DRONACHARYA
College of Engineering

Computer Science & Engineering

Data Communication and Computer Networks

(MTCSE-101-A)

DATA COMMUNICATION AND COMPUTER NETWORKS: syllabus

- UNIT 3
- Transport layer: Transport layer services , UDP , TCP Protocols , TCP services , TCP features, Connection Management , Congestion control SCTP Protocol (**Stream Control Transmission Protocol**), SCTP Services, SCTP features , an SCTP Association
- Application layer: SMTP , POP, IMAP and MIME ,DHCP (Dynamic Host Configuration Protocol) , DHCP Operations , **Configuration FTP** , SSH

Transport layer protocol

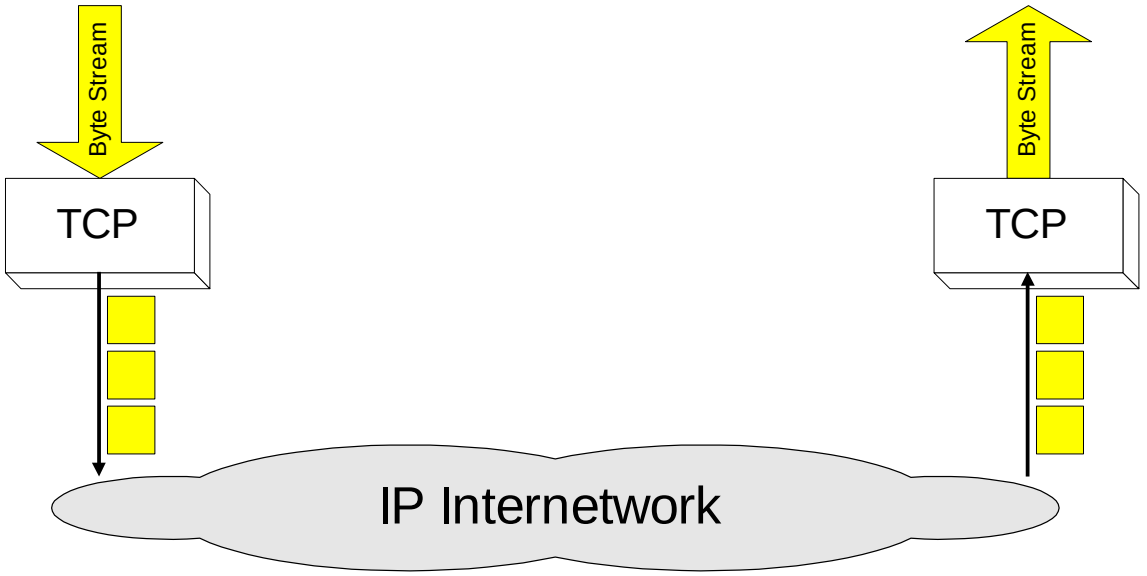
- The **transmission control protocol** (**TCP**) is used for applications in which **reliable** connections between hosts are necessary. TCP checks for transmission errors, lost packets, packets out of order, etc, and tries to automatically correct these without "bothering" the application program. It also does **flow control**, slowing transmission if it is too fast for the receiver.
- The **user datagram protocol** (**UDP**), is an unreliable transport protocol with no sessions or flow control and optional error checking. UDP just sends packets as soon as requested and forgets about them. It is faster than TCP, and is suitable for **isochronous** applications like **voice over IP** (**VOIP**) or streaming video where error correction is pointless.

TCP-TRANSMISISON CONTROL PROTOCOL

Overview

TCP = Transmission Control Protocol

- Connection-oriented protocol
- Provides a reliable unicast end-to-end byte stream over an unreliable internetwork.



TCP
Transmission Control Protocol

- TCP is an alternative transport layer protocol over IP.
- TCP provides:
  - Connection-oriented
  - Reliable
  - Full-duplex
  - Byte-Stream

Connection-Oriented

- *Connection oriented* means that a virtual connection is established before any user data is transferred.
- If the connection cannot be established - the user program is notified.
- If the connection is ever interrupted - the user program(s) is notified.

Reliable

- *Reliable* means that every transmission of data is acknowledged by the receiver.
- If the sender does not receive acknowledgement within a specified amount of time, the sender retransmits the data.
- Byte stream is broken up into chunks which are called **seg-ments**
  - Receiver sends acknowledgements (ACKs) for segments
  - **Detecting errors:**

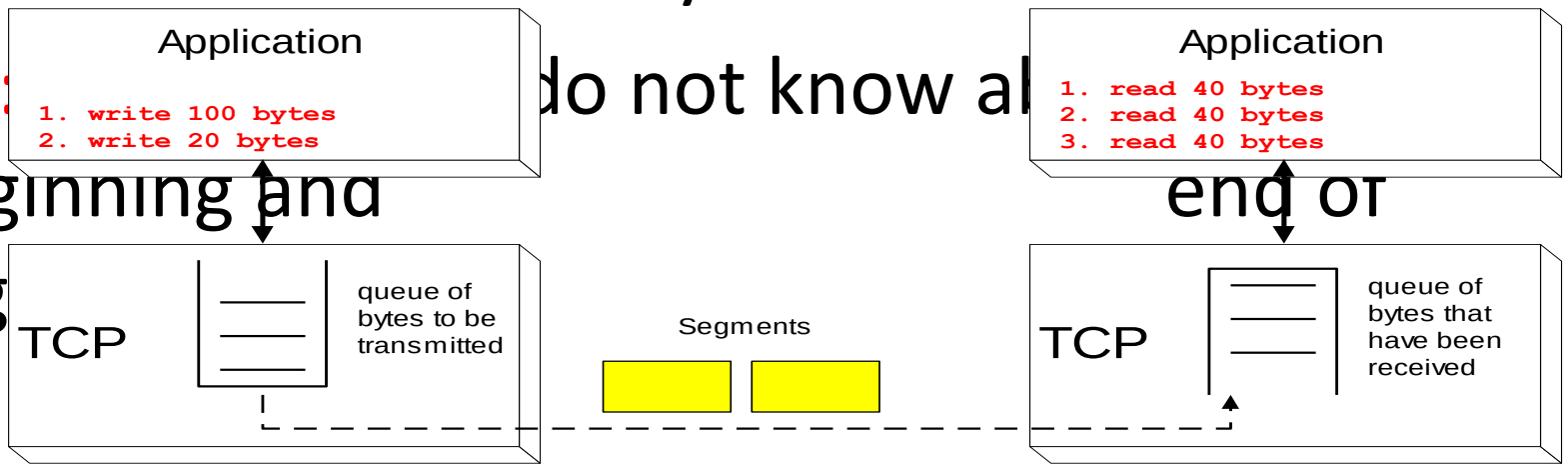
Byte Stream

- *Stream* means that the connection is treated as a stream of bytes.

- The user application does not need to package data in individual datagrams (as with UDP).

Byte Stream Service

- To the lower layers, TCP handles data in blocks, the segments.
- To the higher layers TCP handles data as a sequence of bytes and does not identify boundaries between bytes
- So… do not know al… beginning and … end of seg…



Buffering

- TCP is responsible for buffering data and determining when it is time to send a datagram.

- It is possible for an application to tell TCP to send the data it has buffered without waiting for a buffer to fill up.

Full Duplex

- TCP provides transfer in both directions.

