Transport Layer

JAIPUR (H1)

Elder Tripathi Ji House

Jevan	Mamta	Sameer
Arjun	Shweta	Divya

MUMBAI (H2)

Younger Tripathi Ji House

Umesh	Anju	Kriti
Arun	Karishma	Tushar

LNMIIT
The LNM Institute of Information Technology

- Every month, each member writes a letter to each of the members of the other house
- Each house sends 36 letters to the other house every month
- Each letter is delivered by traditional postal service in a separate envelope
- Every month, Arjun visits all the house members (H1), collects the letter and give it to postal service guy who visits H1 every month
- Similarly, when the letters arrive from house (H2), Arjun has a job to distribute the letters among all the family members
- Karishma has a similar job in house 2

Transport Layer

- The postal service provides logical communication between the two houses. The postal service moves letters from house to house, not person to person.
- Arjun and Karishma provide logical communication among the family members
- Arjun and Karishma pick up letters and deliver them to their family members
- From a family member's perspective, Arjun and Karishma provide the letters-delivering services
- Arjun and Karishma are only a part (the end-system part) of the end-to-end delivery process
- This household example is an excellent analogy for explaining how the transport layer relates to the network layer

application messages = letters in envelopes
processes = family members
hosts (end systems) = houses
transport-layer protocol = arjun and karishma
network-layer protocol = postal service

Transport Layer

- Application processes use the logical communication provided by the transport layer to send messages to each other

- Transport layer protocols are implemented in end systems and not in network routers

- The transport layer converts the application-layer messages it receives from a sending application process into transport-layer packets, known as transport-layer ***segments***

- The transport layer then passes the segment to the network layer at the sending end system, where the segment is encapsulated within a network-layer packet (a ***datagram***) and sent to the destination

- 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 transport-layer segments sent between hosts, and then the transport-layer protocol cannot provide delay or bandwidth guarantees for application messages sent between processes

Transport Layer

- **Transport layer is responsible for *process-to-process* delivery**

- **At the transport layer, we need a transport layer address, called a *port number*, to choose among multiple processes running on the destination host. The destination port number is necessary for delivery; the source port number is required for the reply**

- **The client program defines itself with a port number chosen randomly by the transport layer software running on the client host. Servers port number cannot be chosen randomly**

lANA Ranges

The lANA (Internet Assigned Number Authority) has divided the port numbers into three ranges: well known, registered, and dynamic (or private), as shown in Figure 23.4.

- Well-known ports. The ports ranging from 0 to 1023 are assigned and controlled by lANA. These are the well-known ports.
- Registered ports. The ports ranging from 1024 to 49,151 are not assigned or controlled by lANA. They can only be registered with lANA to prevent duplication.
- Dynamic ports. The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used by any process. These are the ephemeral ports.

