UNIT III
TRANSPORT LAYER

Introduction:

The network layer provides end-to-end packet delivery using data-grams or virtual circuits. The transport layer builds on the network layer to provide data transport from a process on a source machine to a process on a destination machine with a desired level of reliability that is independent of the physical networks currently in use. It provides the abstractions that applications need to use the network.

Transport Entity: The hardware and/or software which make use of services provided by the network layer, (within the transport layer) is called transport entity.

Transport Service Provider: Layers 1 to 4 are called Transport Service Provider.

Transport Service User: The upper layers i.e., layers 5 to 7 are called Transport Service User.

Transport Service Primitives: Which allow transport users (application programs) to access the transport service.

TPDU (Transport Protocol Data Unit): Transmissions of message between 2 transport entities are carried out by TPDU. The transport entity carries out the transport service primitives by blocking the caller and sending a packet the service. Encapsulated in the payload of this packet is a transport layer message for the server's transport entity. The task of the transport layer is to provide reliable, cost-effective data transport from the source machine to the destination machine, independent of physicalnetwork or networks currently in use.

TRANSPORT SERVICE

1.Services Provided to the Upper Layers

The ultimate goal of the transport layer is to provide efficient, **reliable, and cost-effective data transmission** service to its users, normally processes in the application layer. To achieve this, the transport layer makes use of the **services pro-vided by the network layer.** The software and/or hardware within the transport layer that does the work is called the **transport entity.** The transport entity can be located in the operating system kernel, in a library package bound into network applications, in a separate user process, or even on the network interface card.

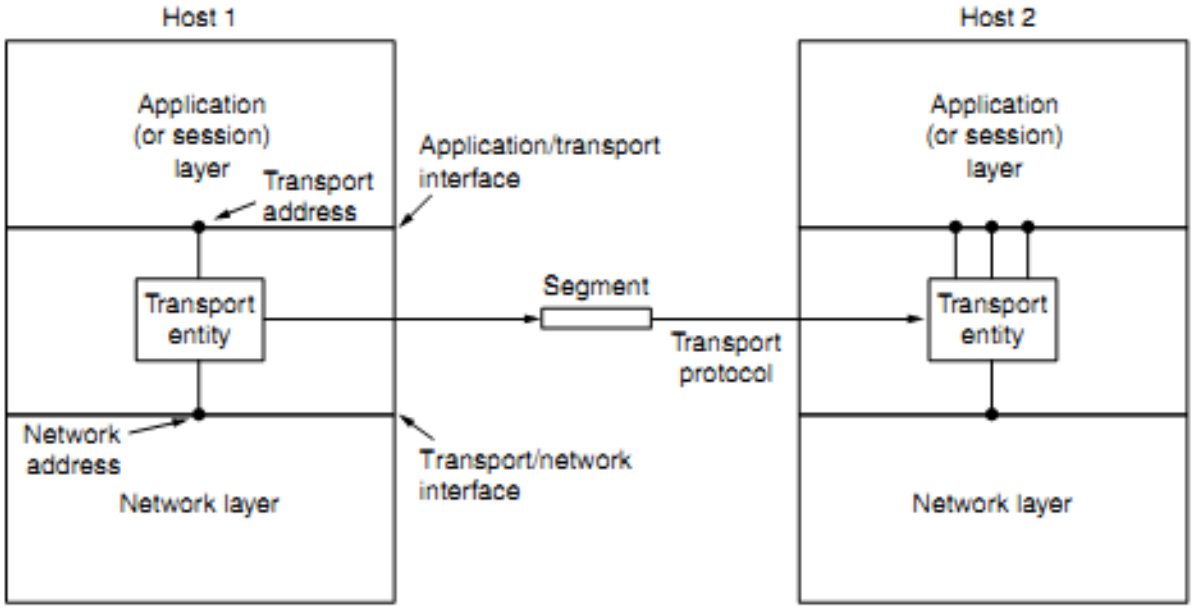

Fig 4.1: The network, Application and transport layer

There are two types of network service

o Connection-oriented

o Connectionless

Similarly, there are also two types of transport service. The connection-oriented transport service is similar to the connection-oriented network service in many ways.