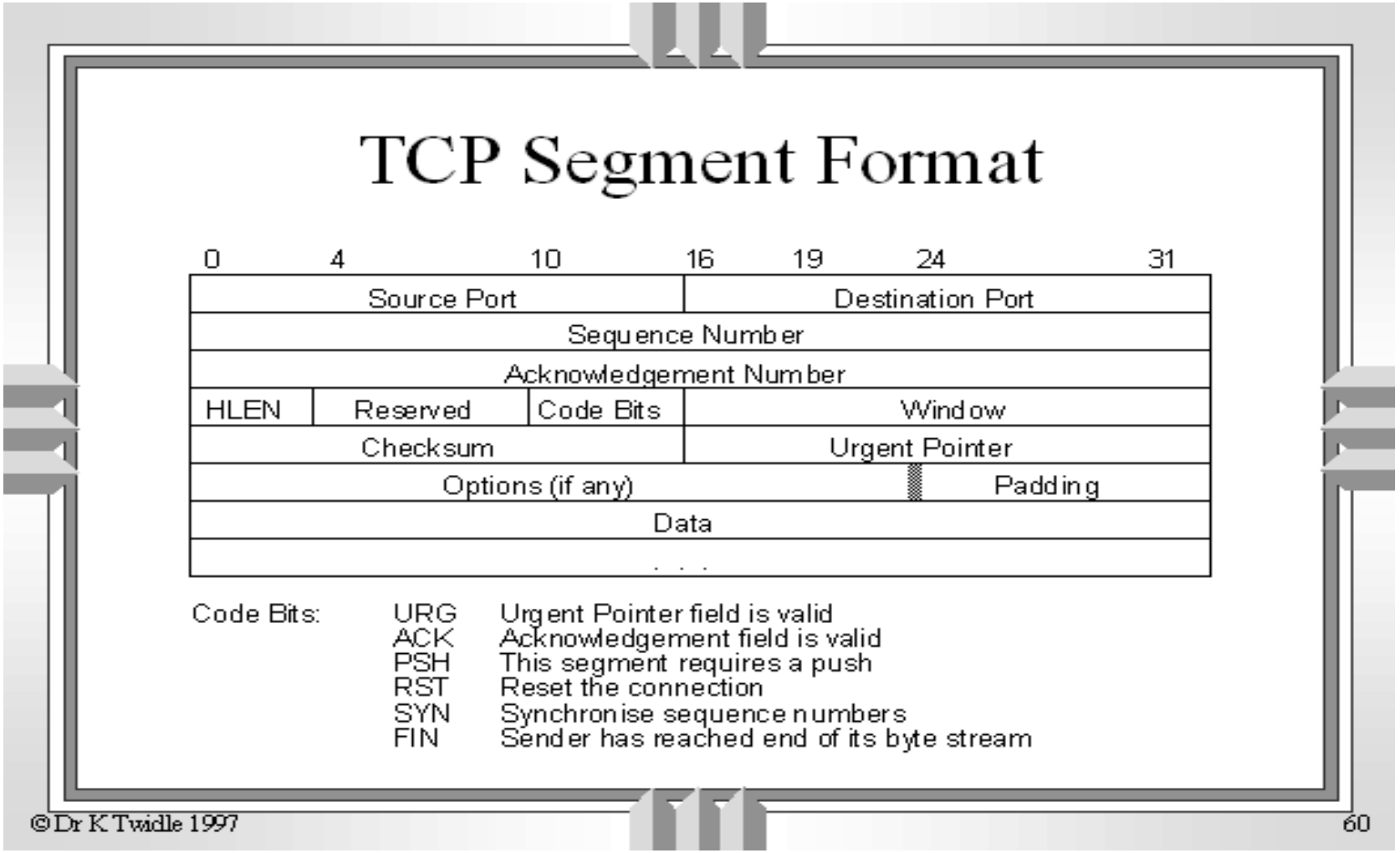
- To the application program these appear as 2 unrelated data streams, although TCP can piggyback control and data communication by providing control information (such as an ACK) along with user data.

TCP Segments

- The chunk of data that TCP asks IP to deliver is called a *TCP segment*.

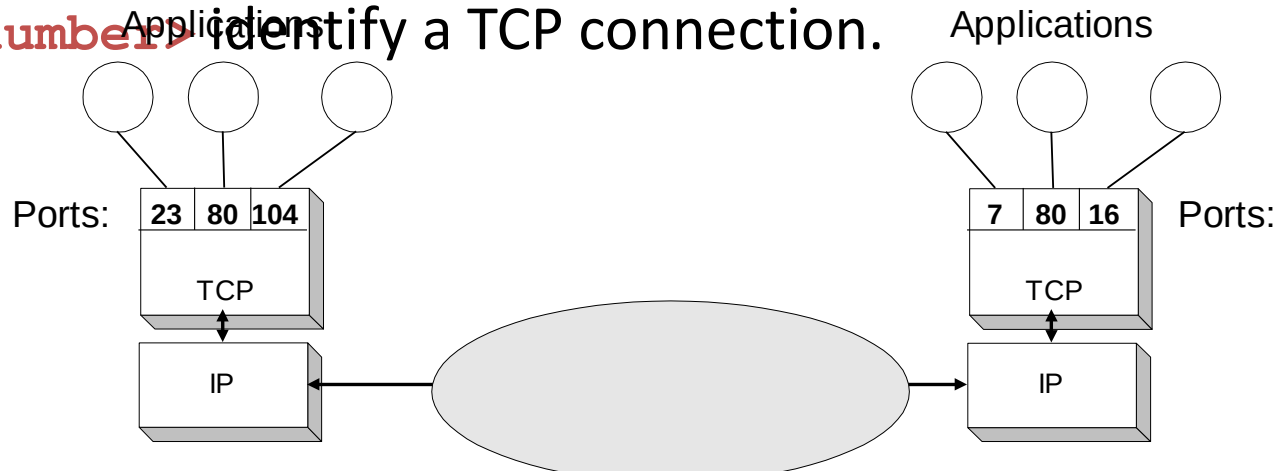
- Each segment contains:
  - data bytes from the byte stream
  - control information that identifies the data bytes

TCP Segment Format



TCP header fields

- **Port Number:**
  - A port number identifies the endpoint of a connection.
  - A pair `<IP address, port number>` identifies one endpoint of a connection.
  - Two pairs `<client IP address, server port number>` and `<server IP address, server port number>` identify a TCP connection.



TCP header fields

- **Sequence Number (SeqNo):**
  - Sequence number is 32 bits long.
  - So the range of SeqNo is

    $0 <= SeqNo <= 2^{32} - 1 \approx 4.3$ Gbyte

  - Each sequence number identifies a byte in the byte stream
  - **Initial Sequence Number (ISN) of a connection is set during connection establishment**

TCP header fields

- **Acknowledgement Number (AckNo):**
  - Acknowledgements are piggybacked, I.e

    a segment from A -> B can contain an acknowledgement for a data sent in the B -> A direction

  - A hosts uses the AckNo field to send acknowledgements. (If a host sends an AckNo in a segment it sets the "**ACK flag**")
  - The AckNo contains the next SeqNo that a hosts wants to receive

    Example: The acknowledgement for a segment

TCP header fields

- **Acknowledge Number (cont'd)**
  - TCP uses the sliding window flow protocol to regulate the flow of traffic from sender to receiver
  - TCP uses the following variation of sliding window:
    - no NACKs (**N**egative **ACK**nowledgement)
    - only cumulative ACKs
- Example:

  Assume: Sender sends two segments with "1..1500" and "1501..3000", but receiver only gets the second segment.

  In this case, the receiver cannot acknowledge the second packet. It can only send AckNo=1

TCP header fields

- **Header Length (4bits):**
  - Length of header in 32-bit words
  - Note that TCP header has variable length (with minimum 20 bytes)

TCP header fields

- **Flag bits:**
  - **URG: Urgent pointer is valid**
    - If the bit is set, the following bytes contain an urgent message in the range:

      SeqNo <= urgent message <= SeqNo+urgent pointer
  - **ACK: Acknowledgement Number is valid**
  - **PSH: PUSH Flag**
    - Notification from sender to the receiver that the receiver should pass all data that it has to the application.
    - Normally set by sender when the sender's buffer is empty

