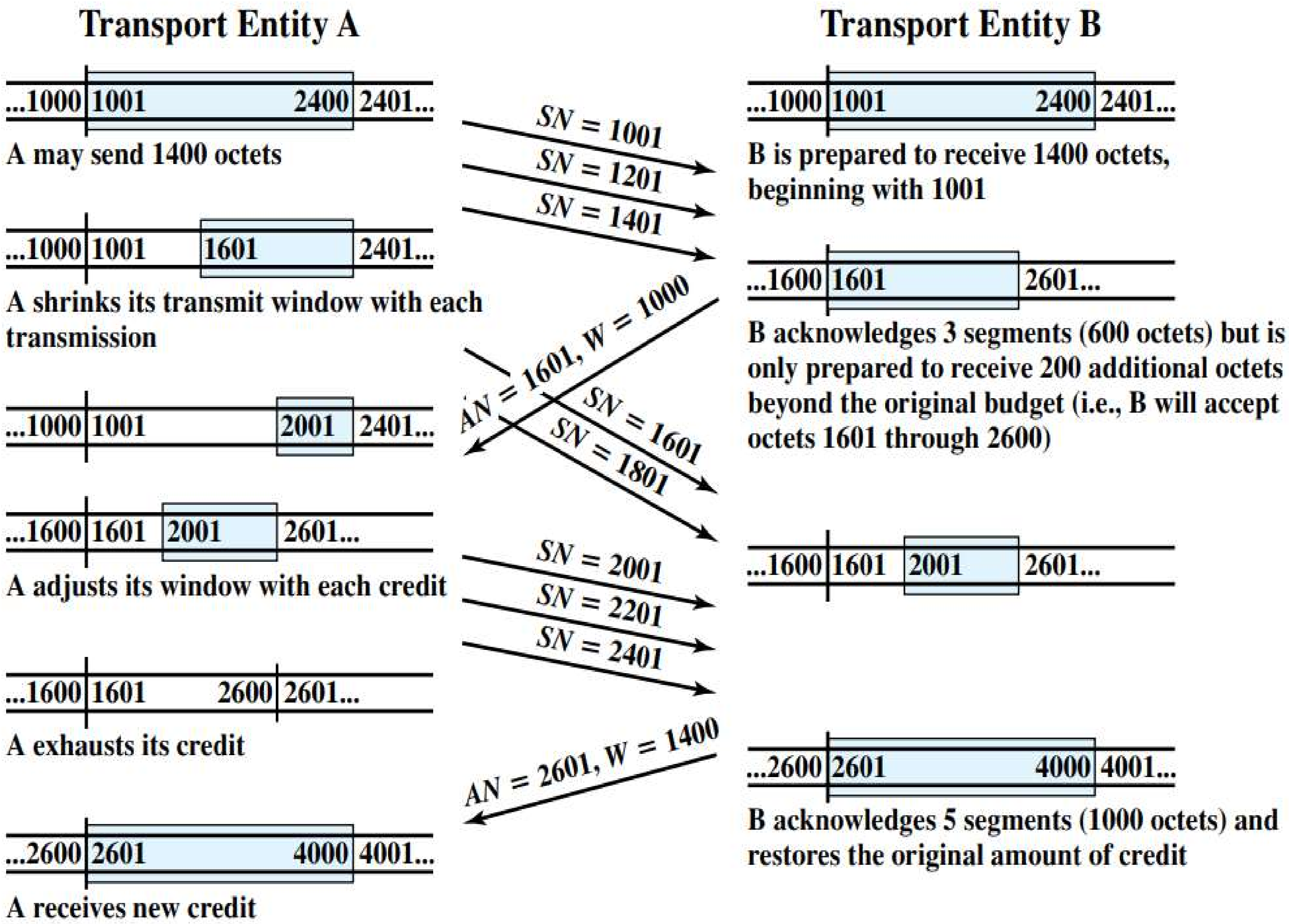
- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