Transport Layer

- **Socket address:- Combination of an IP address and Port number**

- **Encapsulation and Decapsulation**

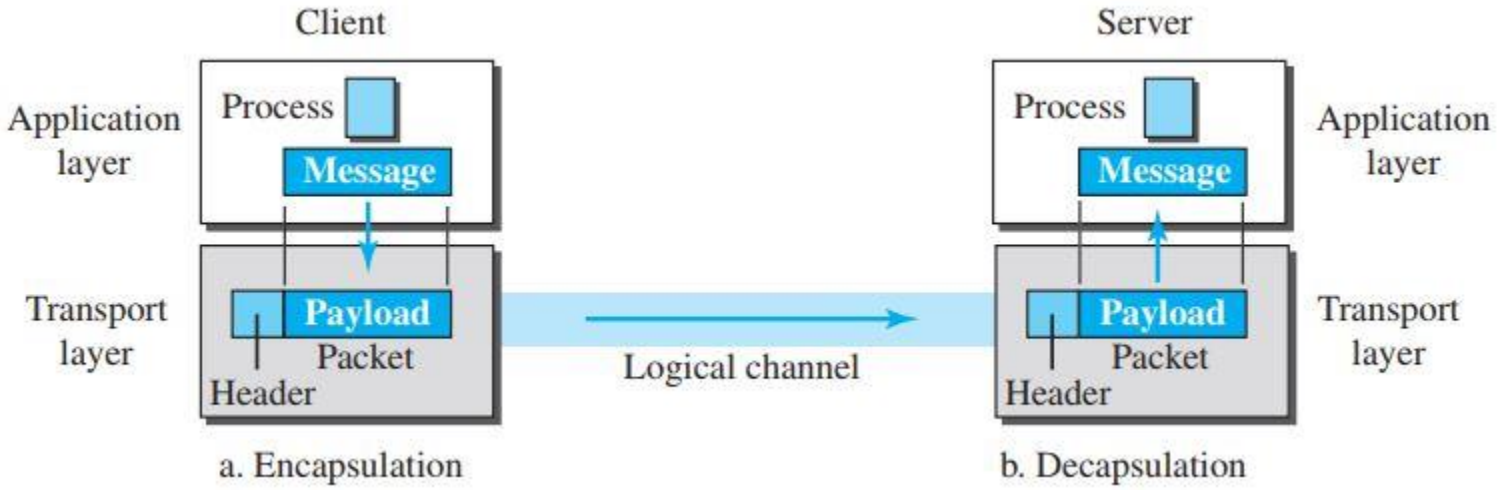

Fig 1. Encapsulation and Decapsulation

- **Multiplexing**

- **Demultiplexing**

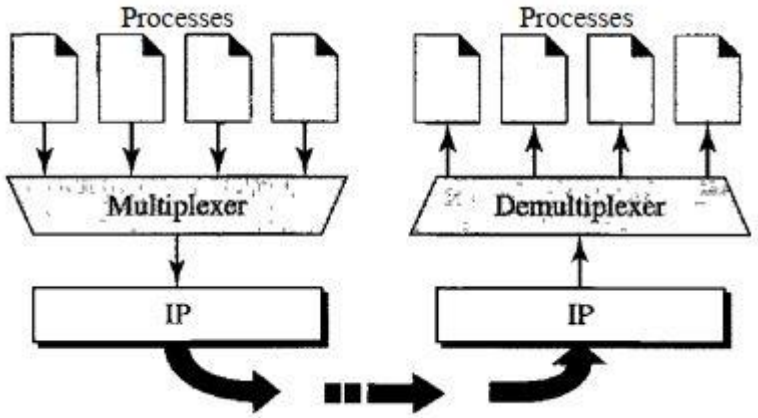

Fig 2. Multiplexing and Demultiplexing

Transport Layer

Port	Protocol	UDP	TCP	SCTP	Description
7	Echo	√	√	√	Echoes back a received datagram
9	Discard	√	√	√	Discards any datagram that is received
11	Users	√	√	√	Active users
13	Daytime	√	√	√	Returns the date and the time
17	Quote	√	√	√	Returns a quote of the day
19	Chargen	√	√	√	Returns a string of characters
20	FTP-data		√	√	File Transfer Protocol
21	FTP-21		√	√	File Transfer Protocol
23	TELNET		√	√	Terminal Network
25	SMTP		√	√	Simple Mail Transfer Protocol
53	DNS	√	√	√	Domain Name Service
67	DHCP	√	√	√	Dynamic Host Configuration Protocol
69	TFTP	√	√	√	Trivial File Transfer Protocol
80	HTTP		√	√	HyperText Transfer Protocol
111	RPC	√	√	√	Remote Procedure Call
123	NTP	√	√	√	Network Time Protocol
161	SNMP-server	√			Simple Network Management Protocol
162	SNMP-client	√			Simple Network Management Protocol

Fig 3. Well known port numbers

User Datagram Protocol (UDP)

- **UDP is connectionless and unreliable protocol**
- **UDP is simple protocol using minimum overhead**