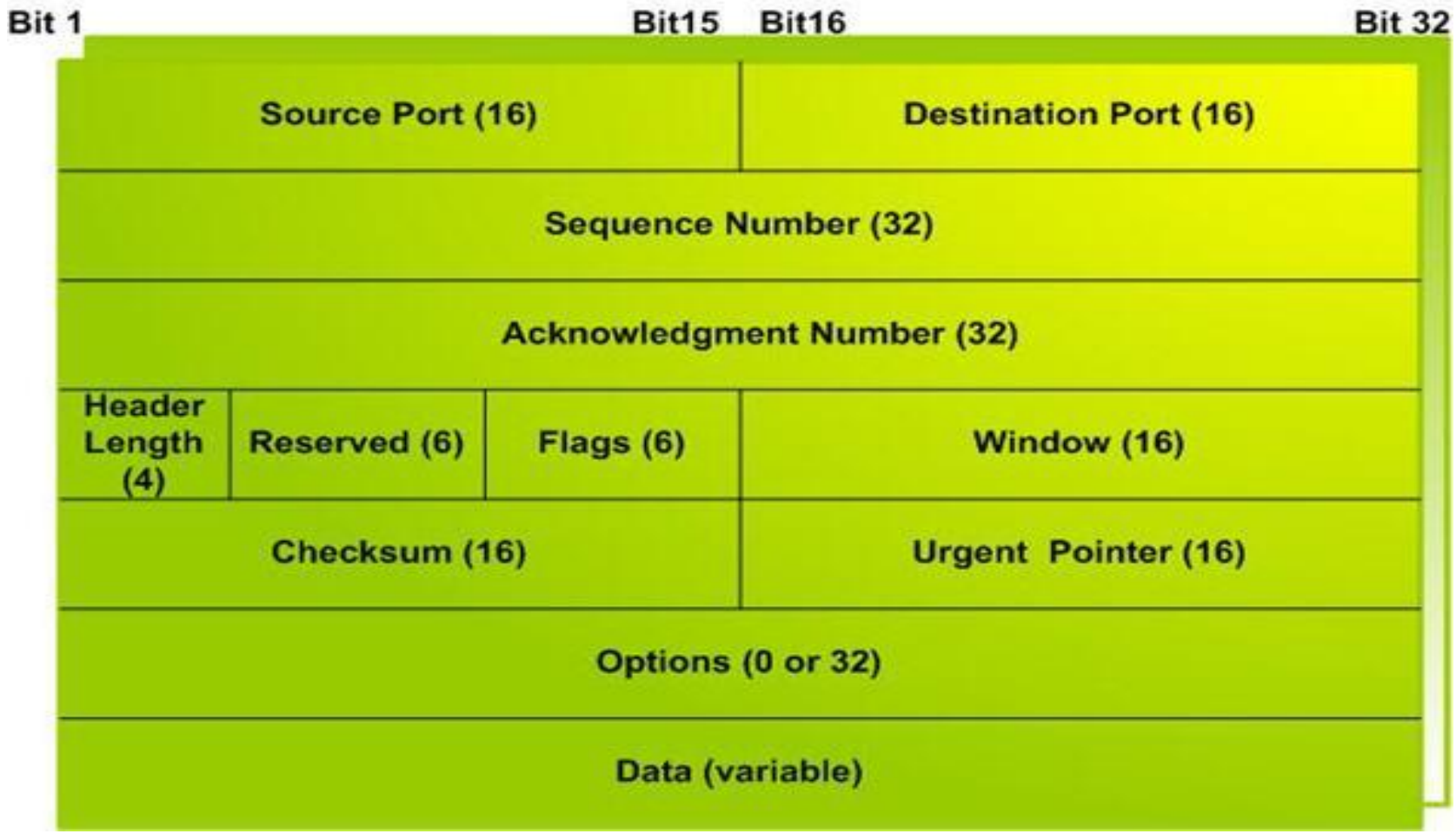
TCP header fields

- **Flag bits:**
  - **RST: Reset the connection**
    - The flag causes the receiver to reset the connection
    - Receiver of a RST terminates the connection and indicates higher layer application about the reset
  - **SYN: Synchronize sequence numbers**
    - Sent in the first packet when initiating a connection
  - **FIN: Sender is finished with sending**
    - Used for closing a connection
    - Both sides of a connection must send a **FIN**

TCP header fields

- **Window Size:**
  - Each side of the connection advertises the window size
  - Window size is the maximum number of bytes that a receiver can accept.
  - Maximum window size is 2^{16}-1= 65535 bytes
- **TCP Checksum:**
  - TCP checksum covers over both TCP header **and** TCP data (also covers some parts of the IP header)
- **Urgent Pointer:**

The TCP Segment Format



TCP Lingo

- When a client requests a connection it sends a "SYN" segment (a special TCP segment) to the server port.
- SYN stands for synchronize. The SYN message includes the client's ISN.
- **ISN is Initial Sequence Number**.
- Every TCP segment includes a *Sequence Number* that refers to the first byte of *data* included in the segment.
- Every TCP segment includes an

And more...

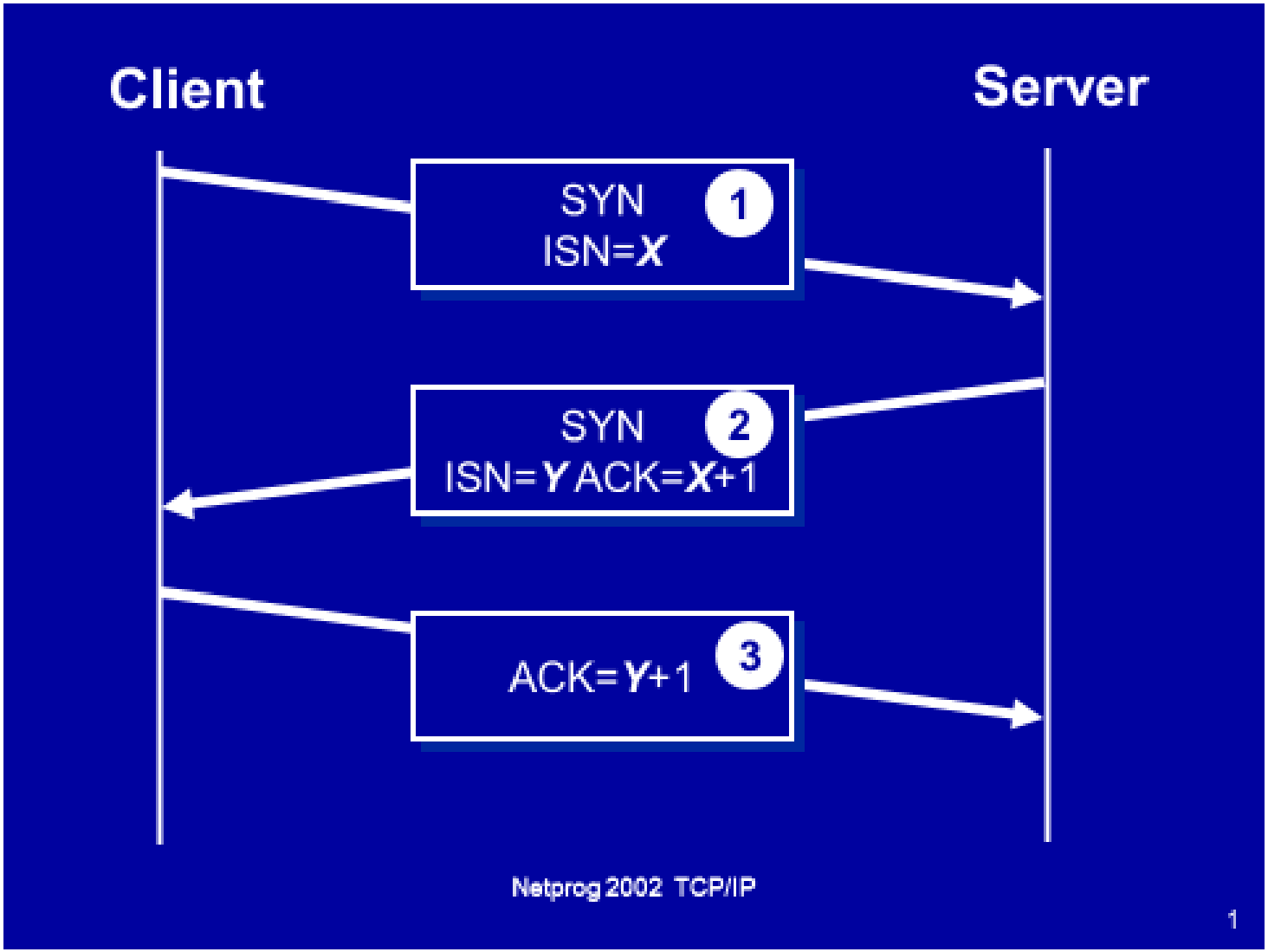
- There are a bunch of control flags:
  - URG: urgent data included.
  - ACK: this segment is (among other things) an acknowledgement.
  - RST: error – connection must be reset.
  - SYN: synchronize Sequence Numbers (setup)
  - FIN: polite connection termination.
- MSS: Maximum segment size (A TCP option)
- Window: Every ACK includes a Window field that tells the sender how many bytes it can

TCP Connection Creation

- Programming details later - for now we are concerned with the actual communication.
- A *server* accepts a connection.
  - Must be looking for new connections!
- **A *client* requests a connection.**
  - **Must *know* where the server is!**
- **A client starts by sending a SYN segment with the following information:**
  - **Client's ISN (generated pseudo-randomly)**
  - Maximum Receive Window for client.

Server Response

- When a waiting server sees a new connection request, the server sends back a SYN segment with:
  - **Server's ISN (generated pseudo-randomly)**
  - **Request Number is Client ISN+1**
  - Maximum Receive Window for server.
  - Optionally (but usually) MSS
  - No payload! (Only TCP headers)
- **When the Server's SYN is received, the client sends back an ACK with:**
  - **Acknowledgment Number is Server's ISN+1**


Client
Server
SYN
ISN=X
1
SYN
ISN=Y ACK=X+1
2
ACK=Y+1
3
Netprog 2002 TCP/IP
1

TCP 3-way handshake

1. Client: “I want to talk, and I’m starting with byte number ***X***”.

2. Server: “OK, I’m here and I’ll talk. My first byte will be called number ***Y***, and I know your first byte will be number ***X+1***”.

3. Client: “Got it - you start at byte number ***Y+1***”.

?. Bill: “Monica, I’m afraid I’ll syn and byte your ack”

ACKs

- A receiver doesn't have to ACK every segment (it can ACK many segments with a single ACK segment).
- Each ACK can also contain outgoing data (piggybacking).
- If a sender doesn't get an ACK after some time limit, it resends the data.