- **Error control** (process-to-process): **entire message** arrives at the receiving transport layer **without error, loss, duplication and in the same order they were sent**
  - Error detection is done using checksum and correction by retransmission
  - Implemented by a **sliding window ARQ (Automatic Repeat Request).**
  - **Every transmission** of data is **acknowledged** by the receiver.
  - Acknowledgements are **cumulative.**
  - If the sender does not receive ACK within a specified amount of time, the sender **retransmits** the data.
  - **Accepts out of order but does Not send negative acknowledgements**,
  - if a segment is not acknowledged before time-out, it is considered to be either **corrupted or lost** and the sender will **retransmit the segment only when it times-out**

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
    - UDP
    - TCP

# *Example of sliding – window protocol*




For more details you can refer to: William Stallings, “DATA AND COMPUTER COMMUNICATIONS, eighth edition”, Prentice Hall; 8th edition (January 1, 2007),

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

# *Example of sliding – window protocol (explanation)*

Figure above illustrates sliding window mechanism. For simplicity, we show data flow in one direction only and assume that 200 octets of data are sent in each segment.

Initially, through the connection establishment process, the sending and receiving sequence numbers are synchronized and A is granted an initial credit allocation of W= 1400 octets, beginning with octet number SN=1001. The first segment transmitted by A contains data octets numbered 1001 through 1200. After sending 600 octets in three segments, A has shrunk its window to a size of 800 octets (numbers 1601 through 2400).

After B receives these three segments, 600 octets out of its original 1400 octets of credit are accounted for, and 800 octets of credit are outstanding. Now suppose that, at this point, B is capable of absorbing W=1000 octets of incoming data on this connection. Accordingly, B acknowledges receipt of all octets through 1600 (AN=1601) and issues a credit of W=1000 octets. This means that A can send octets 1601 through 2600 (5 segments).

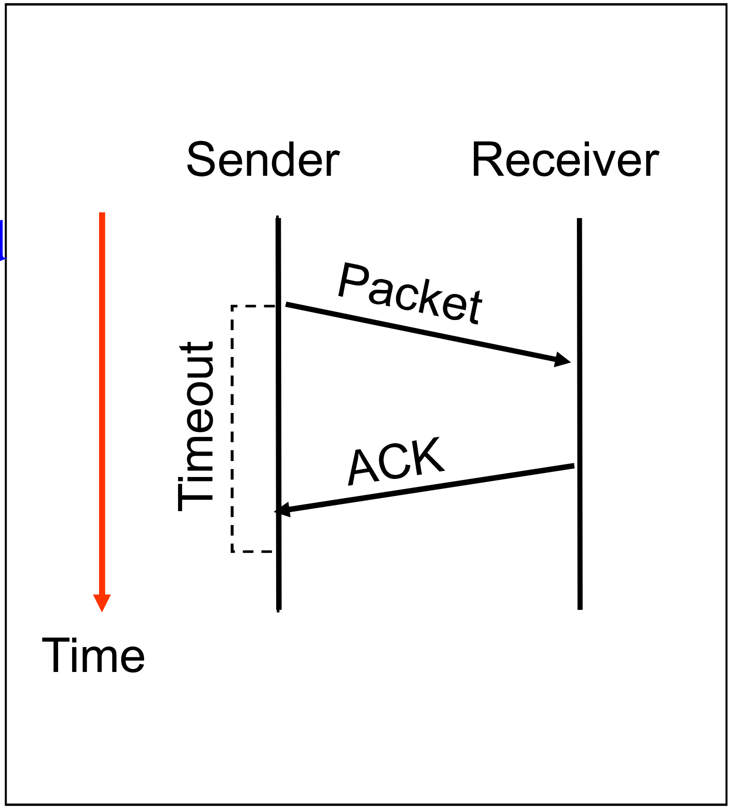
However, by the time that B's message has arrived at A, A has already sent two segments, containing octets 1601 through 2000 (which was permissible under the initial allocation). Thus, A's remaining credit upon receipt of B's credit allocation is only 600 octets (3 segments). As the exchange proceeds, A advances the trailing edge of its window each time that it transmits and advances the leading edge only when it is granted credit.