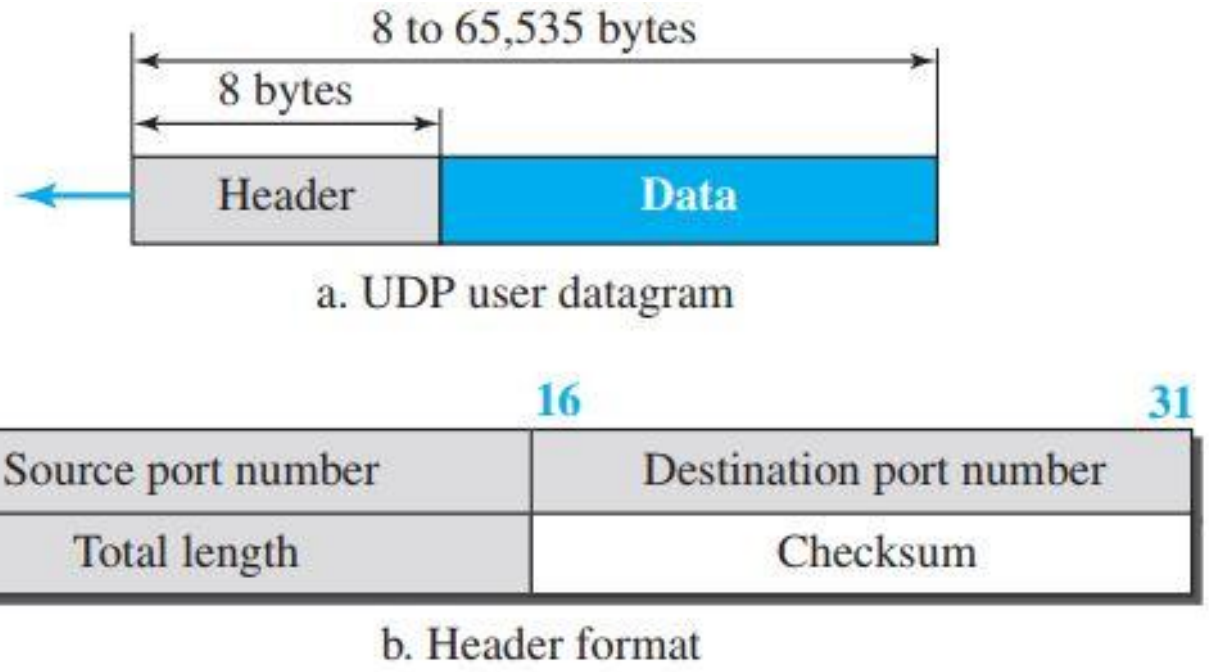

Fig 4. UDP packet format

UDP Services

- **Connectionless services**
  - UDP is an independent datagram
  - No connection establishment, no connection termination
  - Processes sending short messages, messages less than 65,507 bytes (65,535 minus 8 bytes for the UDP header and minus 20 bytes for the IP header), can use UDP

- **Flow control**
  - No flow control and hence no window mechanism
  - Receiver may overflow with incoming messages

- **Error control**
  - No error control mechanism except for checksum
  - When error is detected using checksum, the UDP datagram is discarded

UDP Services

- **Congestion control**
  - UDP is connectionless and hence doesn't provide congestion control

- **Encapsulation and Decapsulation**

- **Queueing**

UDP Applications

- UDP is suitable for a process that requires simple request response communication
- UDP is suitable for a process with internal flow and error-control mechanism
- UDP is suitable transport protocol for multicasting
- UDP is suitable for management process (like SNMP)
- UDP is suitable for route updating protocols like RIP
- UDP is suitable for real-time applications

LNMIIT
The LNM Institute of Information Technology

Transmission Control Protocol (TCP)

- **TCP is a connection oriented, reliable protocol**
- **It has connection establishment, data transfer, and connection termination phase**
- **TCP provides a full-duplex service**
  - Application data can flow from process A to process B at the same time when application data is flowing from process B to process A
- **TCP is point to point i.e., from single sender to single receiver**
- **TCP follows a three-way handshake**
- **Once the connection is made, the two application processes can send data to each other**

Transmission Control Protocol (TCP)

- **Once the data passes through the door (socket), the data is in hand of TCP running in the client**