TCP Segment Order

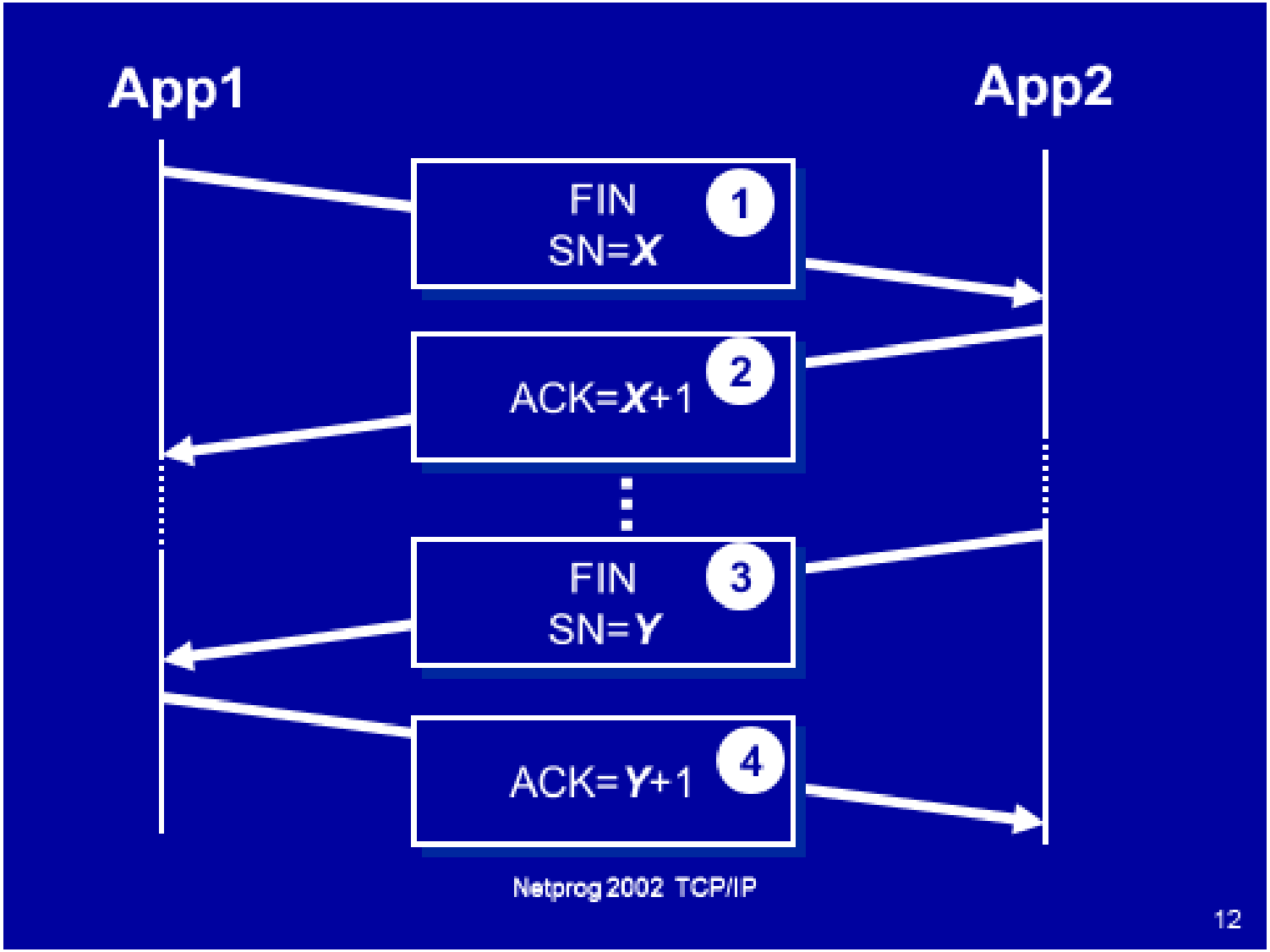
- Most TCP implementations will accept out-of-order segments (if there is room in the buffer).
- Once the missing segments arrive, a single ACK can be sent for the whole thing.
- Remember: IP delivers TCP segments, and IP is not reliable - IP datagrams can be lost or arrive out of order.

Termination

- The TCP layer can send a RST segment that terminates a connection if something is wrong.
- Usually the application tells TCP to terminate the connection politely with a FIN segment.

FIN

- Either end of the connection can initiate termination.
- A FIN is sent, which means the application is done sending data.
- The FIN is ACK'd.
- The other end must now send a FIN.
- That FIN must be ACK'd.


App1
App2
FIN
SN=X
1
ACK=X+1
2
FIN
SN=Y
3
ACK=Y+1
4
Netprog 2002 TCP/IP
12

TCP Termination

1. *App1: "I have no more data for you".*

2. *App2: "OK, I understand you are done sending."*

   dramatic pause…

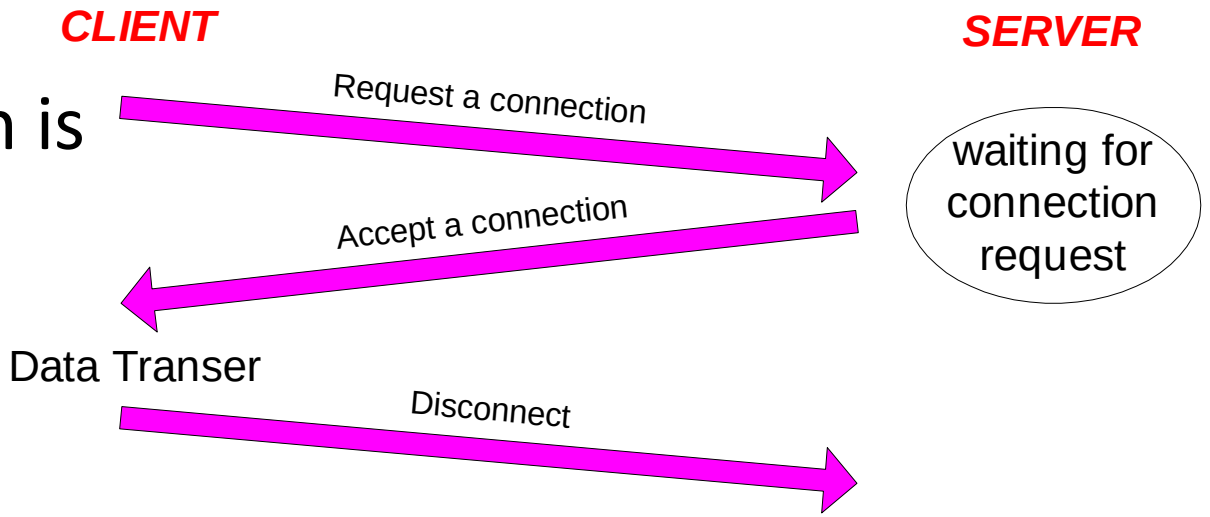
3. *App2: "OK - Now I'm also done sending data".*

4. *App1: "Roger, Over and Out, Goodbye, Hastalavista Baby, Adios, It's been real ..."*

   camera fades to black ...

Connection-Oriented

- Before any data transfer, TCP establishes a **connection**:
  - One TCP entity is waiting for a connection ("**server**")
  - The other TCP entity ("**client**") contacts the server
- The actual procedure for setting up connections is more complex.
- Each connection is full duplex