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

# TCP Retransmission: Automatic Repeat reQuest (ARQ)

- Automatic Repeat Request
  - – Receiver sends acknowledgment (ACK) when it receives packet
  - – Sender waits for ACK and timeouts if it does not arrive within some time period

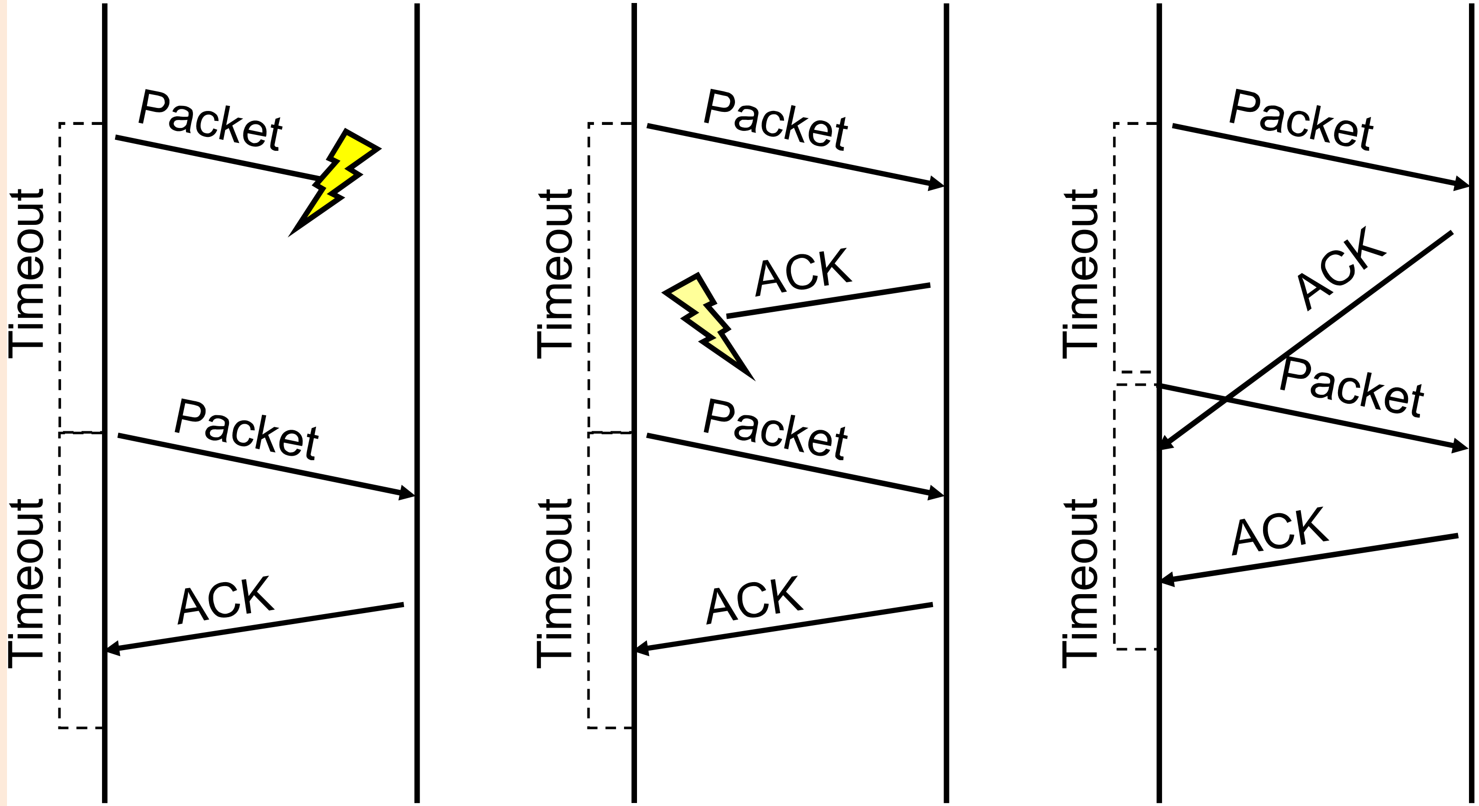



- Simplest ARQ protocol
  - – Stop and wait
  - – Send a packet, stop and wait until ACK arrives

