Transport layer

- Role in providing communication services directly to application processes
- Goal of transport layer is to provide reliable and cost effective services
- Functions : extending network layers delivery service,controlling transmission rate of transport layer entities in order to avoid or recover from congestion within network, transport-layer protocol provides for logical communication between application processes running on different hosts(hosts may be on opposite sides of the planet, connected via numerous routers and a wide range of link types.)

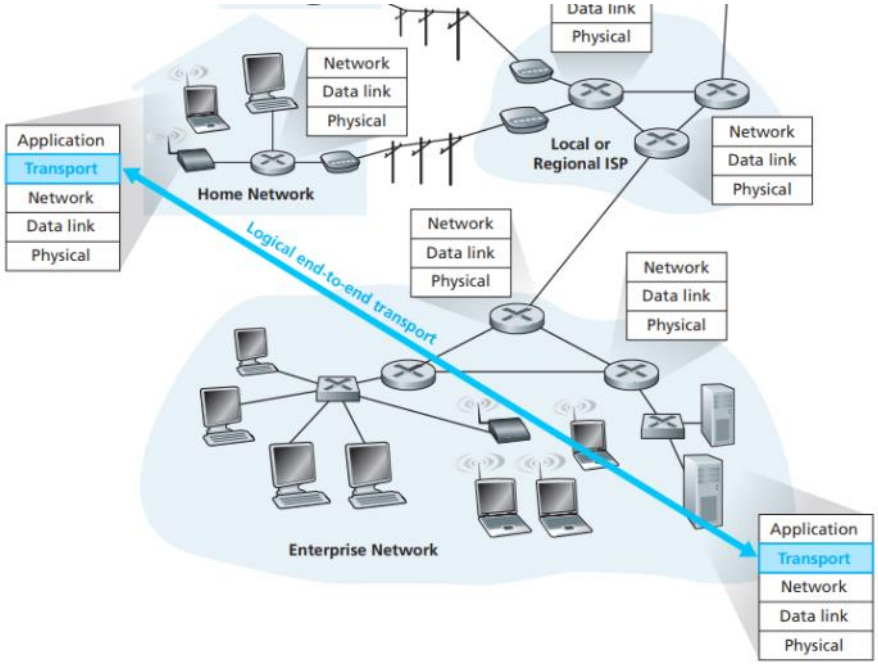


Figure 3.1 ♦ The transport layer provides logical rather than physical communication between application processes

- Multiplexing and demultiplexing:
  - Receiving side the transport layer redirects the segment to appropriate socket by examining appropriate fields in segments. Job of delivering the data in a transport-layer segment to the correct socket is called **demultiplexing**.
  - The job of gathering data chunks at the source host from different sockets, encapsulating each data chunk with header information (that will later be used in demultiplexing) to create segments, and passing the segments to the network layer is called **multiplexing**.

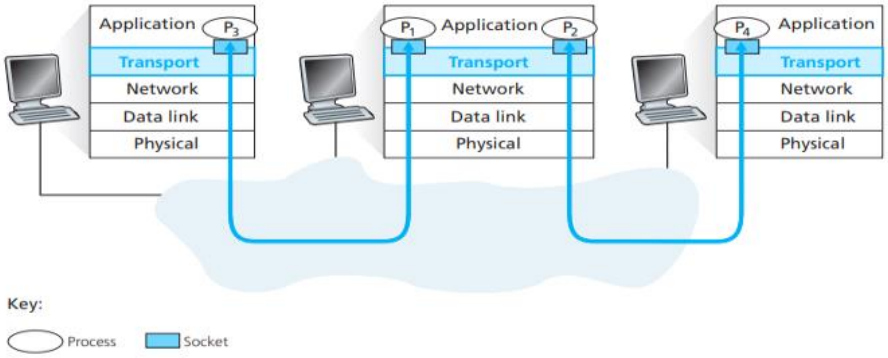
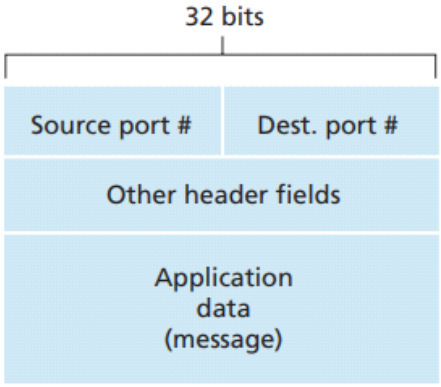
➢ 