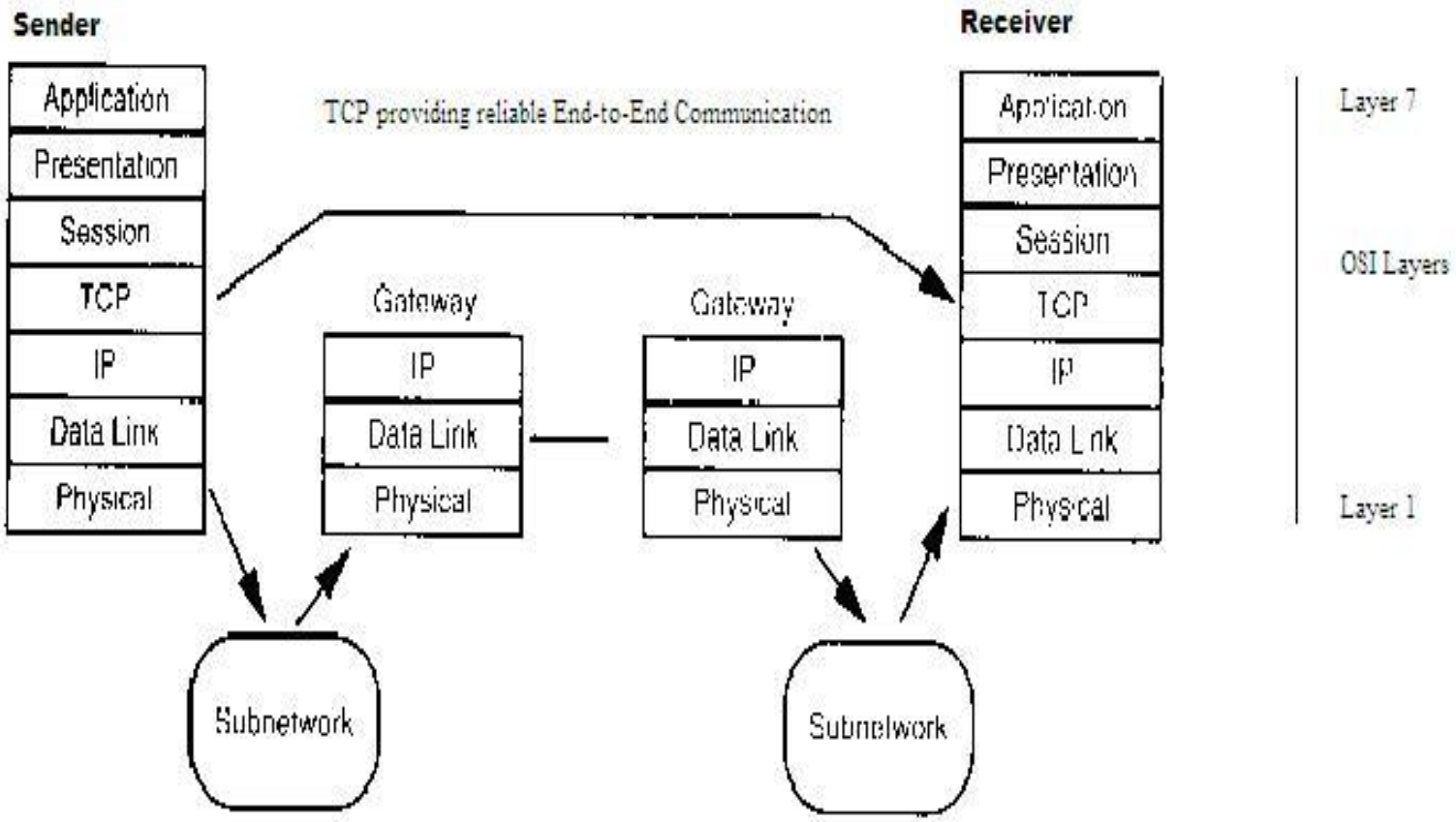


TCP

- **TCP is not a piece of software. It is a communications protocol.**

- TCP manages the flow of datagrams from the higher layers, as well as incoming datagrams from the IP layer. TCP resides in the transport layer, positioned above IP but below the upper layers and their applications.

TCP


Sender
Receiver
Application
Presentation
Session
TCP
IP
Data Link
Physical
TCP providing reliable End-to-End Communication
Gateway
IP
Data Link
Physical
Gateway
IP
Data Link
Physical
Application
Presentation
Session
TCP
IP
Data Link
Physical
Subnetwork
Subnetwork
Layer 7
OSI Layers
Layer 1

Connection Management in TCP

- **Opening a TCP Connection**
- **Closing a TCP Connection**
- **Special Scenarios**
- **State Diagram**

TCP Connection Establishment

- TCP uses a **three-way handshake** to open a connection:

(1) ACTIVE OPEN: Client sends a segment with

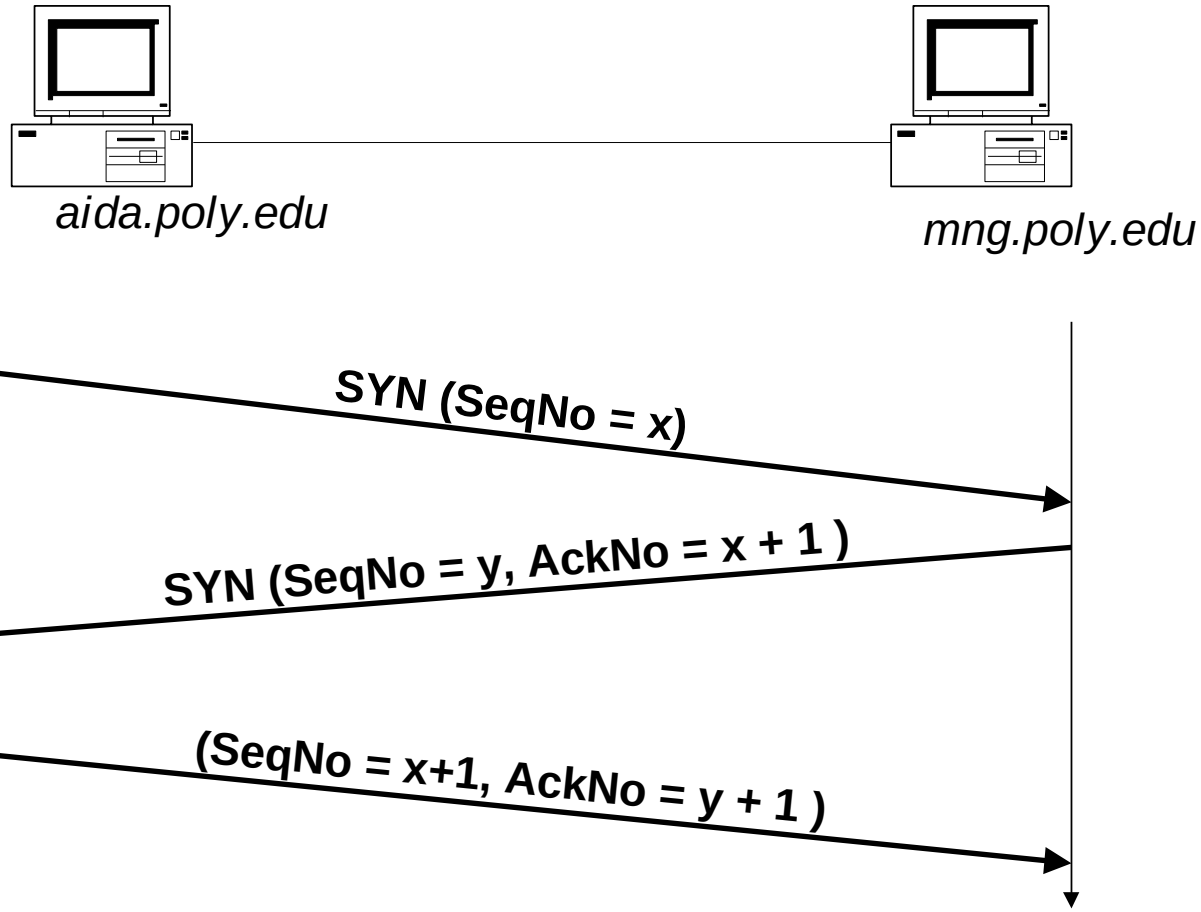
  - SYN bit set *
  - port number of client
  - initial sequence number (ISN) of client

(2) PASSIVE OPEN: Server responds with a segment with

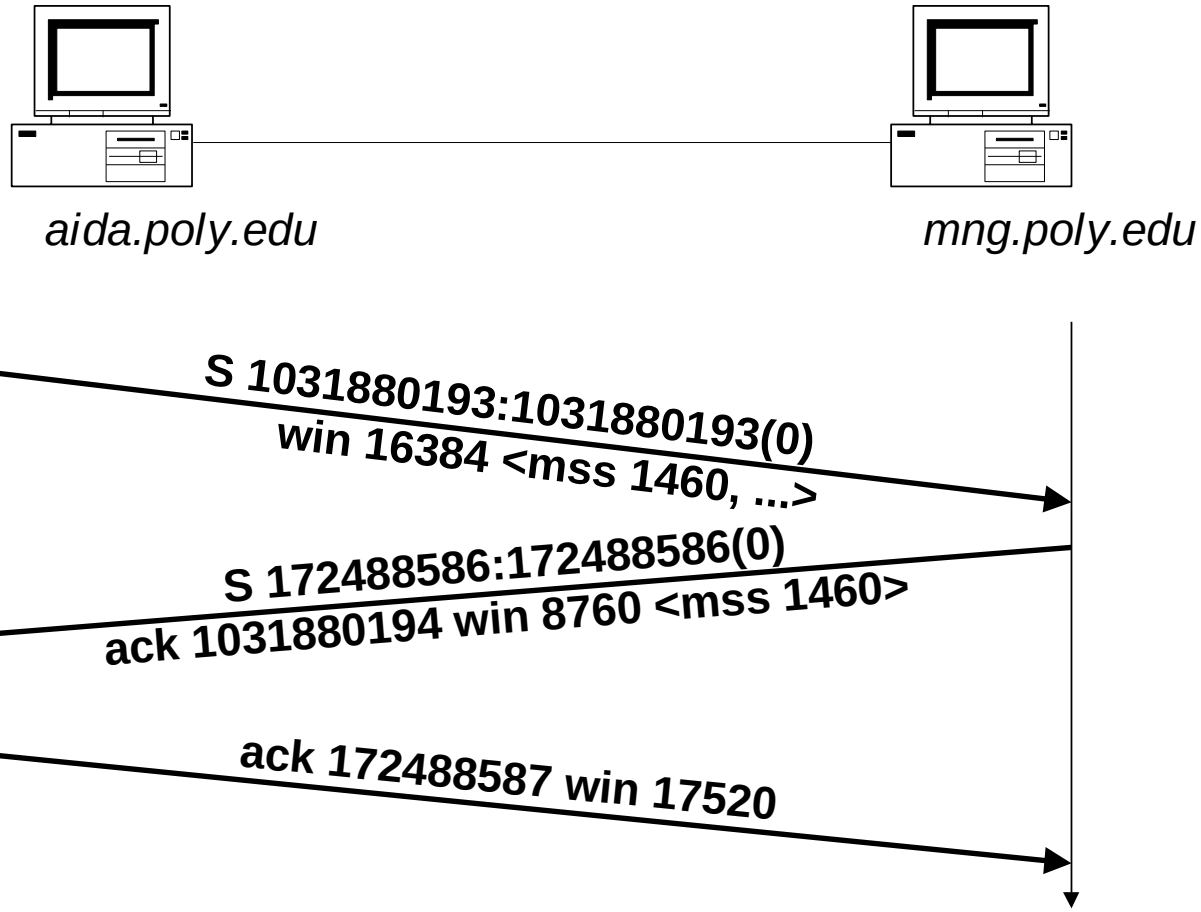
  - SYN bit set *
  - initial sequence number of server
  - ACK for ISN of client