# Reasons for Retransmission

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP




Packet lost

ACK lost
DUPLICATE PACKET

Early timeout
DUPLICATE PACKETS

# How Long Should Sender Wait?

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

- Sender sets a timeout to wait for an ACK
  - Too short: wasted retransmissions
  - Too long: excessive delays when packet lost
- TCP sets timeout as a function of the Round-Trip Time (RTT)
  - Expect ACK to arrive after an RTT
  - … plus a fudge factor to account for queuing
- But, how does the sender know the RTT?
  - Can estimate the RTT by watching the ACKs
  - Smooth estimate: keep a running average of the RTT
    - EstimatedRTT = a * EstimatedRTT + (1 –a ) * SampleRTT
  - Compute timeout: TimeOut = 2 * EstimatedRTT

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

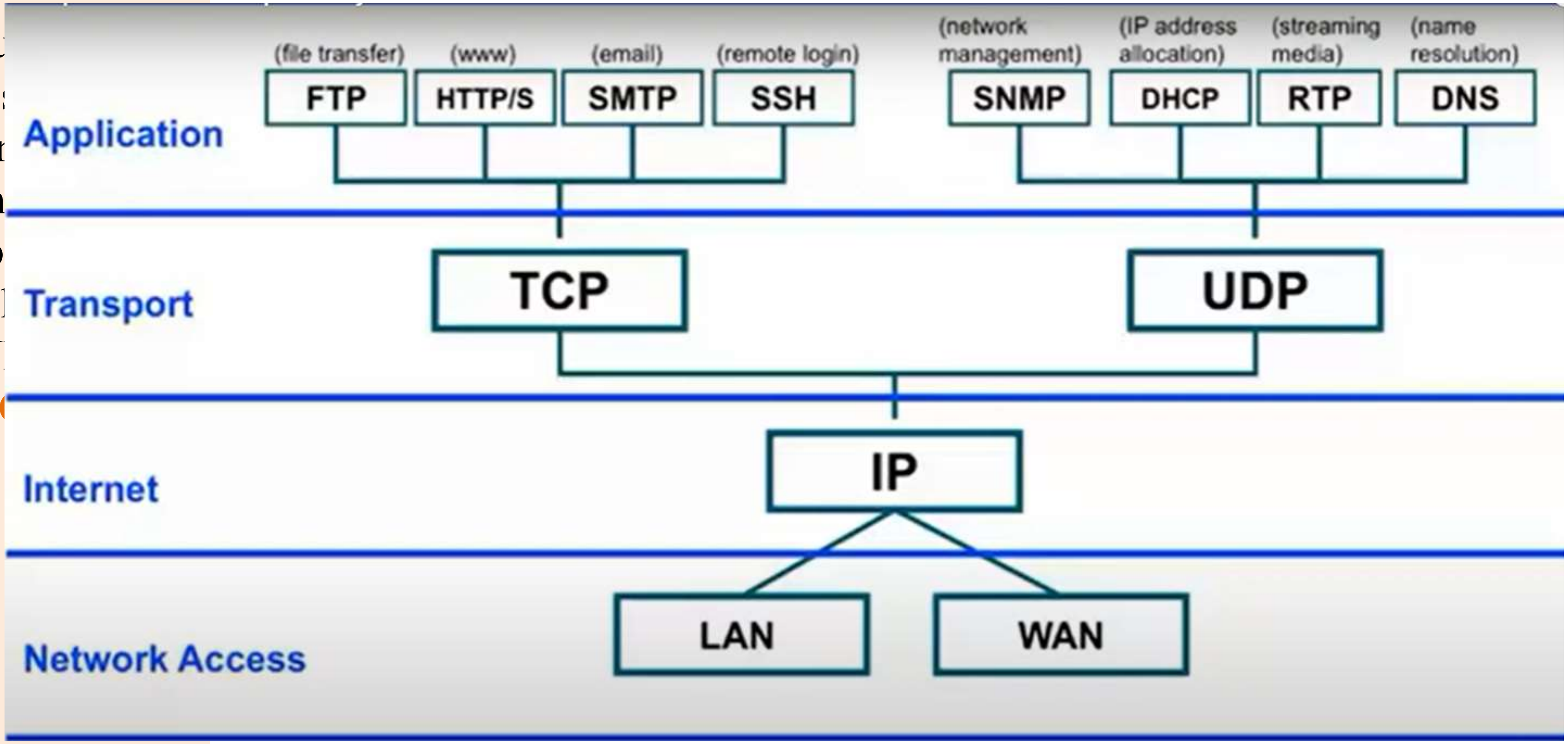
## 2.2.3. TCP Applications

- Following applications require reliable data transfer through TCP:
  - WWW using HTTP
  - Electronic mail using SMTP
  - Telnet
  - File transfer using FTP