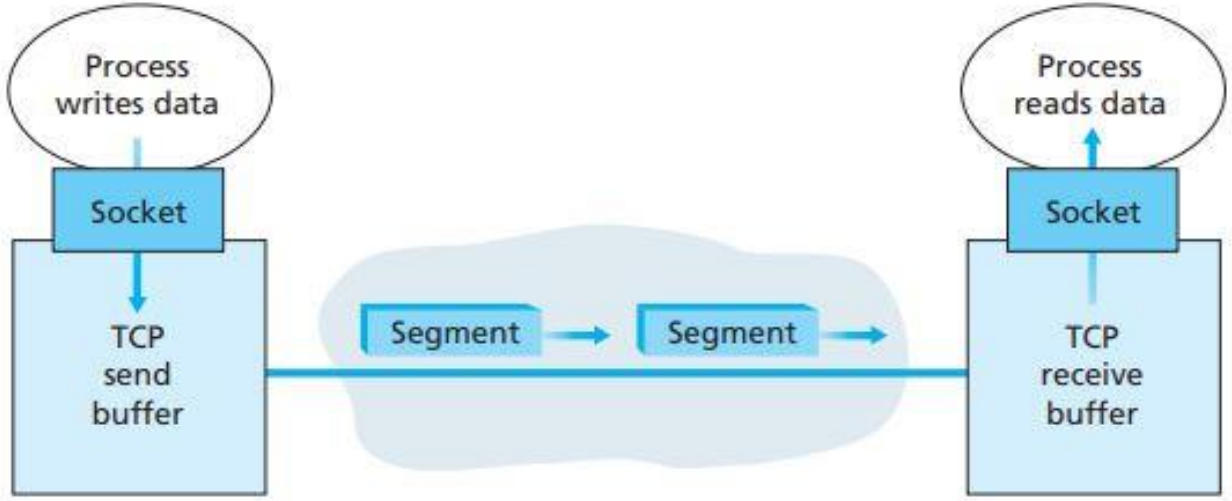

Fig 5. TCP buffer

- **TCP directs the data to the send buffer. It grab chunks of data from the send buffer and pass it to network layer**

- **The maximum amount of data that can be grabbed and placed in the segment is limited by Maximum Segment Size (MSS)**

Transmission Control Protocol (TCP)

- MSS is the maximum amount of application data in the segment, excluding header

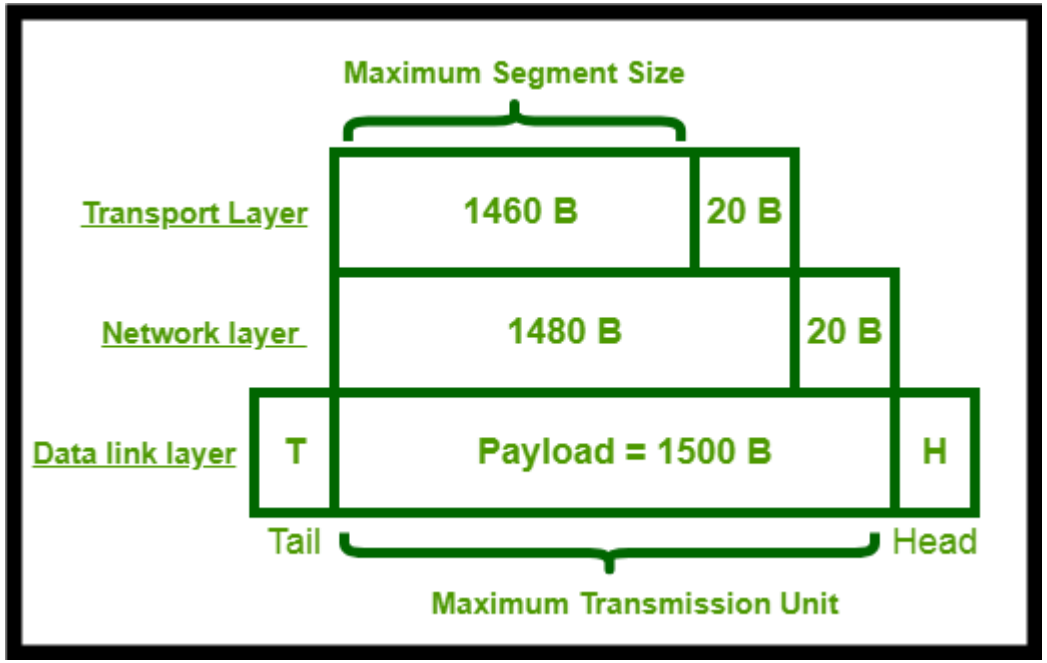

Fig 6. Maximum Segment Size

- TCP pairs each chunk of data with a TCP header and form a TCP segment
- Segments are passed down to network layer and are encapsulated within IP layer datagrams
- At receiver, a segment's data is placed at the buffer