(3) Client acknowledges by sending a segment with:

Three-Way Handshake



Three-Way Handshake



TCP Connection Termination

- Each end of the data flow must be shut down independently **("half-close")**
- If one end is done it sends a FIN segment. This means that no more data will be sent

- Four steps involved:

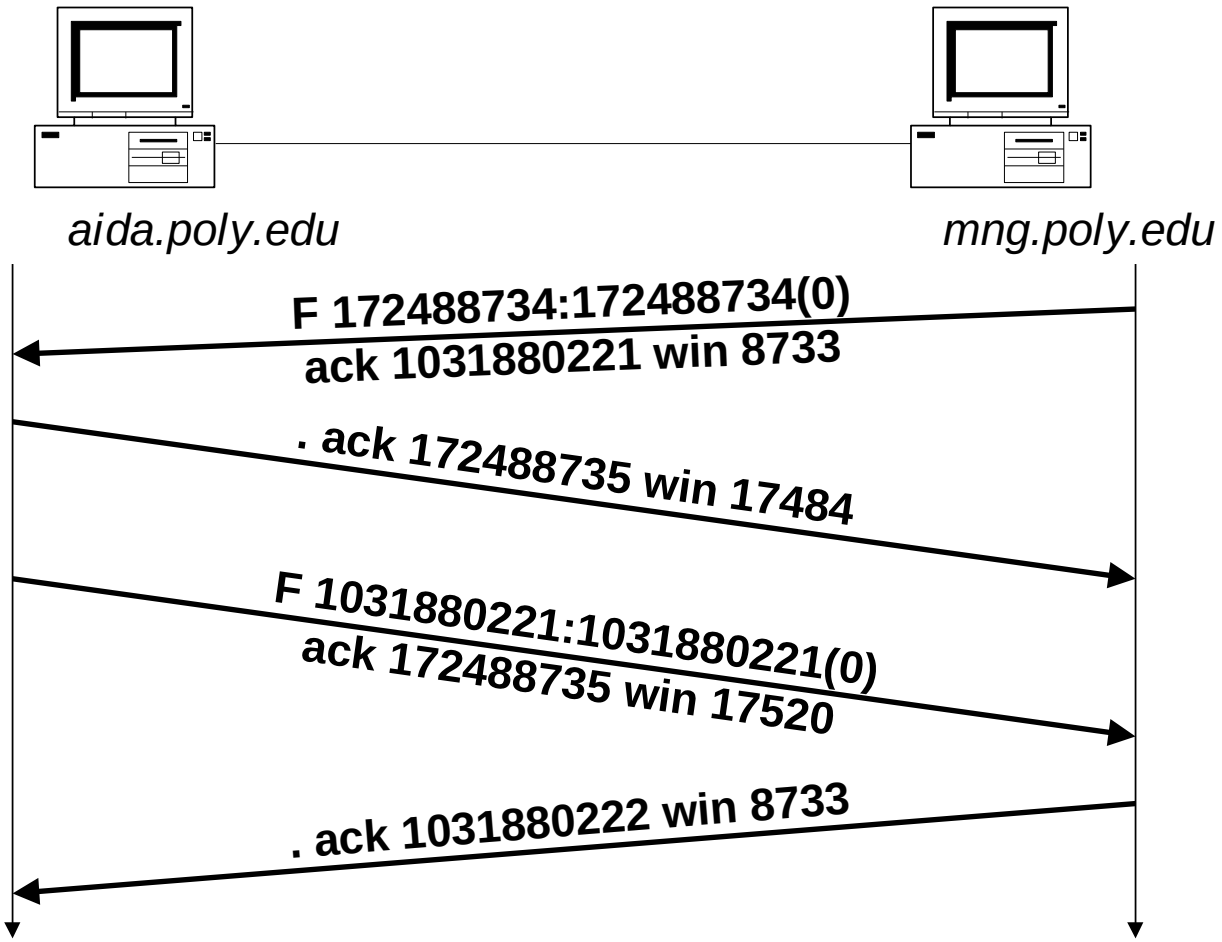
  (1) X sends a FIN to Y **(active close)**

  (2) Y ACKs the FIN,

  (at this time: Y can still send data to X)