In both cases, connections have three phases:

o Establishment

o Data transfer

o Release.

 Addressing and flow control are also similar in both layers. Furthermore, the connectionless transport service is also very similar to the connectionless network service.

 The bottom four layers can be seen as the transport service provider, whereas the upper layer(s) are the transport service user.

2. Transport Service Primitives

➢ To allow users to access the transport service, the transport layer must provide some operations to application programs, that is, a transport service interface. Each transport service has its own interface.

- The transport service is similar to the network service, but there are also some important differences.
- The **main difference** is that the network service is intended to model the service offered by real networks. Real networks can lose packets, so the network service is generally **unreliable.**
- The (connection-oriented) transport service, in contrast, is **reliable**

As an example, consider two processes connected by pipes in UNIX. They assume the connection between them is perfect. They do not want to know about acknowledgements, lost packets, congestion, or anything like that. What they want is a 100 percent reliable connection. Process A puts data into one end of the pipe, and process B takes it out of the other.
A **second difference** between the network service and transport service is **whom the services are intended for**. The network service is used only by the transport entities. Consequently, the transport service must be convenient and easy to use.

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

Table:4.1 - The primitives for a simple transport service.
Eg: Consider an application with a server and a number of remote clients.
1. The server executes a “LISTEN” primitive by calling a library procedure that makes a

System call to block the server until a client turns up.
2. When a client wants to talk to the server, it executes a “CONNECT” primitive, with “CONNECTION REQUEST” TPDU sent to the server.

3. When it arrives, the TE unblocks the server and sends a “CONNECTION ACCEPTED” TPDU back to the client.

4. When it arrives, the client is unblocked and the connection is established. Data can now be exchanged using “SEND” and “RECEIVE” primitives.

5. When a connection is no longer needed, it must be released to free up table space within the 2 transport entries, which is done with “DISCONNECT” primitive by sending “DISCONNECTION REQUEST”

MULTIPLEXING:

In networks that use virtual circuits within the subnet, each open connection consumes some table space in the routers for the entire duration of the connection. If buffers are dedicated to the virtual circuit in each router as well, a user who left a terminal logged into a remote machine, there is need for multiplexing. There are 2 kinds of multiplexing:

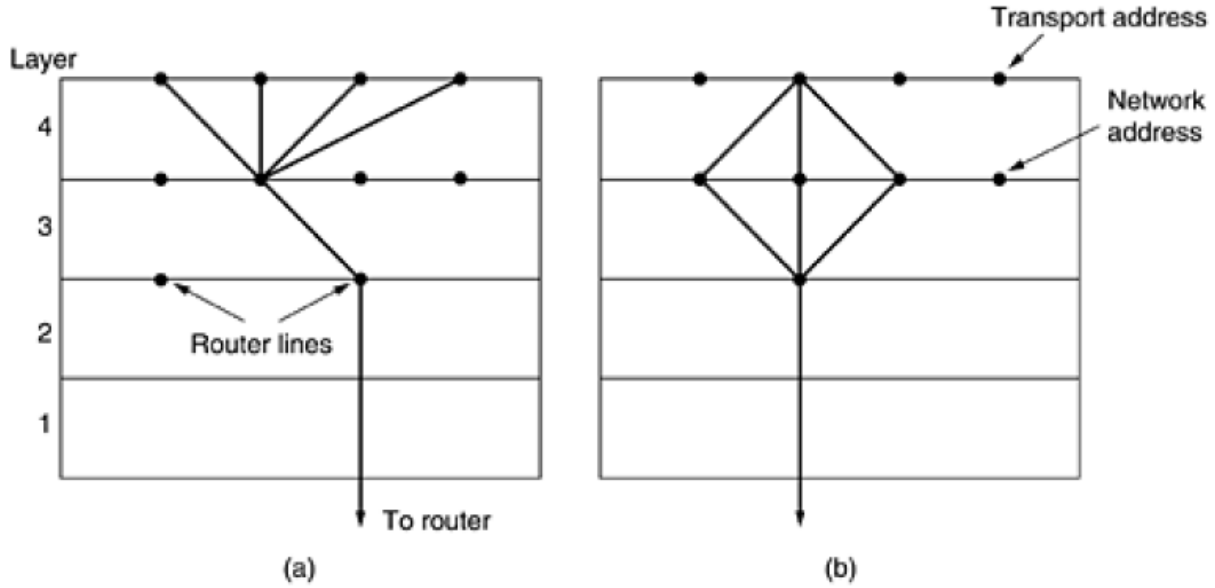

Figure 4.8. (a) Upward multiplexing. (b) Downward multiplexing

(a). UP-WARD MULTIPLEXING:

In the below figure, all the 4 distinct transport connections use the same network connection to the remote host. When connect time forms the major component of the carrier's bill, it is up to the transport layer to group port connections according to their destination and map each group onto the minimum number of port connections.

(b). DOWN-WARD MULTIPLEXING:

 If too many transport connections are mapped onto the one network connection, the

performance will be poor.

 If too few transport connections are mapped onto one network connection, the service

will be expensive.

The possible solution is to have the transport layer open multiple connections and distribute the traffic among them on round-robin basis, as indicated in the below figure:
With ‘k’ network connections open, the effective band width is increased by a factor of ‘k’.

Connection Oriented Transport service(TCP)

Connection oriented services modeled after the telephone system.

To use a connection-oriented network service, the service user first establishes a connection, uses the connection and then releases the connection.

The essential aspect of a connection is that it acts like a tube: the sender pushes objects (bits) in at one end, and the receiver text them out at the other end. In most cases the order is preserved so that the bits arrive in the order they were sent.

In some cases when a connection is established, the sender receiver and the subnet conduct a negotiation about parameters to be used, such as maximum message size, quality of service required and other issues. Typically, one side makes a proposal and Other Side can accept it reject it or make a counter proposal.

Reliable connection oriented services has to minor variations : **reliable message streams** example: sequence of pages, **Reliable byte stream** example: remote login and **unreliable connection** digitized voice.

Connection oriented transmission has three stages. These are:

connection establishment: The connection oriented services, before transmitting data first the sender has to establish the connection by which data can be sent.

data transfer: after the connection gets established, the sender starts sending data packets to the receiver.

connection termination: after all the data gets transferred the connection has to be terminated

Connectionless Services Transport service(UDP)

Connectionless : connectionless service is model after the Postal System.

Each message (letter) carries the full destination address, and each one is routed through the system independent of all the others. Normally when two messages are sent to the system destination, the first one sent will be the first one to arrive. however it is possible that the first one sent can be delayed so that the second one arrives first.

Each service can be characterized by a quality of service. some services are reliable in the sense that they never lose data. Usually, a reliable services implemented by having the receiver acknowledge the receipt of each message so the sender is sure that it read arrived.

Unreliable (meaning not acknowledged) connectionless service is often called **datagram service** in analogy with Telegram service, which also does not return and acknowledgement to the sender example emails.

The **acknowledged datagram service** can be provided for these applications. It is like sending a registered letter and requesting a return receipt. When the receipt comes back, the sender is absolutely sure that the letter was delivered to the intended party and not losing along the way. Example: Registered Mail.

The request-reply service, In this service the sender transmits a single datagram containing a request; that reply contains the answer, Ex: Database Query.

TCP CONGESTION CONTROL:

TCP does to try to prevent the congestion from occurring in the first place in the following way:
When a connection is established, a suitable window size is chosen and the receiver specifies a window based on its buffer size. If the sender sticks to this window size, problems will not occur due to buffer overflow at the receiving end. But they may still occur due to internal congestion within the network. Let's see this problem occurs.