TCP Services

- TCP delivers and receives data as stream of bytes
- TCP creates environment where two processes seem to be connected by an imaginary tube

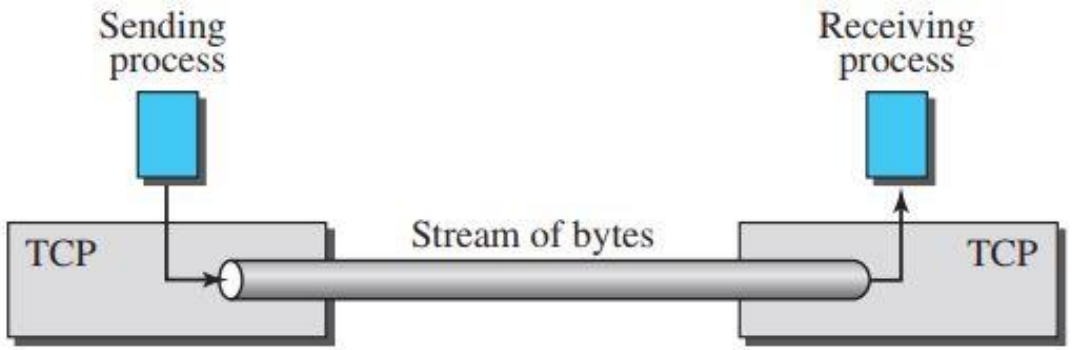

Fig 7. TCP stream of bytes

- The sending and receiving processes writes and reads with different rate which creates demand of buffer
- The buffers are hundreds or thousands of bytes depending on implementation

Sending and Receiving Buffers

- At sender, the buffer has three chambers
- The white section has empty chambers that are filled by the sending process
- The blue section holds bytes that have been sent but have yet to be acknowledged
- The black section shows bytes that have been written but not yet sent
- The operation of the buffer at the receiver is simpler

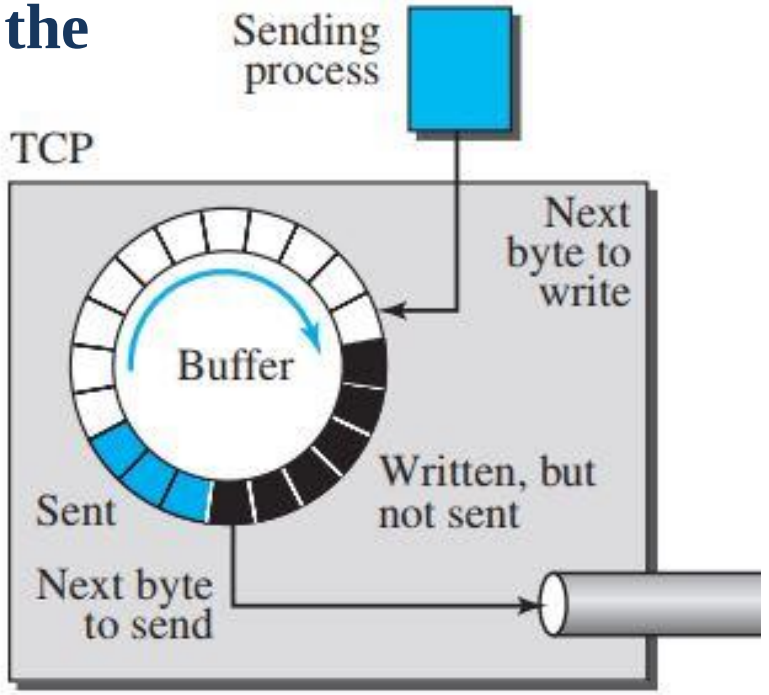

Fig 8. TCP sender buffer

Sending and Receiving Buffers

- **The white area contains empty chambers to be filled by bytes received from network**