Figure 3.2 ♦ Transport-layer multiplexing and demultiplexing

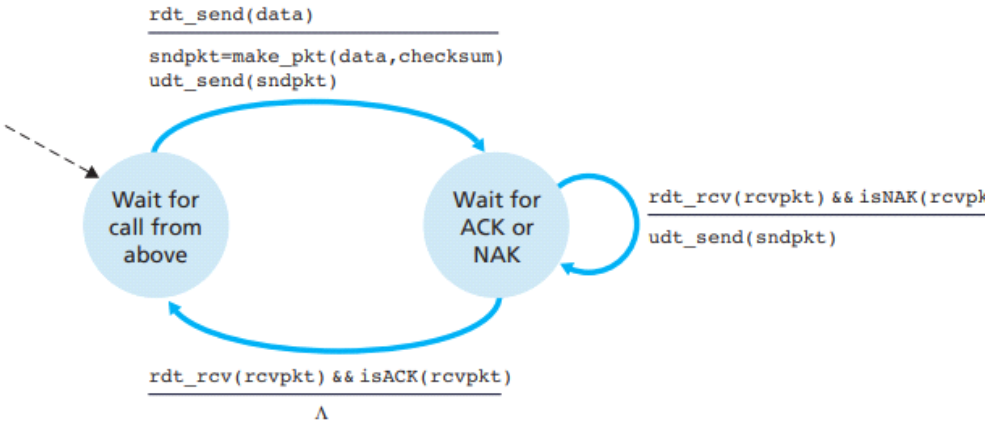
➢ Working of multiplexing and demultiplexing: transport-layer multiplexing requires (1) that sockets have unique identifiers, and (2) that each segment have special fields that indicate the socket to which the segment is to be delivered. Special fields are the source port number field and the destination port number field. The port numbers ranging from 0 to 1023 are called well-known port numbers and are restricted, which means that they are reserved for use by well-known application protocols such as HTTP (which uses port number 80) and FTP (which uses port number 21).

➢ 

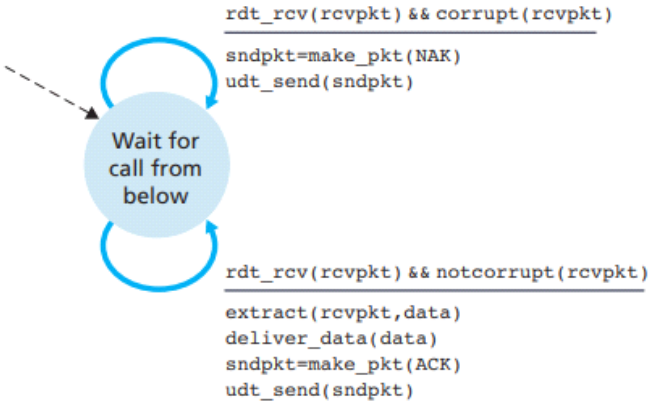
Source and destination port-number fields in a transport-layer segment

➢ Working of demultiplexing: when a segment arrives at the host, the transport layer examines the destination port number in the segment and directs the segment to the corresponding socket. The segment's data then passes through the socket into the attached process

➢ Principles of reliable communication:
  - Stop and wait:



a. rdt2.0: sending side



b. rdt2.0: receiving side

igure 3.10 ♦ rdt2.0–A protocol for a channel with bit errors

The receiver-side FSM for rdt2.0 still has a single state. On packet arrival, the receiver replies with either an ACK or a NAK, depending on whether or not the received packet is corrupted. rdt_rcv(rcvpkt) && corrupt(rcvpkt) corresponds to the event in which a packet is received and is found to be in error

- ACK or NAK packet could be corrupted
- add a new field to the data packet and have the sender number its data packets by putting a sequence number into this field. The receiver then need only check this sequence number to determine whether or not the received packet is a retransmission

- GO back N (GBN), sliding window protocol
- :
  - Allowed to transfer multiple packets without waiting for ack, but constrained to a N number of unacknowledged packets.