  (3) and Y sends a FIN to X **(passive close)**

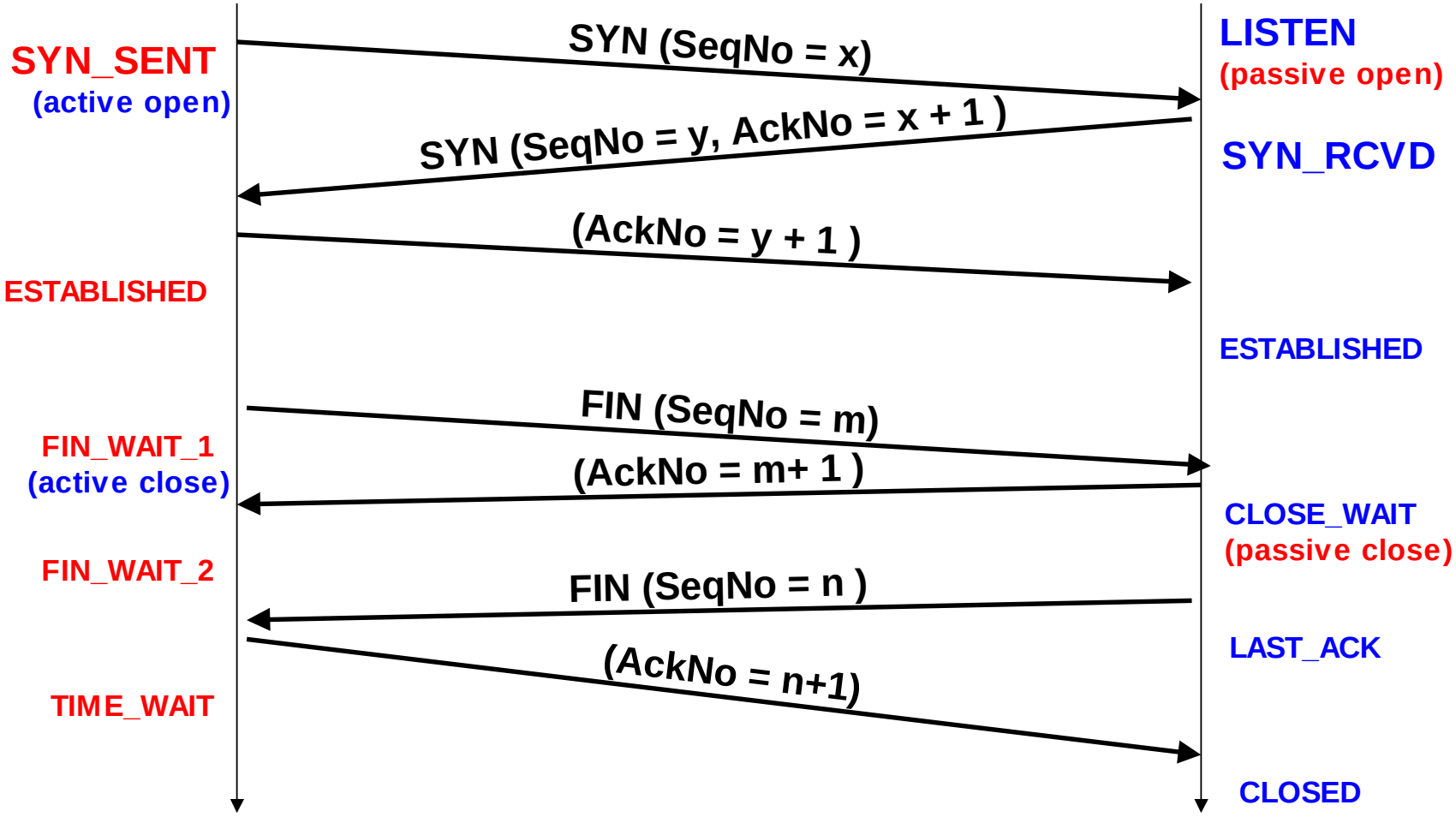
TCP Connection Termination



TCP States

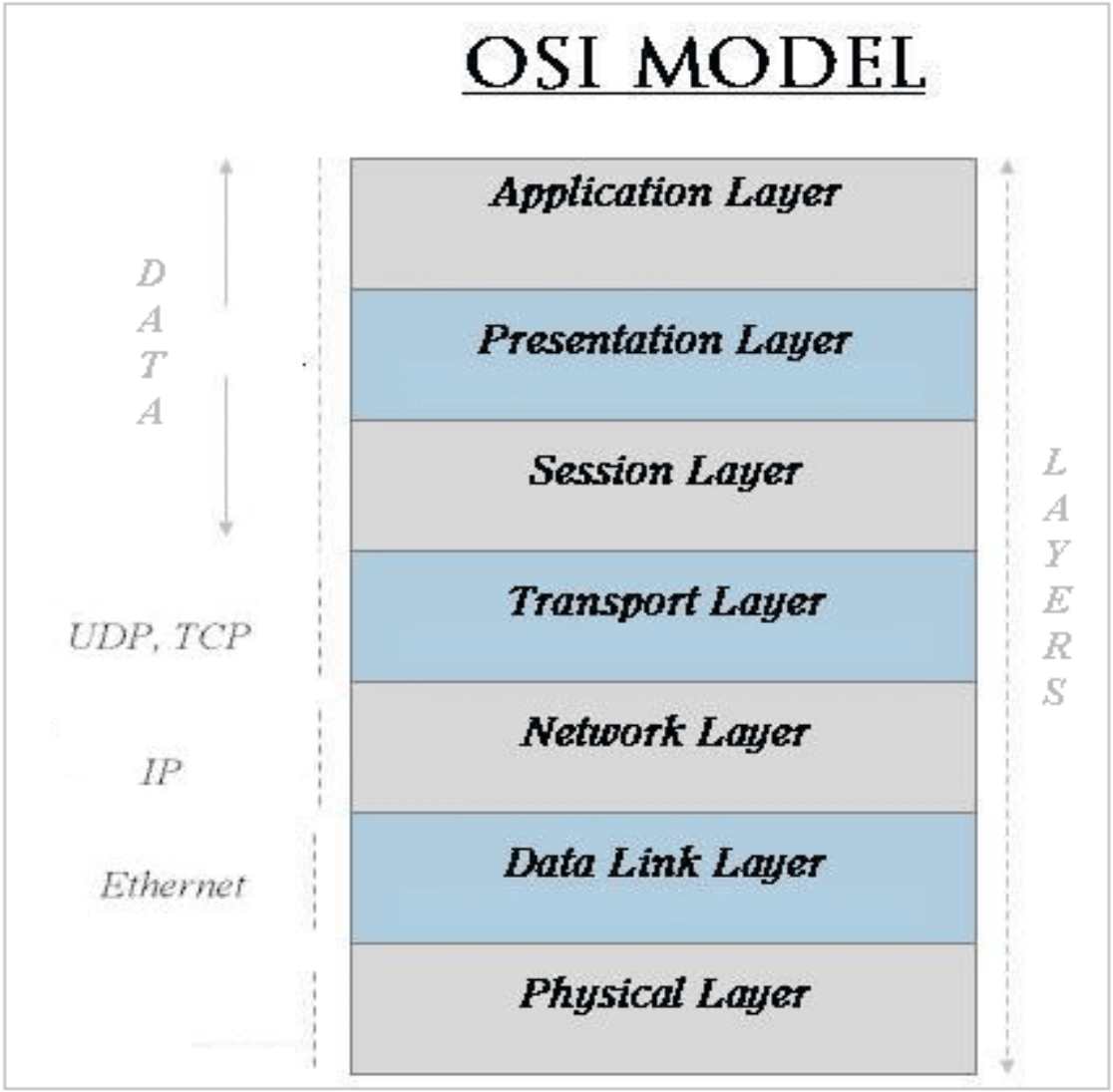
State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for Ack
SYN SENT	The client has started to open a connection
ESTABLISHED	Normal data transfer state
FIN WAIT 1	Client has said it is finished
FIN WAIT 2	Server has agreed to release
TIMED WAIT	Wait for pending packets (“2MSL wait state”)
CLOSING	Both Sides have tried to close simultanesously
CLOSE WAIT	Server has initiated a release
LAST ACK	Wait for pending packets

TCP States in "Normal" Connection Lifetime



UDP: User Datagram Protocol

- The User Datagram Protocol (UDP) is a transport layer protocol defined for use with the IP network layer protocol. It is defined by RFC 768 written by John Postel. 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.**
- **UDP provides a minimal, unreliable, best-effort, message-passing transport to applications and upper-layer protocols**


OSI MODEL
Application Layer
Presentation Layer
Session Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer
DATA
UDP, TCP
IP
Ethernet
LAYERS

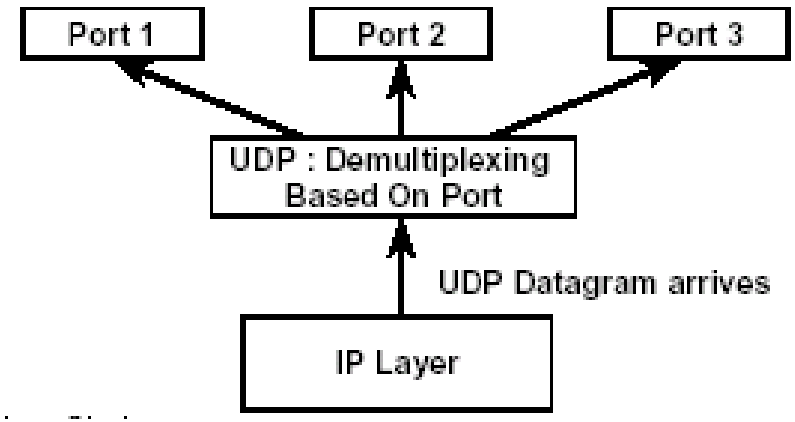
UDP Characteristics

- **End-to-End**: an application sends/receives data to/from another application.
- **Connectionless**: Application does not need to preestablish communication before sending data; application does not need to terminate communication when finished.
- **Message-oriented**: application sends/receives individual messages (UDP datagram), not packets.
- **Best-effort**: same best-effort delivery

Protocol Port Number

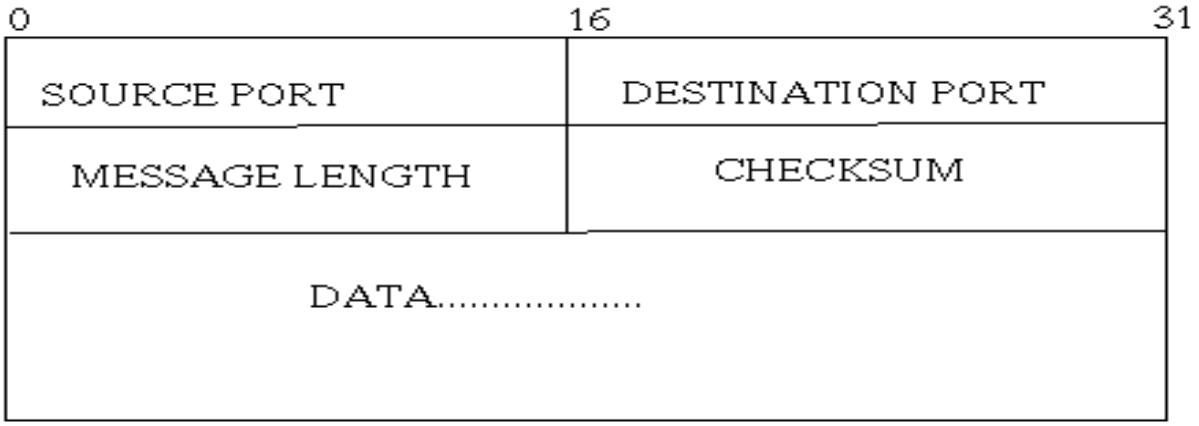
- UDP uses ***Port Number*** to identify an application as an endpoint.
- UDP messages are delivered to the port specified in the message by the sending application
- In general, a port can be used for any datagram, as long as the sender and the receiver agrees
- In practice, a collection of well-known ports are used for special purposes such as telnet, ftp, and email. E.g. port 7 for Echo application.
- Local operating system provides an interface for processes to specify and access a port.