# 2.2.3. TCP Applications

**Outlines**

- Introdu
- Process
commun
across n
- Transp
Protocol
    - U
    - T

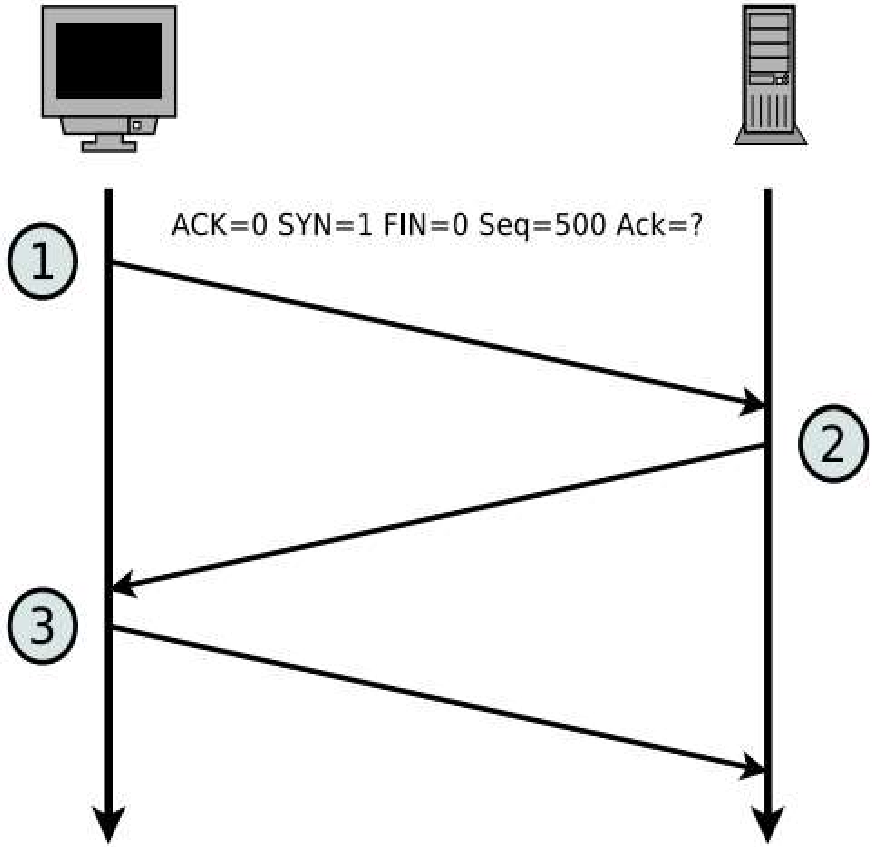


# Exercise

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

1. The diagram shows the establishment of a TCP connection.
Complete the information in the table for the TCP messages 2 and 3 according to TCP messages 1.