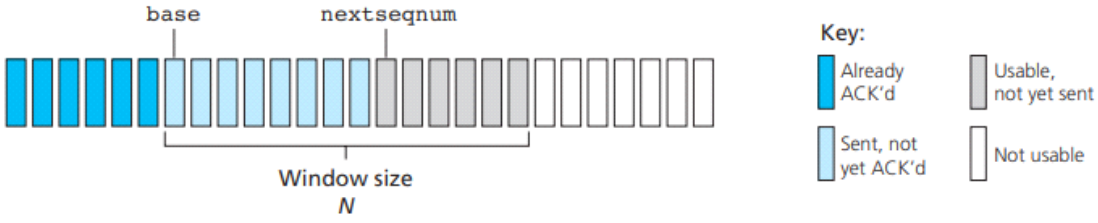


Figure 3.19 ♦ Sender's view of sequence numbers in Go-Back-N

- Base to be the sequence number of the oldest unacknowledged

packet and *nextseqnum* to be the smallest unused sequence number (that is, the sequence number of the next packet to be sent), then four intervals in the range of sequence numbers can be identified. Sequence numbers in the interval *[0,base-1]* correspond to packets that have already been transmitted and acknowledged. The interval *[base,nextseqnum-1]* corresponds to packets that have been sent but not yet acknowledged. Sequence numbers in the interval *[nextseqnum,base+N-1]* can be used for packets that can be sent immediately, should data arrive from the upper layer. Finally, sequence numbers greater than or equal to *base+N* cannot be used until an unacknowledged packet currently in the pipeline (specifically, the packet with sequence number base) has been acknowledged

- Sender

  Window-based transmission: The sender maintains a window of size N, which specifies the maximum number of packets it can transmit before waiting for acknowledgments (ACKs) from the receiver. This allows for pipelined transmission, improving efficiency by utilizing the bandwidth while waiting for ACKs.

  Cumulative acknowledgments: The GBN protocol uses cumulative ACKs. An ACK for packet n signifies that all packets up to and including sequence number n have been received correctly. This simplifies the sender's logic as it only needs to track the next expected sequence number (nextseqnum) based on the received ACKs.

  Timeout and retransmission: The sender employs a timer for each transmitted packet. If a timeout occurs for a packet (no ACK received within a predefined time), the sender assumes the packet or ACK is lost and retransmits all packets within the current window. This ensures that missing packets are eventually delivered.

- Receiver

  In-order delivery: The receiver only accepts and delivers packets to the upper layer if they arrive in the expected sequence. This guarantees that the data is presented to the application in the correct order.

  Discarding out-of-order packets: Even if a packet is correctly received, the receiver discards it if it arrives out-of-order. This simplifies the receiver's logic as it eliminates the need for complex buffering mechanisms to handle out-of-order packets. However, this can lead to wasted bandwidth if the discarded packets are not resent successfully on the first attempt.

  Selective ACKs (implicit): While not explicitly mentioned in the text, GBN relies on implicit selective acknowledgments through the cumulative ACK strategy. An ACK for n implies all previous packets were received correctly, while the absence of an ACK for a specific sequence number indicates a gap and the need for retransmission.

  Overall, the GBN protocol offers a balance between complexity and efficiency. It achieves reliable data transfer with a simpler sender implementation compared to protocols that require buffering for out-of-order packets. However, the downside is the potential for wasted bandwidth due to discarded out-of-order packets and retransmissions.

- techniques include the use of sequence numbers, cumulative acknowledgments, checksums, and a timeout/retransmit operation.

- Connection-oriented transport : TCP
  - Connection oriented because two processes handshake before sending data.
  - Supports full duplex(parallel data transfer), point to point(one sender to one receiver)
  - Client sends request as TCP segment and server sends back TCP response segment. And a third segment from the client contains the payload(three way handshake).

