**UNIT-III**

Network layer Design issues, Routing algorithms, congestion control algorithms, Internetworking.

Transport layer design issues, connection management, Transport protocol X 25, session layer design issues, and Remote procedure cell.

**1. Network layer Design issues**

These issues include the service provided to the transport layer and the internal design of the subnet.

1. Store-and-Forward Packet Switching

2. Services Provided to the Transport Layer

3. Implementation of Connectionless Service

4. Implementation of Connection-Oriented Service

5. Comparison of Virtual-Circuit and Datagram Subnets

**1. Store-and-Forward Packet Switching**

The major components of the network are the ISP’s equipment (routers connected by transmission lines),Shown inside the shaded oval, and the customer’s equipment, shown outside the oval. Host H1 is directly connected to one of the ISP’s routers. In contrast, H2 is on a LAN with a router, F, owned and operated by the customer. This router also has a leased line to the ISP’s equipment. We have shown F as being outside the oval because it does not belong to the ISP’s.

This equipment is used as follows. A host with a packet to send transmits it to the nearest router, either on its own LAN or over a point-to-point link to the carrier. The packet is stored there until it has fully arrived so the checksum can be verified. Then it is forwarded to the next router along the path until it reaches the destination host, where it is delivered. This mechanism is called **Store-and-Forward Packet Switching.**

****

**2. Services Provided to the Transport Layer**

The network layer provides services to the transport layer at the network layer/transport layer interface. The service need to be carefully designed with the following goals in mind:

1. The services should be independent of the router technology.
2. Topology of network should be hidden.
3. The network addresses available to the transport layer should use a uniform, even across LANs and WANs.
4. Network layer designers have freedom in writing specifics of services to transport layer.

**3. Implementation of Connectionless Service**

* No connection setup.
* Message is broken into packets, called **Datagram** (in analogy with telegram).
* Each packet is individually routed.
* Routers decide line based on routing table.
* Packets may follow different paths.
* Not guaranteed to arrive in order.



**Example: Internet**

**4. Implementation of Connection-Oriented Service**

* Path from source to destination must be established before any data can be sent.
* Connection is called VC **(Virtual Circuit).**
* Avoid choosing new route for each packet.
* Same route used for all packets in connection.
* Each packet has ID for which VC it belongs to.



**Example: Telephone**

**5. Comparison of Virtual-Circuit and Datagram Subnets**

****

|  |  |
| --- | --- |
| **Connectionless service** | **Connection oriented service** |
| In connectionless communication there is no need to establish connection between source and destination. | But in connection-oriented communication connection must established before data transfer. |
| Connectionless communication is not reliable. | Connection-oriented communication is more reliable. |
| In connectionless communication information cannot be resend because the destination does not inform the source about data is received or not. | In connection-oriented communication information can be resend if there is an error in receiver side(Missing data, corrupt data etc.) |
| In connectionless communication requires far less overhead. | In connection-oriented communication have higher overhead. |
| Not Guarantees a delivery | Guarantees a delivery |
| Ex: UDP (User Datagram Protocol) (Datagram) | Ex: TCP(Transmission Control Protocol)—(Virtual Circuits) |

**2. Routing algorithms**

As packets are transported from source nodes to destination nodes, they often require multiple hops to make the journey. Routing is the process of deciding what path a packet will take to reach its destination.

**Routing involves the design of:**

* + An algorithm to compute routes.
	+ Data structures to help choose routes.

**Routing can be two types:**

**Direct Routing**

* When destination is on the same network.
* Find the MAC address.
* Encapsulate the datagram in MAC frame.
* Send the frame to destination.

**Indirect Routing**

 Find out which is the next host to send the datagram.

**Routing Algorithms:** Is a part of Network layer software. Decides which output link an incoming packet should be transmitted on. There are two types of algorithms

1. **Static Routing (also called Non-Adaptive Algorithms)**

The routes are decided or computed in advance or off-line.

**Ex:** Shortest path algorithm

1. **Dynamic Routing (Also called Adaptive Algorithms)**

Change the routing decisions based on changes in the topology or traffic. Get the routing information from adjacent routers, or when they change routes.

**Ex:** Distance vector, Link state algorithms

**The Optimality principle:**

The set of optimal routes from all sources to a given destination form a tree rooted at the destination such as tree is called **sink tree.** The goal of the routing algorithms is to discover and use the sink trees for all routers.

**Sink Tree based on No. of Hops**



**1. Shortest Path Routing**

* It is simple and easy to understand.
* The idea is to build a graph of the subnet, with each node of the graph representing a router and each arc of the graph representing a communication line (often called a link).
* To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph.
* One way of measuring path length is the **number of hops**.
* Using this metric, the paths *ABC* and *ABE* in Fig. 5-7 are equally long.
* Another metric is the **geographic distance** in kilometers, in which case *ABC* is clearly much longer than *ABE.*

***Figure 5-7. The first five steps used in computing the shortest path from* A *to* D*. The arrows indicate the working node.***