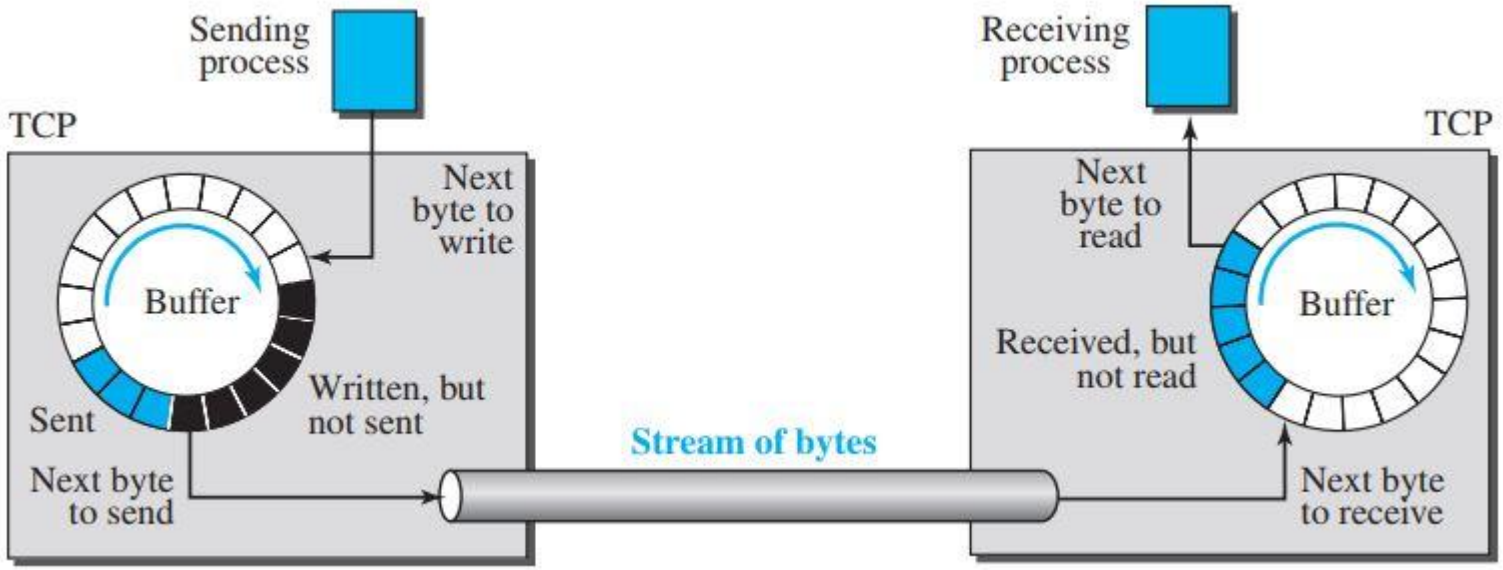

Fig 9. TCP sender and receiver buffer

- **The blue sections contain received bytes that is read by the receiving process**

- **When a byte is read by the receiving process, the chamber is recycled and added to the pool of empty chambers**

Sending and Receiving Buffers

- **Network layer as a service provider for TCP, send data as packets, not as a stream of bytes**