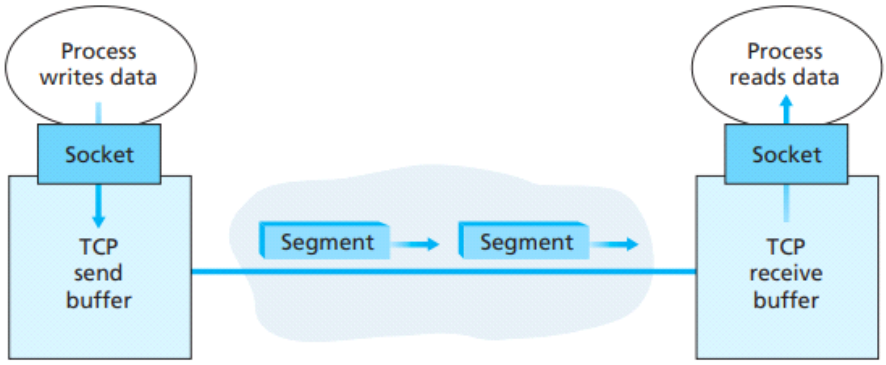
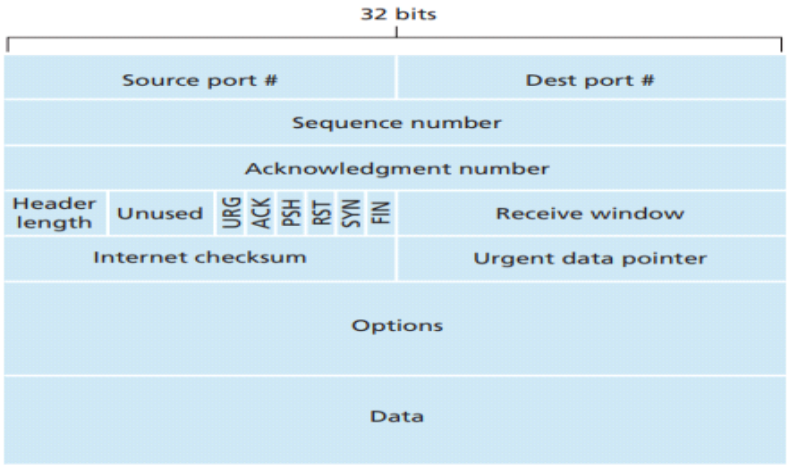
- 

Figure 3.28 ♦ TCP send and receive buffers

- TCP segments - client data with TCP header.
- TCP segment structure



TCP Segment Structure:
TCP segments, the basic units of data transfer in TCP connections, have a header containing various control fields. Two crucial fields for reliable data delivery are:

- **Sequence Number (32 bits):** This number identifies the absolute position of the first byte of data in the segment within the entire byte stream being transmitted. It essentially assigns a unique number to each byte in the data stream.
- **Acknowledgment Number (32 bits):** This number signifies the next byte that the sender expects to receive from the other side. It acts as an acknowledgment for successfully received data up to a certain point in the stream.

Understanding Sequence Numbers:

- **Byte Stream View:** TCP treats data as an unstructured but ordered stream of bytes. Sequence numbers reflect this concept by tracking the byte position within the stream, not the individual segments.
- **Assigning Sequence Numbers:** The sequence number for a segment corresponds to the byte stream number of the first byte it carries. For instance, if the first byte in a data stream is numbered 0 and the Maximum Segment Size (MSS) is 1000 bytes, the first segment will have a sequence number of 0, the second 1000, and so on.

Acknowledgment Numbers and Reliable Delivery:

- **Cumulative Acknowledgment:** Unlike sequence numbers that track absolute byte positions, acknowledgment numbers employ a cumulative acknowledgment strategy. The number acknowledges all bytes received correctly up to (but not including) the specified sequence number.
- **Example:** Imagine Host A has received all bytes numbered 0 through 535 from Host B and is about to send a segment. Since Host A is waiting for byte 536 and subsequent bytes, it includes 536 in the acknowledgment number field of the segment it sends to B.

Out-of-Order Delivery and Receiver Handling:

TCP allows segments to arrive out of order due to network variations. The text acknowledges that TCP specifications don't dictate how receivers handle these segments. Here are two common approaches:

1. **Discarding Out-of-Order Segments:** This simplifies receiver design but can lead to wasted bandwidth if the discarded segments need retransmission.
2. **Buffering Out-of-Order Segments:** Receivers can store out-of-order segments and wait for missing bytes to fill the gaps. This is more bandwidth-efficient but requires more complex receiver logic.

In practice, TCP implementations typically buffer out-of-order segments for efficient data delivery.

Choosing Initial Sequence Numbers:

The text mentions that both sides of a TCP connection choose an initial sequence number randomly. This helps prevent mistaking segments from old, terminated connections for valid segments in new connections that reuse the same port numbers.