* In the general case, the labels on the arcs could be computed as a function of the **distance**, **bandwidth**, **average traffic**, **communication cost**, **mean queue length**, **measured delay**, and other factors.
* By changing the weighting function, the algorithm would then compute the ''shortest'' path measured according to any one of a number of criteria or to a combination of criteria.
* Each node is labeled (in parentheses) with its distance from the source node along the best known path. Initially, no paths are known, so all nodes are labeled with infinity.
* As the algorithm proceeds and paths are found, the labels may change, reflecting better paths. A label may be either tentative or permanent. Initially, all labels are tentative.
* When it is discovered that a label represents the shortest possible path from the source to that node, it is made permanent and never changed thereafter.
* We want to find the shortest path from *A* to *D*. We start out by marking node *A* as permanent, indicated by a filled-in circle.
* Then we examine, in turn, each of the nodes adjacent to *A* (the working node), relabeling each one with the distance to *A*.
* Whenever a node is relabeled, we also label it with the node from which the probe was made so that we can reconstruct the final path later.
* Having examined each of the nodes adjacent to *A*, we examine all the tentatively labeled nodes in the whole graph and make the one with the smallest label permanent, as shown in Fig. 5-7(b). This one becomes the new working node.
* We now start at *B* and examine all nodes adjacent to it. If the sum of the label on *B* and the distance from *B* to the node being considered is less than the label on that node, we have a shorter path, so the node is relabeled. After all the nodes adjacent to the working node have been inspected and the tentative labels changed if possible, the entire graph is searched for the tentatively-labeled node with the smallest value. This node is made permanent and becomes the working node for the next round. Figure 5-7 shows the first five steps of the algorithm.
* The only difference between the program and the algorithm described above is that in Fig. 5-8, we compute the shortest path starting at the terminal node, *t*, rather than at the source node, *s*.
* Since the shortest path from *t* to *s* in an undirected graph is the same as the shortest path from *s* to *t*, it does not matter at which end we begin (unless there are several shortest paths, in which case reversing the search might discover a different one).
* The reason for searching backward is that each node is labeled with its predecessor rather than its successor. When the final path is copied into the output variable, *path*, the path is thus reversed. By reversing the search, the two effects cancel, and the answer is produced in the correct order.

**2 Flooding**

* Another static algorithm is **flooding**, in which every incoming packet is sent out on every outgoing line except the one it arrived on. Flooding obviously generates vast numbers of duplicate packets, in fact, an infinite number unless some measures are taken to damp the process.
* One such measure is to have a hop counter contained in the header of each packet, which is decremented at each hop, with the packet being discarded when the counter reaches zero. Ideally, the hop counter should be initialized to the length of the path from source to destination. If the sender does not know how long the path is, it can initialize the counter to the worst case, namely, the full diameter of the subnet.
* An alternative technique for damming the flood is to keep track of which packets have been flooded, to avoid sending them out a second time. Achieve this goal is to have the source router put a sequence number in each packet it receives from its hosts.
* Each router then needs a list per source router telling which sequence numbers originating at that source have already been seen. If an incoming packet is on the list, it is not flooded.
* To prevent the list from growing without bound, each list should be augmented by a counter, *k*, meaning that all sequence numbers through *k* have been seen. When a packet comes in, it is easy to check if the packet is a duplicate; if so, it is discarded. Furthermore, the full list below *k* is not needed, since *k* effectively summarizes it.
* A variation of flooding that is slightly more practical is **selective flooding**. In this algorithm the routers do not send every incoming packet out on every line, only on those lines that are going approximately in the right direction.
* There is usually little point in sending a westbound packet on an eastbound line unless the topology is extremely peculiar and the router is sure of this fact.
* Flooding is not practical in most applications, but it does have some uses. For example, in **military applications**, where large numbers of routers may be blown to bits at any instant, the tremendous robustness of flooding is highly desirable.
* In **distributed database applications**, it is sometimes necessary to update all the databases concurrently, in which case flooding can be useful.
* In **wireless networks**, all messages transmitted by a station can be received by all other stations within its radio range, which is, in fact, flooding, and some algorithms utilize this property.
* A fourth possible use of flooding is as a metric against which other routing algorithms can be compared. Flooding always chooses the shortest path because it chooses every possible path in parallel. Consequently, no other algorithm can produce a shorter delay (if we ignore the overhead generated by the flooding process itself).