- **TCP groups a number of bytes together into a packet called a segment**

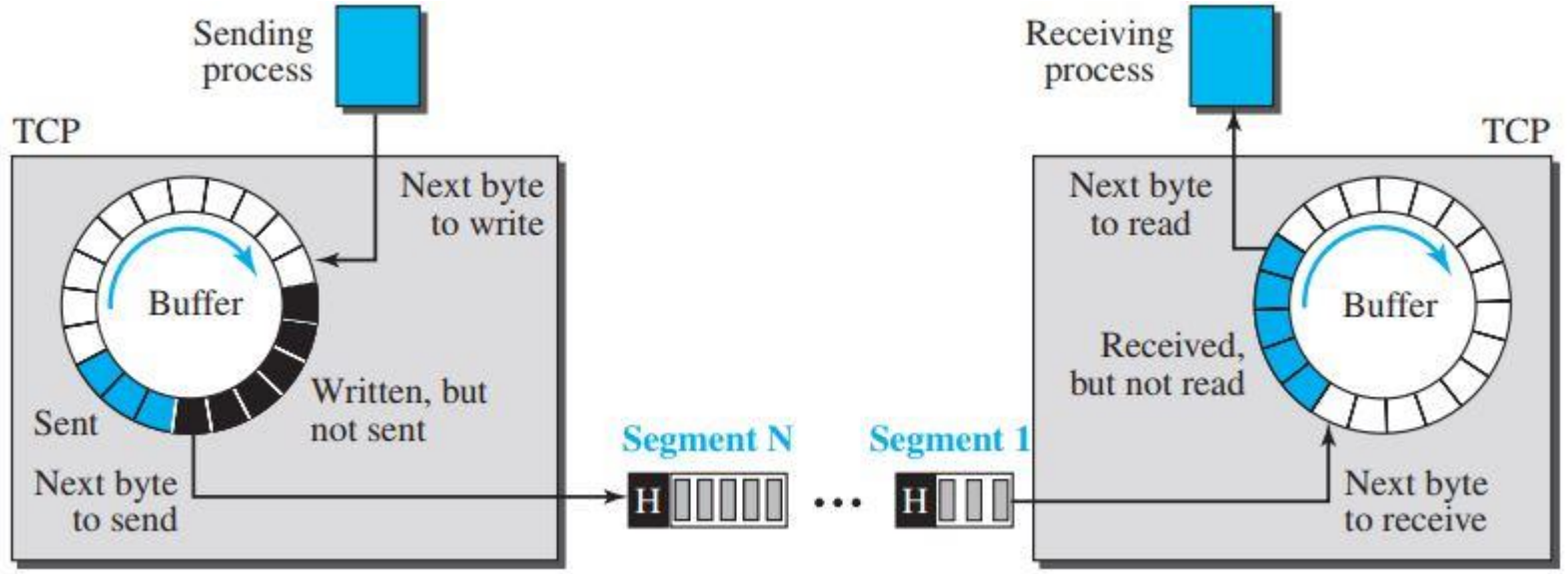

Fig 10. TCP segments

TCP Services

- **Full-Duplex Communication**
  - Data can flow in both the directions at the same time
  - TCP end points has its own sending and receiving buffer

- **Multiplexing and Demultiplexing**
  - Multiplexing at sender and demultiplexing at receiver
  - A connection needs to be established for each pair of processes

- **Connection-Oriented and Reliable Service**

TCP Features

- TCP header has two fields called *sequence number* and *acknowledgement number*
- TCP numbers all data bytes that are transmitted in a connection
- Numbering is independent in each direction
- TCP receives bytes of data from a process, TCP stores them in sending buffer and number them
- Numbering doesn't necessarily starts from 0, TCP can choose any arbitrary number between 0 and $2^{32} - 1$
- For example, if the chosen number is 507 and the total data to be sent is 5000 bytes, then the bytes are numbered from 507 to 5506

TCP Sequence Number

- **TCP assigns a sequence number to each segment that is being sent**

LNMIIT
The LNM Institute of Information Technology

Suppose a TCP connection is transferring a file of 5000 bytes. The first byte is numbered 10001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1000 bytes?

Solution

The following shows the sequence number for each segment:

Segment 1	→	Sequence Number:	10001	**Range:**	10001	to	11000
Segment 2	→	Sequence Number:	11001	**Range:**	11001	to	12000
Segment 3	→	Sequence Number:	12001	**Range:**	12001	to	13000
Segment 4	→	Sequence Number:	13001	**Range:**	13001	to	14000
Segment 5	→	Sequence Number:	14001	**Range:**	14001	to	15000

The value in the sequence number field of a segment defines the number assigned to the first data byte contained in that segment.

TCP Acknowledgement Number

- **TCP is full duplex, when connection is established, both parties can send and receive data at the same time**
- **Sequence number in each direction shows the number of first byte carried by the segment**
- **Each party uses an acknowledgement number to confirm the bytes it has received**
- **The acknowledgement number defines the number of the next byte that the party is expecting**
- **The acknowledgement number is cumulative**
  - ***If a party uses an acknowledgment number 6880, it has received all bytes from starting till 6879. It doesn't mean that the party has received a total 6879 bytes because the first byte number need not to be 0***