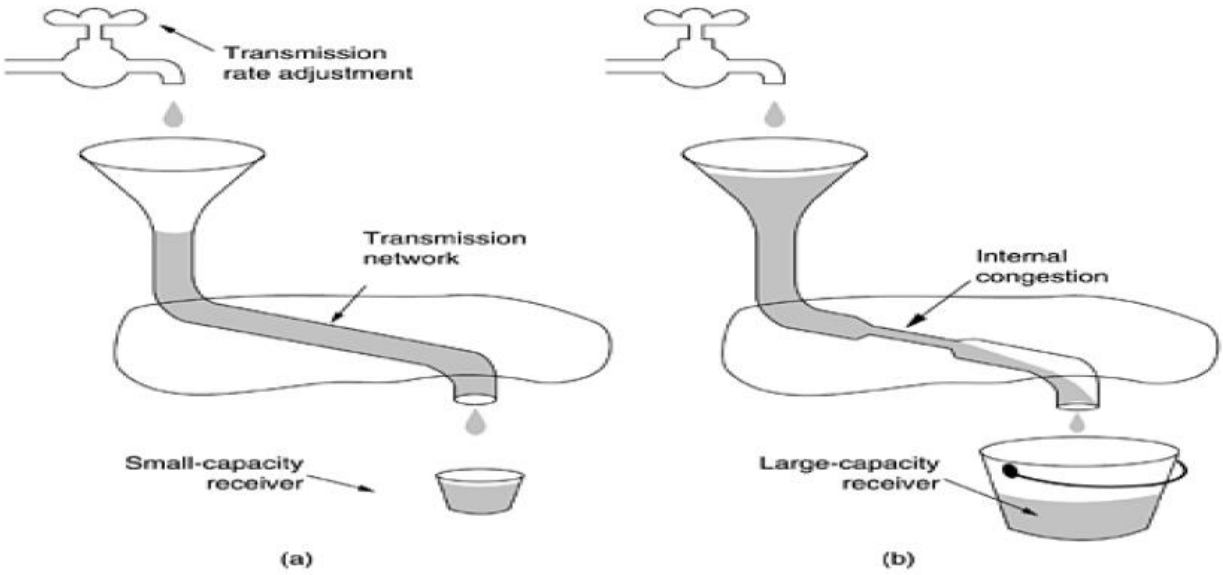

Figure 4.16. (a) A fast network feeding a low-capacity receiver. (b) A slow network feeding a high-capacity receiver.

In fig (a): We see a thick pipe leading to a small- capacity receiver. As long as the sender does not send more water than the bucket can contain, no water will be lost.
In fig (b): The limiting factor is not the bucket capacity, but the internal carrying capacity of the n/w. if too much water comes in too fast, it will backup and some will be lost.

 When a connection is established, the sender initializes the congestion window to the size of the max segment in use our connection.

 It then sends one max segment .if this max segment is acknowledged before the timer goes off, it adds one segment s worth of bytes to the congestion window to make it two maximum size segments and sends 2 segments.

 As each of these segments is acknowledged, the congestion window is increased by one max segment size.

 When the congestion window is 'n' segments, if all 'n' are acknowledged on time, the congestion window is increased by the byte count corresponding to 'n' segments.

 The congestion window keeps growing exponentially until either a time out occurs or the receiver's window is reached.

 The internet congestion control algorithm uses a third parameter, the **"threshold"** in addition to receiver and congestion windows.

Different congestion control algorithms used by TCP are:
- RTT variance Estimation.
- Exponential RTO back-off Re-transmission Timer Management
- Karn's Algorithm
- Slow Start
- Dynamic window sizing on congestion
- Fast Retransmit Window Management
- Fast Recovery

TCP TIMER MANAGEMENT:
TCP uses 3 kinds of timers:
1. Retransmission timer
2. Persistence timer
3. Keep-Alive timer.

1. Retransmission timer: When a segment is sent, a timer is started. If the segment is acknowledged before the timer expires, the timer is stopped. If on the other hand, the timer goes off before the acknowledgement comes in, the segment is retransmitted and the timer is started again. The algorithm that constantly adjusts the time-out interval, based on continuous measurements of n/w performance was proposed by JACOBSON and works as follows:

for each connection, TCP maintains a variable RTT, that is the best current estimate of the round trip time to the destination inn question.