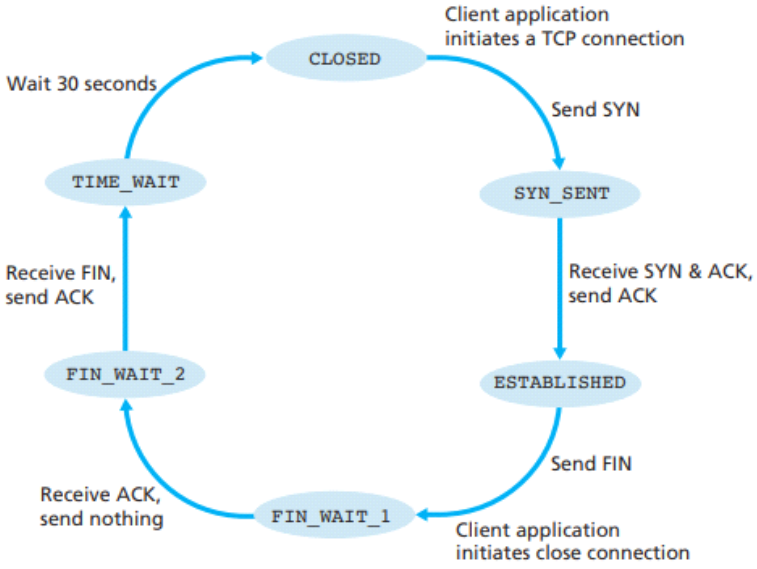


Figure 3.41 ♦ A typical sequence of TCP states visited by a client TCP

-

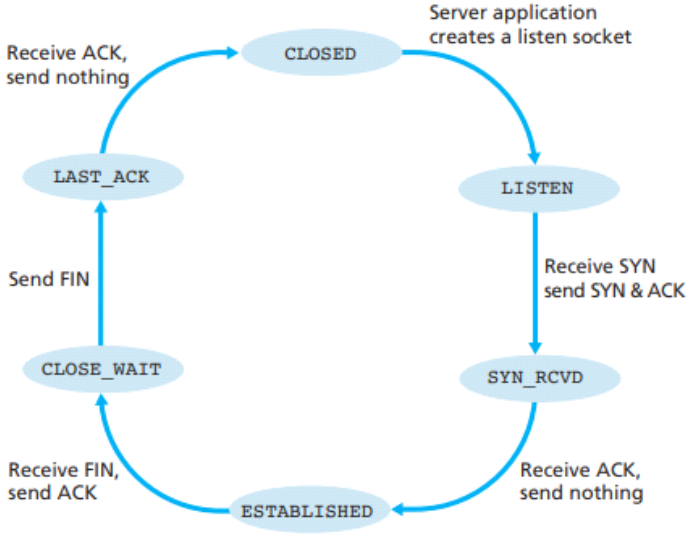
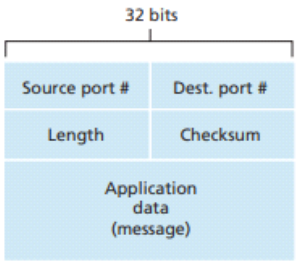


Figure 3.42 ♦ A typical sequence of TCP states visited by a server-side TCP

- ~Connection less transport : UDP
  - In here the devices are communicating with only IP.

- 

- It just takes the data to send and add source and destination port
- Dns uses udp
- UDP sends data immediately once an application passes it, without any delay. It does not establish a connection, maintain any connection state, or track any parameters. This makes UDP faster and allows a server to support more active clients.

3.3.1 UDP Segment Structure

The UDP segment structure is defined in RFC 768.
The application data occupies the data field of the UDP segment. For example, for DNS, the data field contains either a query message or a response message.
The UDP header has only four fields, each consisting of two bytes: Source port, Destination port, Length, and Checksum.
The port numbers allow the destination host to pass the application data to the correct process running on the destination end system.
The length field specifies the number of bytes in the UDP segment (header plus data).
The checksum is used by the receiving host to check whether errors have been introduced into the segment.
3.3.2 UDP Checksum