**3. Distance Vector Routing**

* Modern computer networks generally use dynamic routing algorithms rather than the static ones described above because static algorithms do not take the current network load into account.
* Two dynamic algorithms in particular, distance vector routing and link state routing, are the most popular. **Distance vector routing** algorithms operate by having each router maintain a table (i.e, a vector) giving the best known distance to each destination and which line to use to get there. These tables are updated by exchanging information with the neighbors.
* The distance vector routing algorithm is sometimes called by other names, most commonly the distributed **Bellman-Ford** routing algorithm and the **Ford-Fulkerson** algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962). It was the original ARPANET routing algorithm and was also used in the Internet under the name RIP.
* In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in the subnet. This entry contains two parts: the preferred outgoing line to use for that destination and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.
* The router is assumed to know the ''distance'' to each of its neighbors. If the metric is hops, the distance is just one hop. If the metric is queue length, the router simply examines each queue. If the metric is delay, the router can measure it directly with special ECHO packets that the receiver just timestamps and sends back as fast as it can.
* As an example, assume that delay is used as a metric and that the router knows the delay to each of its neighbors. Once every *T* msec each router sends to each neighbor a list of its estimated delays to each destination. It also receives a similar list from each neighbor. Imagine that one of these tables has just come in from neighbor *X*, with *Xi* being *X*'s estimate of how long it takes to get to router *i*. If the router knows that the delay to *X* is *m* msec, it also knows that it can reach router *i* via *X* in *Xi* + *m* msec. By performing this calculation for each neighbor, a router can find out which estimate seems the best and use that estimate and the corresponding line in its new routing table. Note that the old routing table is not used in the calculation.
* This updating process is illustrated in Fig. 5-9. Part (a) shows a subnet. The first four columns of part (b) show the delay vectors received from the neighbors of router *J*. *A* claims to have a 12-msec delay to *B*, a 25-msec delay to *C*, a 40-msec delay to *D*, etc. Suppose that *J* has measured or estimated its delay to its neighbors, *A*, *I, H*, and *K* as 8, 10, 12, and 6 msec, respectively.

**Figure 5-9. (a) A subnet. (b) Input from A, I, H, K, and the new routing table for J.**



* Consider how *J* computes its new route to router *G*. It knows that it can get to *A* in 8 msec, and *A*claims to be able to get to *G* in 18 msec, so *J* knows it can count on a delay of 26 msec to *G* if it forwards packets bound for *G* to *A*. Similarly, it computes the delay to *G* via *I*, *H*, and *K* as 41 (31 + 10), 18 (6 + 12), and 37 (31 + 6) msec, respectively. The best of these values is 18, so it makes an entry in its routing table that the delay to *G* is 18 msec andthat the route to use is via *H*. The same calculation is performed for all the other destinations, with the new routing table shown in the last column of the figure.

**The Count-to-Infinity Problem**

* Distance vector routing works in theory but has a serious drawback in practice: although it converges to the correct answer, it may do so slowly. In particular, it reacts rapidly to good news, but leisurely to bad news. Consider a router whose best route to destination *X* is large. If on the next exchange neighbor *A* suddenly reports a short delay to *X*, the router just switches over to using the line to *A* to send traffic to *X*. In one vector exchange, the good news is processed.
* To see how fast good news propagates, consider the five-node (linear) subnet of Fig. 5-10, where the delay metric is the number of hops. Suppose *A* is down initially and all the other routers know this. In other words, they have all recorded the delay to *A* as infinity.
* ***Figure 5-10. The count-to-infinity problem.***



* When *A* comes up, the other routers learn about it via the vector exchanges. For simplicity we will assume that there is a gigantic gong somewhere that is struck periodically to initiate a vector exchange at all routers simultaneously. At the time of the first exchange, *B* learns that its left neighbor has zero delay to *A*. *B* now makes an entry in its routing table that *A* is one hop away to the left. All the other routers still think that *A* is down. At this point, the routing table entries for *A* are as shown in the second row of Fig. 5-10(a). On the next exchange, *C* learns that *B* has a path of length 1 to *A*, so it updates its routing table to indicate a path of length 2, but *D* and *E* do not hear the good news until later. Clearly, the good news is spreading at the rate of one hop per exchange. In a subnet whose longest path is of length *N* hops, within *N* exchanges everyone will know about newly-revived lines and routers.
* Now let us consider the situation of Fig. 5-10(b), in which all the lines and routers are initially up. Routers *B*, *C*, *D*, and *E*have distances to *A* of 1, 2, 3, and 4, respectively. Suddenly *A* goes down, or alternatively, the line between *A* and *B* is cut, which is effectively the same thing from *B*'s point of view.
* At the first packet exchange, *B* does not hear anything from *A*. Fortunately, *C* says: Do not worry; I have a path to *A* of length 2. Little does *B* know that *C*'s path runs through *B* itself. For all *B* knows, *C* might have ten lines all with separate paths to *A* of length 2. As a result, *B* thinks it can reach *A* via *C*, with a path length of 3. *D* and *E* do not update their entries for *A* on the first exchange.
* On the second exchange, *C* notices that each of its neighbors claims to have a path to *A* of length 3. It picks one of the them at random and makes its new distance to *A* 4, as shown in the third row of Fig. 5-10(b). Subsequent exchanges produce the history shown in the rest of Fig. 5-10(b).
* From this figure, it should be clear why bad news travels slowly: no router ever has a value more than one higher than the minimum of all its neighbors. Gradually, all routers work their way up to infinity, but the number of exchanges required depends on the numerical value used for infinity. For this reason, it is wise to set infinity to the longest path plus 1. If the metric is time delay, there is no well-defined upper bound, so a high value is needed to prevent a path with a long delay from being treated as down. Not entirely surprisingly, this problem is known as the **count-to-infinity** problem. There have been a few attempts to solve it (such as split horizon with poisoned reverse in RFC 1058), but none of these work well in general. The core of the problem is that when *X* tells *Y* that it has a path somewhere, *Y* has no way of knowing whether it itself is on the path.