- When a segment is sent, a timer is started, both to see how long the acknowledgement takes and to trigger a retransmission if it takes too long.
- If the acknowledgement gets back before the timer expires, TCP measures how long the measurements took say M
- It then updates RTT according to the formula

RTT = α RTT + (1-α) M
Where α = a smoothing factor that determines how much weight is given to the old value. Typically, α ***=7/8***
Retransmission timeout is calculated as
D = α D + (1-α) | RTT-M |
Where D = another smoothed variable, Mean RTT = expected acknowledgement value
M = observed acknowledgement value

Timeout = RTT+(4*D)

2. Persistence timer:

It is designed to prevent the following deadlock:

 The receiver sends an acknowledgement with a window size of '0' telling the sender to wait later, the receiver updates the window, but the packet with the update is lost now both the sender and receiver are waiting for each other to do something

 when the persistence timer goes off, the sender transmits a probe to the receiver the response to the probe gives the window size

 if it is still zero, the persistence timer is set again and the cycle repeats

 if it is non zero, data can now be sent

3. Keep-Alive timer: When a connection has been idle for a long time, this timer may go off to cause one side to check if other side is still there. If it fails to respond, the connection is terminated.

Principles of Reliable Data Transfer

The internet network layer provides only best effort service with no guarantee that packets arrive at their destination. Also, since each packet is routed individually it is possible that packets are received out of order. For connection-oriented service provided by TCP, it is necessary to have a reliable data transfer (RDT) protocol to ensure delivery of all packets and to enable the receiver to deliver the packets in order to its application layer.

A simple alternating bit RDT protocol can be designed using some basic tools. This protocol is also known as a stop-and-wait protocol: after sending each packet the sender stops and waits for feedback from the receiver indicating that the packet has been received.

Stop-and-wait RDT protocols have poor performance in a long-distance connection. At best, the sender can only transmit one packet per round-trip time. For a 1000 mile connection this amounts to approximately 1 packet (about 1500 bytes) every 20 ms. That results in a pathetic 75 KB per second rate.

To improve transmission rates, a realistic RDT protocol must use pipelining. This allows the sender to have a large number of packets "in the pipeline". This phrase refers to packets that have been sent but whose receipt has not yet verified by the receiver.

Principles

The following tools are essential for any RDT protocol implemented on top of an unreliable network.

Error detection

Sequence numbering

Feedback

Timers

Error Detection

The data link and network layers have error detection for detecting bit errors in packets. However, error detection schemes can never detect all errors so it is helpful to have additional error detection in the transport layer to reduce the frequency of undetected errors.

Lower layer protocols with error detection usually have a simple policy for dealing with errors: discard the packet. A transport layer RDT protocol typically just does the same thing, letting its algorithm for dealing with missing packets deal with the problem. Since the lower layers may discard packets an RDT protocol must allow for the possibility that a receiver is not even aware of an attempted transmission.

Sequence Numbering

Packets in the network layer are routed individually. This makes it possible that they are received in a different order than they are transmitted. Sequence numbering is essential for restoring the transmitted order.

Feedback

Feedback involves information sent by the receiver back to the sender about reception of sent packets. This is essential for recovery of missing packets. The feedback takes the form of acknowledgments (ACKs) with one of three forms:

Negative acknowledgment - "I did not receive the packet with sequence number sn."

Positive individual acknowledgment - "I received the packet with sequence number sn."

Positive cumulative acknowledgment - "I have received all packets with sequence numbers up to but not including sn."

Most reliable data transfer protocols use only one of these types of acknowledgment. Negative acknowledgments are useful in human communication, but only because the acknowledgment is not lost, though it may be garbled. Since negative acknowledgment packets can be lost in the internet, they are not useful. They will not be considered in this presentation.

Cumulative acknowledgments allow acknowledgment of numerous packets at a time. They can be useful in pipelined protocols.

Timers

Packet loss in the network layer does not discriminate between data packets and acknowledgment packets. A sender in a reliable data transfer protocol needs to set a timer for transmitted packets. Generally the sender does one of two things:

Resends a packet after a timer fires.

Sends a new packet after an acknowledgment (positive) arrives.

If an acknowledgment arrives before the timer fires the sender stops the timer so that it will not fire.