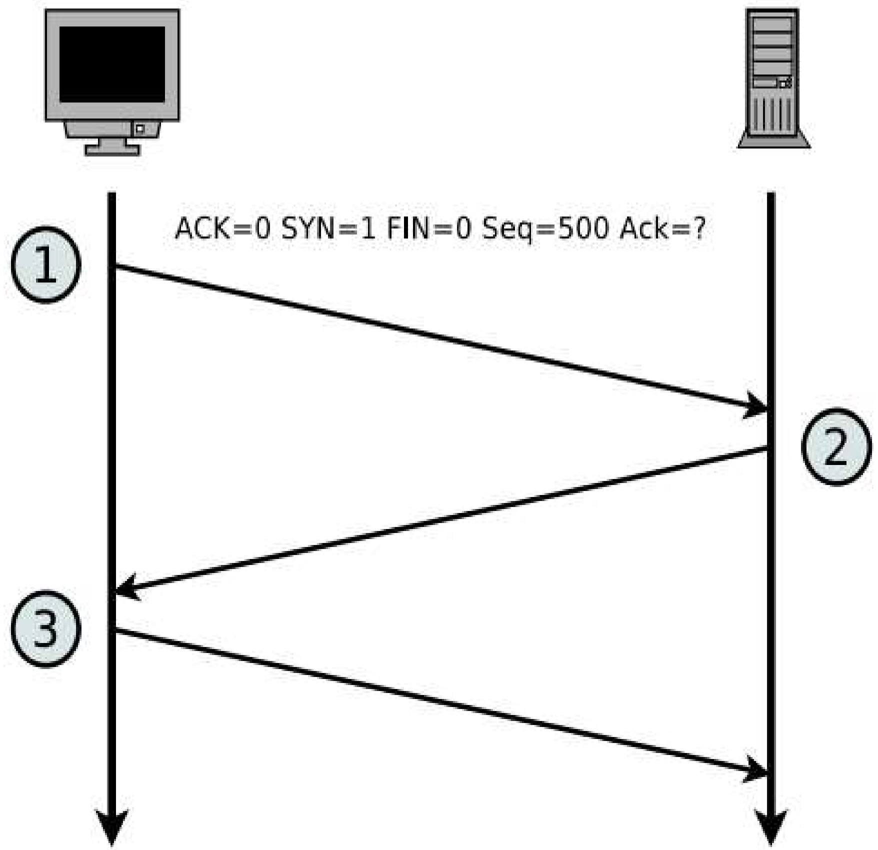

| Message | ACK | SYN | FIN | Seq number | Ack number |
|---|---|---|---|---|---|
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |

# Solution

### Outlines

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

The diagram shows the establishment of a TCP connection.
Complete the information in the table for the TCP messages 2 and 3 according to TCP messages 1.



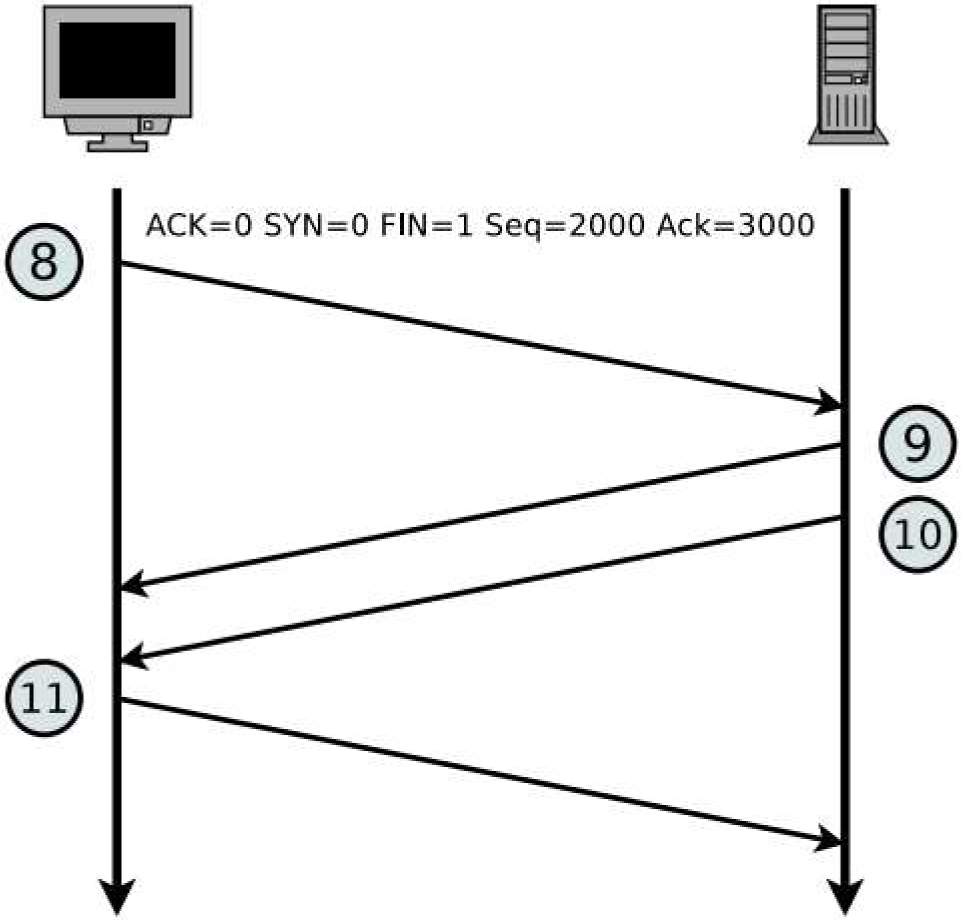

| Message | ACK | SYN | FIN | Seq number | Ack number |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 500 | 0 |
| 2 | 1 | 1 | 0 | 1000 | 501 |
| 3 | 1 | 0 | 0 | 501 | 1001 |