**4. Link State Routing**

* Distance vector routing was used in the ARPANET until 1979, when it was replaced by link state routing. Two primary problems caused its demise. First, since the delay metric was queue length, it did not take line bandwidth into account when choosing routes. Initially, all the lines were 56 kbps, so line bandwidth was not an issue, but after some lines had been upgraded to 230 kbps and others to 1.544 Mbps, not taking bandwidth into account was a major problem.
* Of course, it would have been possible to change the delay metric to factor in line bandwidth, but a second problem also existed, namely, the algorithm often took too long to converge (the count-to-infinity problem). For these reasons, it was replaced by an entirely new algorithm, now called **link state routing**. Variants of link state routing are now widely used.
* The idea behind link state routing is simple and can be stated as five parts. Each router must do the following:
1. Discover its neighbors and learn their network addresses.
2. Measure the delay or cost to each of its neighbors.
3. Construct a packet telling all it has just learned.
4. Send this packet to all other routers.
5. Compute the shortest path to every other router.

***Learning about the Neighbors***

* When a router is booted, its first task is to learn who its neighbors are. It accomplishes this goal by sending a special HELLO packet on each point-to-point line. The router on the other end is expected to send back a reply telling who it is. These names must be globally unique because when a distant router later hears that three routers are all connected to *F*, it is essential that it can determine whether all three mean the same *F*.
* When two or more routers are connected by a LAN, the situation is slightly more complicated. Fig. 5-11(a) illustrates a LAN to which three routers, *A*, *C*, and *F*, are directly connected. Each of these routers is connected to one or more additional routers, as shown.

***Figure 5-11. (a) Nine routers and a LAN. (b) A graph model of (a).***



* One way to model the LAN is to consider it as a node itself, as shown in Fig. 5-11(b). Here we have introduced a new, artificial node, *N*, to which *A*, *C*, and *F* are connected. The fact that it is possible to go from *A* to *C* on the LAN is represented by the path *ANC* here.

***Measuring Line Cost***

* The link state routing algorithm requires each router to know, or at least have a reasonable estimate of, the delay to each of its neighbors. The most direct way to determine this delay is to send over the line a special ECHO packet that the other side is required to send back immediately. By measuring the round-trip time and dividing it by two, the sending router can get a reasonable estimate of the delay. For even better results, the test can be conducted several times, and the average used. Of course, this method implicitly assumes the delays are symmetric, which may not always be the case.
* An interesting issue is whether to take the load into account when measuring the delay. To factor the load in, the round-trip timer must be started when the ECHO packet is queued. To ignore the load, the timer should be started when the ECHO packet reaches the front of the queue.
* Arguments can be made both ways. Including traffic-induced delays in the measurements means that when a router has a choice between two lines with the same bandwidth, one of which is heavily loaded all the time and one of which is not, the router will regard the route over the unloaded line as a shorter path. This choice will result in better performance.
* Unfortunately, there is also an argument against including the load in the delay calculation. Consider the subnet of Fig. 5-12, which is divided into two parts, East and West, connected by two lines, *CF* and *EI*.

***Figure 5-12. A subnet in which the East and West parts are connected by two lines.***



* Suppose that most of the traffic between East and West is using line *CF*, and as a result, this line is heavily loaded with long delays. Including queueing delay in the shortest path calculation will make *EI* more attractive. After the new routing tables have been installed, most of the East-West traffic will now go over *EI*, overloading this line. Consequently, in the next update, *CF* will appear to be the shortest path. As a result, the routing tables may oscillate wildly, leading to erratic routing and many potential problems. If load is ignored and only bandwidth is considered, this problem does not occur. Alternatively, the load can be spread over both lines, but this solution does not fully utilize the best path. Nevertheless, to avoid oscillations in the choice of best path, it may be wise to distribute the load over multiple lines, with some known fraction going over each line.

***Building Link State Packets***

* Once the information needed for the exchange has been collected, the next step is for each router to build a packet containing all the data. The packet starts with the identity of the sender, followed by a sequence number and age (to be described later), and a list of neighbors. For each neighbor, the delay to that neighbor is given. An example subnet is given in Fig. 5-13(a) with delays shown as labels on the lines. The corresponding link state packets for all six routers are shown in Fig. 5-13(b).

***Figure 5-13. (a) A subnet. (b) The link state packets for this subnet.***



* Building the link state packets is easy. The hard part is determining when to build them. One possibility is to build them periodically, that is, at regular intervals. Another possibility is to build them when some significant event occurs, such as a line or neighbor going down or coming back up again or changing its properties appreciably.

**5. Hierarchical Routing**

* As networks grow in size, the router routing tables grow proportionally. Not only is router memory consumed by ever-increasing tables, but more CPU time is needed to scan them and more bandwidth is needed to send status reports about them. At a certain point the network may grow to the point where it is no longer feasible for every router to have an entry for every other router, so the routing will have to be done hierarchically, as it is in the telephone network.
* When hierarchical routing is used, the routers are divided into what we will call **regions**, with each router knowing all the details about how to route packets to destinations within its own region, but knowing nothing about the internal structure of other regions. When different networks are interconnected, it is natural to regard each one as a separate region in order to free the routers in one network from having to know the topological structure of the other ones.
* For huge networks, a two-level hierarchy may be insufficient; it may be necessary to group the regions into clusters, the clusters into zones, the zones into groups, and so on, until we run out of names for aggregations.
* Figure 5-15 gives a quantitative example of routing in a two-level hierarchy with five regions. The full routing table for router 1*A* has 17 entries, as shown in Fig. 5-15(b). When routing is done hierarchically, as in Fig. 5-15(c), there are entries for all the local routers as before, but all other regions have been condensed into a single router, so all traffic for region 2 goes via the 1*B* -2*A* line, but the rest of the remote traffic goes via the 1*C* -3*B* line. Hierarchical routing has reduced the table from 17 to 7 entries. As the ratio of the number of regions to the number of routers per region grows, the savings in table space increase.