UDP Multiplexing & Demultiplexing

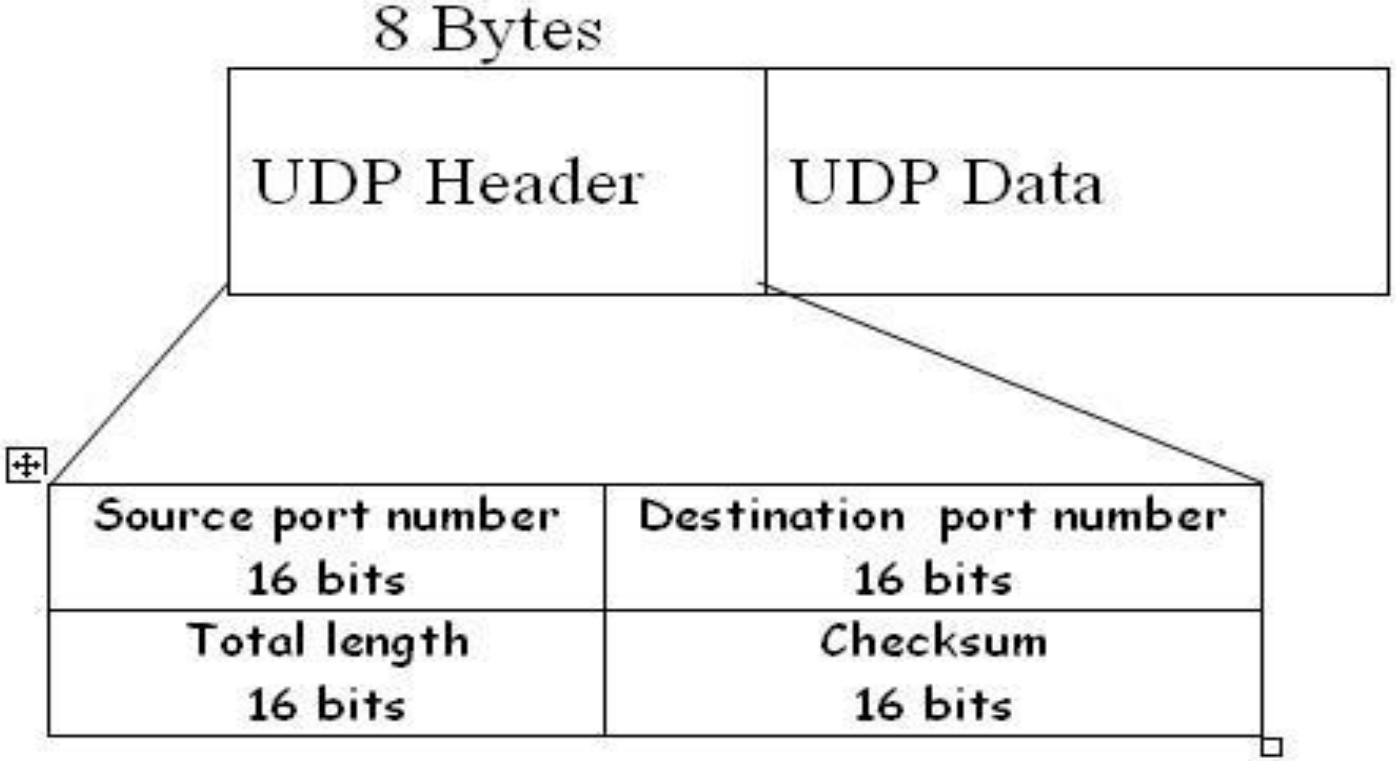


- Sender: multiplexing of UDP datagrams.
  - UDP datagrams are received from multiple application programs.
  - A single sequence of UDP datagrams is passed to IP layer.
- Receiver: demultiplexing of UDP datagrams.
  - Single sequence of UDP datagrams received from IP layer.
  - UDP datagram received is passed to appropriate application.

UDP Datagram Format

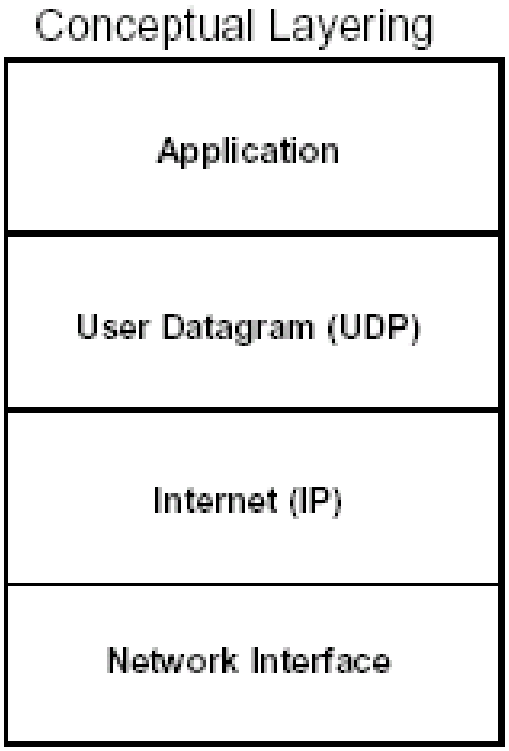


- Source Port - 16 bit port number
- Destination Port - 16 bit port number
- Length (of UDP header + data) - 16 bit count of octets
- UDP checksum - 16 bit field. if 0, then there is no checksum, else it is a checksum over a pseudo header + UDP data area

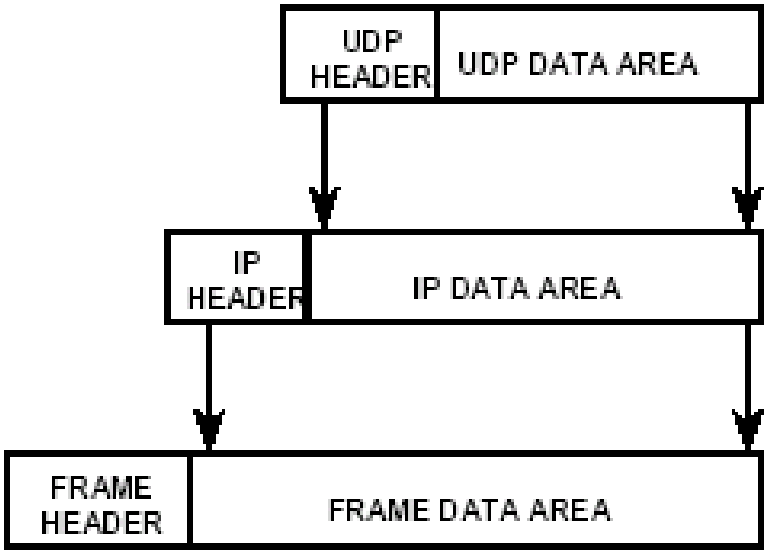

8 Bytes
UDP Header
UDP Data
Source port number
16 bits
Destination port number
16 bits
Total length
16 bits
Checksum
16 bits

Encapsulation and Layering

- Protocol layering



- UDP encapsulation



- UDP message is encapsulated into an IP datagram.
- IP datagram in turn is encapsulated into a physical frame for actually delivery.

UDP—User Datagram Protocol

- An unreliable, connectionless transport layer protocol
- Two additional functions beyond IP:
  - Demultiplexing: deliver to different upper layer entities such as DNS, RTP, SNMP based on the destination port # in the header. i.e., UDP can support multiple applications in the same end systems.
  - (Optionally) check the integrity of entire UDP. (recall IP only checks the integrity of IP header.)
    - If source does not want to compute checksum, fill checksum with all 0s.
    - If compute checksum and the checksum happens to be 0s, then fill all 1s.
    - UDP checksum computation is similar to IP checksum, with two more:
      - Add extra 0s to entire datagram if not multiple of 16 bits.
      - Add pseudoheader to the beginning of datagram. UDP pseudoheader

TCP	UDP
Reliable	Unreliable
Connection-oriented	Connectionless
Segment retransmission and flow control through windowing	No windowing or retransmission
Segment sequencing	No sequencing
Acknowledge segments	No acknowledgement

TCP	UDP
Keeps track of lost packets. Makes sure that lost packets are re-sent	Doesn't keep track of lost packets
Adds sequence numbers to packets and reorders any packets that arrive in the wrong order	Doesn't care about packet arrival order
Slower, because of all added additional functionality	Faster, because it lacks any extra features
Requires more computer resources, because the OS needs to keep track of ongoing communication sessions and manage them on a much deeper level	Requires less computer resources
Examples of programs and services that use TCP: - HTTP - HTTPS - FTP - Many computer games	Examples of programs and services that use UDP: - DNS - IP telephony - DHCP - Many computer games

TCP vs. UDP

No.	TCP	UDP
1	This Connection oriented protocol	This is connection-less protocol
2	The TCP connection is byte stream	The UDP connection is a message stream
3	It does not support multicasting and broadcasting	It supports broadcasting
4	It provides error control and flow control	The error control and flow control is not provided
5	TCP supports full duplex transmission	UDP does not support full duplex transmission
6	It is reliable service of data transmission	This is an unreliable service of data transmission
7	The TCP packet is called as segment	The UDP packet is called as user datagram.

Differences Between TCP and UDP	
TCP	**UDP**
Sequenced	Unsequenced
Reliable -sequence numbers, acknowledgments, and 3-way handshake	Unreliable -best effort only
Connection Oriented	Connectionless
Virtual Circuits	Low Overhead
Checksum for Error Checking	Checksum for Error Checking
Uses buffer management to avoid overflow, uses sliding window to maximize bandwidth efficiency	No flow control
Assigns datagram size dynamically for efficiency	Every datagram segment is the same size