The UDP checksum provides for error detection.
UDP at the sender side performs the 1s complement of the sum of all the 16-bit words in the segment, with any overflow encountered during the sum being wrapped around. This result is put in the checksum field of the UDP segment.
At the receiver, all four 16-bit words are added, including the checksum. If no errors are introduced into the packet, then the sum at the receiver will be 1111111111111111. If one of the bits is a 0, then we know that errors have been introduced into the packet.
The reason why UDP provides a checksum is that there is no guarantee that all the links between source and destination provide error checking. Furthermore, even if segments are correctly transferred across a link, it's possible that bit errors could be introduced when a segment is stored in a router's memory. Given that neither link-by-link reliability nor in-memory error detection is guaranteed, UDP must provide error detection at the transport layer, on an end-end basis, if the end-end data transfer service is to provide error detection. This is an example of the celebrated end-end principle in system design [Saltzer 1984], which states that since certain functionality (error detection, in this case) must be implemented on an end-end basis.
. Although UDP
provides error checking, it does not do anything to recover from an error.

Principles of conjunction control:

Congestion control is a critical aspect of network communication. It addresses the problem of packet loss typically resulting from the overflowing of router buffers when the network becomes congested. While packet retransmission can address the symptom of network congestion (the loss of a specific transport-layer segment), it does not treat the cause of network congestion, which is too many sources attempting to send data at too high a rate.

To treat the cause of network congestion, mechanisms are needed to throttle senders in the face of network congestion. Various approaches can be taken to

avoid, or react to, network congestion. The performance received by upper-layer applications is a manifestation of network congestion.

The text also mentions the available bit-rate (ABR) service in asynchronous transfer mode (ATM) networks and indicates a forthcoming discussion on TCP's congestion-control algorithm. This suggests that different network protocols and services may have different strategies for managing congestion.

In essence, congestion control is about managing network resources efficiently to prevent packet loss and ensure reliable data transfer. It's a fundamental problem in networking that requires careful consideration and effective solutions.

The principles of congestion control revolve around two key concepts: efficiency and fairness.

Efficiency refers to the optimal use of network resources. The goal is to maximize the throughput, which is the rate of successful message delivery over a communication channel. High efficiency means the network can handle a large amount of data transmission without becoming congested1.

Fairness refers to the equitable allocation of network resources among all users or applications. In a fair system, all users get a reasonable share of the network resources. There are different interpretations of what is considered fair, and different congestion control algorithms may aim for different types of fairness. For example, max-min fairness prioritizes the user with the least allocation, while proportional fairness allocates resources proportional to the users' demands23.

In the context of congestion control, these principles are applied in the design of algorithms that manage data transmission rates to prevent network congestion. For instance, TCP (Transmission Control Protocol) uses a congestion window and adjusts its size based on network conditions to prevent or alleviate network congestion1.

In network-assisted congestion control, feedback from network-layer components (like routers) is used to adjust the data transmission rate. For example, in ATM (Asynchronous Transfer Mode) available bit-rate (ABR) congestion control, a router can inform the sender explicitly of the transmission rate it can support on an outgoing link1.

It's important to note that achieving both high efficiency and fairness in a network can be challenging, as these two goals can sometimes be in conflict. For example, striving for complete fairness might lead to underutilization of network resources, reducing efficiency23.

- Approaches to control conjunction:
  - End-to-End Congestion Control: In this approach, the network layer does not provide explicit support to the transport layer for congestion control. The presence of congestion in the network must be inferred by the end systems based on observed network behavior, such as packet loss and delay. An example of this approach is TCP, which takes this end-to-end approach towards congestion control as the IP layer provides no feedback to the end systems regarding network congestion. TCP interprets segment loss (indicated by a timeout or a triple duplicate acknowledgment) as a sign of network congestion and decreases its window size accordingly. There are also more recent proposals for TCP congestion control that use increasing round-trip delay values as indicators of increased network congestion.

  - Network-Assisted Congestion Control: In this approach, network-layer components (like routers) provide explicit feedback to the sender about the congestion state in the network. This feedback can be as simple as a single bit indicating congestion at a link. This approach was taken in the early IBM SNA

and DEC DECnet architectures, and has been proposed for TCP/IP networks. It is also used in ATM available bit-rate (ABR) congestion control. More sophisticated network feedback is also possible. For example, one form of ATM ABR congestion control allows a router to inform the sender explicitly of the transmission rate it can support on an outgoing link. The XCP protocol provides router-computed feedback to each source, carried in the packet header, regarding how that source should increase or decrease its transmission rate