**Figure 5-15. Hierarchical routing.**

* Unfortunately, these gains in space are not free. There is a penalty to be paid, and this penalty is in the form of increased path length. For example, the best route from 1*A* to 5*C* is via region 2, but with hierarchical routing all traffic to region 5 goes via region 3, because that is better for most destinations in region 5.
* When a single network becomes very large, an interesting question is: How many levels should the hierarchy have? For example, consider a subnet with 720 routers. If there is no hierarchy, each router needs 720 routing table entries. If the subnet is partitioned into 24 regions of 30 routers each, each router needs 30 local entries plus 23 remote entries for a total of 53 entries. If a three-level hierarchy is chosen, with eight clusters, each containing 9 regions of 10 routers, each router needs 10 entries for local routers, 8 entries for routing to other regions within its own cluster, and 7 entries for distant clusters, for a total of 25 entries. Kamoun and Kleinrock (1979) discovered that the optimal number of levels for an *N* router subnet is ln*N*, requiring a total of *e* ln*N* entries per router. They have also shown that the increase in effective mean path length caused by hierarchical routing is sufficiently small that it is usually acceptable.

**3. Transport layer design issues**

**Introduction:**

* The transport layer is the fourth layer from the bottom in the OSI reference model.
* It is responsible for message delivery from process running in source computer to the process running in the destination computer.
* Transport layer does not perform any function in the intermediate nodes.
* It is active only in the end systems.
* Data Link Layer is responsible for delivery of frames between two neighboring nodes over a link.
	+ This is called ***node-to-node delivery***.
* Network Layer is responsible for delivery of datagram’s between two hosts.
	+ This is called ***host-to-host delivery***.
* Transport Layer is responsible for delivery of entire message from one process running on source to another process running on destination. This is called ***process-to process delivery.***

**Design Issues of Transport layer:**

* The transport layer delivers the message from one process to another process running on two different hosts.
* Thus, it has to perform number of functions to ensure the accurate delivery of message.
* The various functions of transport layer are:

1. Establishing, Maintaining & Releasing Connection

2. Addressing

3. Data Transfer

4. Flow Control

5. Error Control

6. Congestion Control

**1. Establishing, Maintaining & Releasing Connection**

* The transport layer establishes, maintains & releases end-to-end transport connection on the request of upper layers.
* Establishing a connection involves allocation of buffers for storing user data, synchronizing the sequence numbers of packets etc.
* A connection is released at the request of upper layer

**2. Addressing**

* In order to deliver the message from one process to another, an addressing scheme is required.
* Several processes may be running on a system at a time.
* In order to identify the correct process out of the various running processes, transport layer uses an addressing scheme called ***port number***.
* Each process has a specific port number.

**3. Data Transfer**

* Transport layer breaks user data into smaller units and attaches a transport layer header to each unit forming a TPDU (Transport Layer Data Unit).
* The TPDU is handed over to the network layer for its delivery to destination.
* The TPDU header contains port number, sequence number, acknowledgement number, checksum and other fields.

**4. Flow Control**

* Like data link layer, transport layer also performs flow control.
* However, flow control at transport layer is performed end-to-end rather than node-to-node.
* Transport Layer uses a sliding window protocol to perform flow control.

**5. Error Control**

* Transport layer also provides end-to-end error control facility.
* Transport layer deals with several different types of errors:
	+ - Error due to damaged bits.
		- Error due to non delivery of TPDUs.
		- Error due to duplicate delivery of TPDUs.
* Error due to delivery of TPDU to a wrong destination.

**6. Congestion Control**

* Transport layer also handles congestion in the networks.
* Several different congestion control algorithms are used to avoid congestion.

**Transport layer services:**

Transport layer protocols can provide two types of services:

1. Connection Oriented Service

2. Connectionless Service

**1. Connection Oriented Service**

* In connection oriented service, a connection is first established between sender and the receiver.
* Then, transfer of user data takes place.
* At the end, connection is released.
* The connection oriented service is generally reliable.
* Transport layer protocols that provide connection oriented service are **TCP** and **SCTP** (Stream Control Transmission Protocol).

**2. Connectionless Service**

* In this service, the packets are sent from sender to receiver without the establishment of connection.
* In such service, packets are not numbered.
* The packets may be lost, corrupted, delayed or disordered.
* Connectionless service is unreliable.
* Transport layer protocol that provides this service is **UDP**.