# Exercise

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
    - UDP
    - TCP

The diagram shows the termination of a TCP connection. Complete the table.



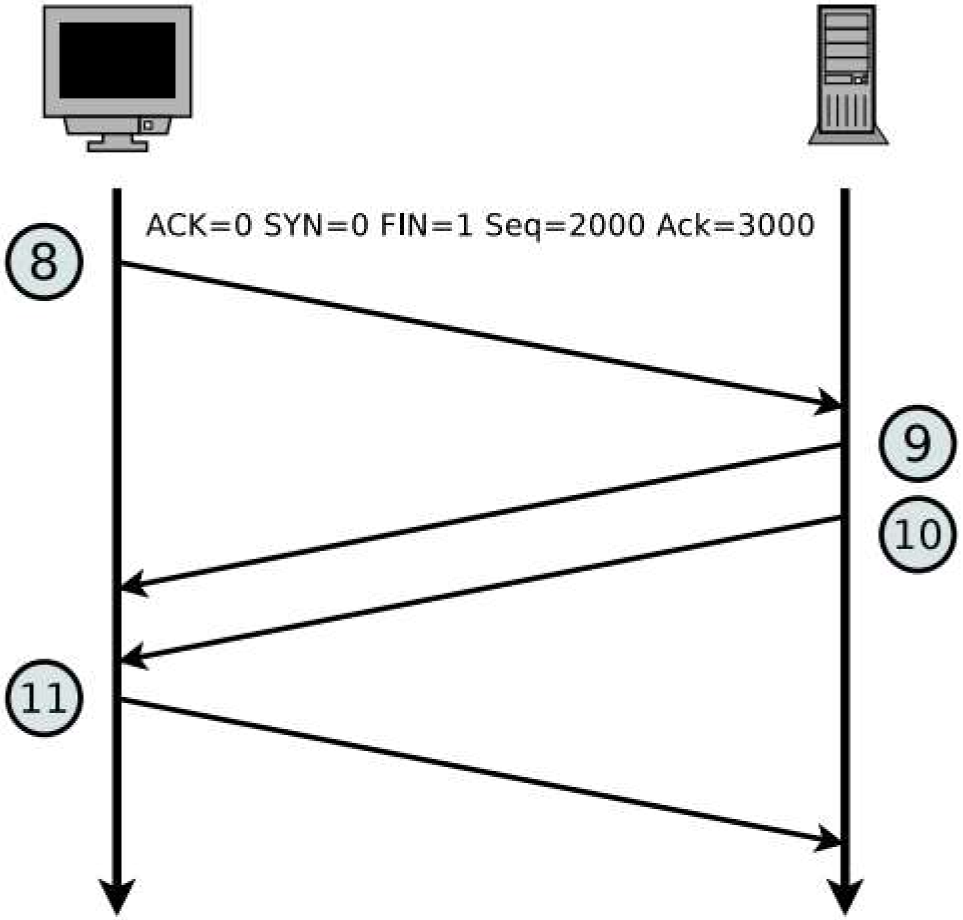

| Message | ACK | SYN | FIN | Seq number | Ack number |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

**Outlines**

- Introduction
- Processes communicating across network
- Transport Layer Protocols
  - UDP
  - TCP

# Solution

The diagram shows the termination of a TCP connection. Complete the table.




| Message | ACK | SYN | FIN | Seq number | Ack number |
|---|---|---|---|---|---|
| 8 | 0 | 0 | 1 | 2000 | 3000 |
| 9 | 1 | 0 | 0 | 3000 | 2001 |
| 10 | 0 | 0 | 1 | 3000 | 2001 |
| 11 | 1 | 0 | 0 | 2001 | 3001 |