**Elements of transport protocol**

**1. Addressing**

* In order to deliver data from one process to another, address is required.
* In order to deliver data from one node to another, MAC address is required.
* Such an address is implemented at Data Link Layer and is called **Physical Addressing**.
	+ In order to deliver data from one network to another, IP address is required.
	+ Such an address is implemented at Network Layer and is called **Logical Addressing**.
	+ Similarly, in order to deliver data from a process running on source to process running on destination, transport layer defines the **Service Point Address** or **Port Numbers**.

**2. Port Numbers**

* Each communicating process is assigned a specific port number.
* In order to select among multiple processes running on a destination host, a port number is required.
* The port numbers are 16-bit integers between 0 and 65,535.
* Port numbers are assigned by Internet Assigned Number Authority (IANA).
* IANA has divided the port numbers in three categories:

**Well Known Ports:** The ports ranging from 0 to 1023. For Ex: HTTP: 80, SMTP: 25, FTP: 21.

**Registered Ports:** The ports ranging from 1024 to 49,151.These are not controlled by IANA.

**Dynamic Ports:** The ports ranging from 49,152 to 65,535.These can be used by any process.

**3. Socket Address**

* Socket address is a combination of IP address and port number.
* In order to provide communication between two different processes on different networks, both IP address and port number, i.e. socket address is required.

**4. Multiplexing& Demultiplexing**

* A network connection can be shared by various applications running on a system.
* There may be several running processes that want to send data and only one transport layer connection available, and then transport layer protocols may perform multiplexing.
* The protocols accept the messages from different processes having their respective port numbers, and add headers to them.
* The transport layer at the receiver end performs demultiplexing to separate the messages for different processes.
* After checking for errors, the headers of messages are dropped and each message is handed over to the respective processes based on their port numbers.

**5. Connection Establishment:**

* Before communicating, the source device must first determine the availability of the other to exchange data.
* Path must be found through the network by which the data can be sent.
* This is called Connection Establishment.

**6. Connection Release**

* Once all of the data has been transferred, the connection must be released.
* It also requires a **Three-Way Handshaking** mechanism:
* The source sends a disconnect request packet to the destination.
* The destination returns a confirmation packet back to the source.
* The source returns a packet acknowledging the confirmation.

#### 4. Design Issues of Session Layer

The session layer is level five of the seven level OSI model. It responds to service requests from the presentation layer and issues service requests to the transport layer.

The session layer provides the mechanism for opening, closing and managing a session between end-user application processes, i.e. a semi-permanent dialogue. Communication sessions consist of requests and responses that occur between applications.

Session layers are commonly used in application environments that make use of remote procedure calls (RPCs).

An example of a session layer protocol X.225 or ISO 8327.

In case of a connection loss this protocol may try to recover the connection. If a connection is not used for a long period, the session layer protocol may close it and re-open it.

It provides for either full duplex or half-duplex operation and provides synchronization points in the stream of exchanged messages.

List of Session layer services

* Authentication
* Permissions
* Session restoration (checkpoint and recovery)

**Authentication:**

Authentication is the act of establishing or confirming something (or someone) as authentic, that is that claims made by or about the thing are true. This might involve confirming the identity of a person, the origins of an artifact, or assuring that a computer program is a trusted one.

**Permissions or Access control**

One familiar use of authentication and authorization is access control. A computer system supposed to be used only by those authorized must attempt to detect and exclude the unauthorized. Access to it is therefore usually controlled by insisting on an authentication procedure to establish with some established degree of confidence the identity of the user, thence granting those privileges as may be authorized to that identity.

In some cases, ease of access is balanced against the strictness of access checks. For example, the credit card network does not require a personal identification number, and small transactions usually do not even require a signature. The security of the system is maintained by limiting distribution of credit card numbers, and by the threat of punishment for fraud.

**Checkpoints**

Session layer is responsible for creating several checkpoints; checkpoints are also treated as recovery points i.e. in case of failure the system rollback to its previous checkpoint configuration or action.

**5. Remote Procedure Call (RPC)**

It is a protocol that one program can use to request a service from a program located in another **computer** in a **network** without having to understand **network** details. (A **procedure call** is also sometimes known as a function **call** or a subroutine **call**.) **RPC** uses the client/server model.

**6 Congestion Control Algorithms**

* When too many packets are present in (a part of) the subnet, performance degrades. This situation is called **congestion**. Figure 5-25 depicts the symptom.
* When the number of packets dumped into the subnet by the hosts is within its carrying capacity, they are all delivered (except for a few that are afflicted with transmission errors) and the number delivered is proportional to the number sent.
* However, as traffic increases too far, the routers are no longer able to cope and they begin losing packets. This tends to make matters worse. At very high trafffic, performance collapses completely and almost no packets are delivered.

 ***Figure 5-25. When too much traffic is offered, congestion sets in and performance degrades sharply.***



* Congestion can be brought on by several factors. If all of a sudden, streams of packets begin arriving on three or four input lines and all need the same output line, a queue will build up. If there is **insufficient memory** to hold all of them, packets will be lost.
* Adding more memory may help up to a point, but Nagle (1987) discovered that if routers have an infinite amount of memory, congestion gets worse, not better, because by the time packets get to the front of the queue, they have already timed out (repeatedly) and duplicates have been sent. All these packets will be dutifully forwarded to the next router, increasing the load all the way to the destination.
* **Slow processors** can also cause congestion. If the routers' CPUs are slow at performing the bookkeeping tasks required of them (queueing buffers, updating tables, etc.), queues can build up, even though there is excess line capacity.
* Similarly, **low-bandwidth** lines can also cause congestion. Upgrading the lines but not changing the processors, or vice versa, often helps a little, but frequently just shifts the bottleneck. Also, upgrading part, but not all, of the system, often just moves the bottleneck somewhere else. The real problem is frequently a mismatch between parts of the system. This problem will persist until all the components are in balance.

**5.3.1 General Principles of Congestion Control**

* This leads to dividing all solutions into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made.
* Tools for doing open-loop control include deciding when to accept new traffic, deciding when to discard packets and which ones, and making scheduling decisions at various points in the network. All of these have in common the fact that they make decisions without regard to the current state of the network.
* In contrast, closed loop solutions are based on the concept of a feedback loop. This approach has three parts when applied to congestion control:
1. Monitor the system to detect when and where congestion occurs.
2. Pass this information to places where action can be taken.
3. Adjust system operation to correct the problem.
* A variety of metrics can be used to monitor the subnet for congestion. Chief among these are the percentage of all packets discarded for lack of buffer space, the average queue lengths, the number of packets that time out and are retransmitted, the average packet delay, and the standard deviation of packet delay. In all cases, rising numbers indicate growing congestion.
* The second step in the feedback loop is to transfer the information about the congestion from the point where it is detected to the point where something can be done about it. The obvious way is for the router detecting the congestion to send a packet to the traffic source or sources, announcing the problem. Of course, these extra packets increase the load at precisely the moment that more load is not needed, namely, when the subnet is congested.
* However, other possibilities also exist. For example, a bit or field can be reserved in every packet for routers to fill in whenever congestion gets above some threshold level. When a router detects this congested state, it fills in the field in all outgoing packets, to warn the neighbors.
* Still another approach is to have hosts or routers periodically send probe packets out to explicitly ask about congestion. This information can then be used to route traffic around problem areas. Some radio stations have helicopters flying around their cities to report on road congestion to make it possible for their mobile listeners to route their packets (cars) around hot spots.
* In all feedback schemes, the hope is that knowledge of congestion will cause the hosts to take appropriate action to reduce the congestion. For a scheme to work correctly, the time scale must be adjusted carefully. If every time two packets arrive in a row, a router yells STOP and every time a router is idle for 20 μsec, it yells GO, the system will oscillate wildly and never converge. On the other hand, if it waits 30 minutes to make sure before saying anything, the congestion control mechanism will react too sluggishly to be of any real use. To work well, some kind of averaging is needed, but getting the time constant right is a nontrivial matter.
* They begin by dividing all algorithms into open loop or closed loop. They further divide the open loop algorithms into ones that act at the source versus ones that act at the destination.
* The closed loop algorithms are also divided into two subcategories: explicit feedback versus implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithms, the source deduces the existence of congestion by making local observations, such as the time needed for acknowledgements to come back.
* The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle.
* Two solutions come to mind: increase the resources or decrease the load. For example, the subnet may start using dial-up telephone lines to temporarily increase the bandwidth between certain points. On satellite systems, increasing transmission power often gives higher bandwidth. Splitting traffic over multiple routes instead of always using the best one may also effectively increase the bandwidth. Finally, spare routers that are normally used only as backups (to make the system fault tolerant) can be put on-line to give more capacity when serious congestion appears. However, sometimes it is not possible to increase the capacity, or it has already been increased to the limit.

**5.3.2 Congestion Prevention Policies**

* Let us begin our study of methods to control congestion by looking at open loop systems. These systems are designed to minimize congestion in the first place, rather than letting it happen and reacting after the fact. They try to achieve their goal by using appropriate policies at various levels. In Fig. 5-26 we see different data link, network, and transport policies that can affect congestion (Jain, 1990).

***Figure 5-26. Policies that affect congestion.***

* Let us start at the data link layer and work our way upward. The retransmission policy is concerned with how fast a sender times out and what it transmits upon timeout. A jumpy sender that times out quickly and retransmits all outstanding packets using go back n will put a heavier load on the system than will a leisurely sender that uses selective repeat. Closely related to this is the buffering policy. If receivers routinely discard all out-of-order packets, these packets will have to be transmitted again later, creating extra load. With respect to congestion control, selective repeat is clearly better than go back n.
* Acknowledgement policy also affects congestion. If each packet is acknowledged immediately, the acknowledgement packets generate extra traffic. However, if acknowledgements are saved up to piggyback onto reverse traffic, extra timeouts and retransmissions may result. A tight flow control scheme (e.g., a small window) reduces the data rate and thus helps fight congestion.
* At the network layer, the choice between using virtual circuits and using datagram’s affects congestion since many congestion control algorithms work only with virtual-circuit subnets. Packet queuing and service policy relates to whether routers have one queue per input line, one queue per output line, or both. It also relates to the order in which packets are processed (e.g., round robin or priority based). Discard policy is the rule telling which packet is dropped when there is no space. A good policy can help alleviate congestion and a bad one can make it worse.
* A good routing algorithm can help avoid congestion by spreading the traffic over all the lines, whereas a bad one can send too much traffic over already congested lines. Finally, packet lifetime management deals with how long a packet may live before being discarded. If it is too long, lost packets may clog up the works for a long time, but if it is too short, packets may sometimes time out before reaching their destination, thus inducing retransmissions.
* In the transport layer, the same issues occur as in the data link layer, but in addition, determining the timeout interval is harder because the transit time across the network is less predictable than the transit time over a wire between two routers. If the timeout interval is too short, extra packets will be sent unnecessarily. If it is too long, congestion will be reduced but the response time will suffer whenever a packet is lost.

**5.3.3 Congestion Control in Virtual-Circuit Subnets**

* The congestion control methods described above are basically open loop: they try to prevent congestion from occurring in the first place, rather than dealing with it after the fact. In this section we will describe some approaches to dynamically controlling congestion in virtual-circuit subnets. In the next two, we will look at techniques that can be used in any subnet.
* One technique that is widely used to keep congestion that has already started from getting worse is **admission control**.
* The idea is simple: once congestion has been signaled, no more virtual circuits are set up until the problem has gone away. Thus, attempts to set up new transport layer connections fail. Letting more people in just makes matters worse. While this approach is crude, it is simple and easy to carry out. In the telephone system, when a switch gets overloaded, it also practices admission control by not giving dial tones.
* An alternative approach is to allow new virtual circuits but carefully route all new virtual circuits around problem areas. For example, consider the subnet of Fig. 5-27(a), in which two routers are congested, as indicated.

***Figure 5-27. (a) A congested subnet. (b) A redrawn subnet that eliminates the congestion. A virtual circuit from* A *to* B *is also shown.***



* Suppose that a host attached to router *A* wants to set up a connection to a host attached to router *B*. Normally, this connection would pass through one of the congested routers. To avoid this situation, we can redraw the subnet as shown in Fig. 5-27(b), omitting the congested routers and all of their lines.
* The dashed line shows a possible route for the virtual circuit that avoids the congested routers. Another strategy relating to virtual circuits is to negotiate an agreement between the host and subnet when a virtual circuit is set up.
* This agreement normally specifies the volume and shape of the traffic, quality of service required, and other parameters. To keep its part of the agreement, the subnet will typically reserve resources along the path when the circuit is set up. These resources can include table and buffer space in the routers and bandwidth on the lines. In this way, congestion is unlikely to occur on the new virtual circuits because all the necessary resources are guaranteed to be available.
* This kind of reservation can be done all the time as standard operating procedure or only when the subnet is congested. A disadvantage of doing it all the time is that it tends to waste resources. If six virtual circuits that might use 1 Mbps all pass through the same physical 6-Mbps line, the line has to be marked as full, even though it may rarely happen that all six virtual circuits are transmitting full blast at the same time. Consequently, the price of the congestion control is unused (i.e., wasted) bandwidth in the normal case.

**5.3.4 Congestion Control in Datagram Subnets**

***The Warning Bit***

* The old DECNET architecture signaled the warning state by setting a special bit in the packet's header. So does frame relay. When the packet arrived at its destination, the
* transport entity copied the bit into the next acknowledgement sent back to the source. The source then cut back on traffic.
* As long as the router was in the warning state, it continued to set the warning bit, which meant that the source continued to get acknowledgements with it set. The source monitored the fraction of acknowledgements with the bit set and adjusted its transmission rate accordingly. As long as the warning bits continued to flow in, the source continued to decrease its transmission rate. When they slowed to a trickle, it increased its transmission rate. Note that since every router along the path could set the warning bit, traffic increased only when no router was in trouble.

***Choke Packets***

* It uses a roundabout means to tell the source to slow down. Why not just tell it directly? In this approach, the router sends a **choke packet** back to the source host, giving it the destination found in the packet. The original packet is tagged (a header bit is turned on) so that it will not generate any more choke packets farther along the path and is then forwarded in the usual way.
* When the source host gets the choke packet, it is required to reduce the traffic sent to the specified destination by *X* percent. Since other packets aimed at the same destination are probably already under way and will generate yet more choke packets, the host should ignore choke packets referring to that destination for a fixed time interval. After that period has expired, the host listens for more choke packets for another interval. If one arrives, the line is still congested, so the host reduces the flow still more and begins ignoring choke packets again. If no choke packets arrive during the listening period, the host may increase the flow again. The feedback implicit in this protocol can help prevent congestion yet not throttle any flow unless trouble occurs.
* Hosts can reduce traffic by adjusting their policy parameters, for example, their window size. Typically, the first choke packet causes the data rate to be reduced to 0.50 of its previous rate, the next one causes a reduction to 0.25, and so on. Increases are done in smaller increments to prevent congestion from reoccurring quickly.
* Several variations on this congestion control algorithm have been proposed. For one, the routers can maintain several thresholds. Depending on which threshold has been crossed, the choke packet can contain a mild warning, a stern warning, or an ultimatum.
* Another variation is to use queue lengths or buffer utilization instead of line utilization as the trigger signal. The same exponential weighting can be used with this metric as with *u*, of course.

***Hop-by-Hop Choke Packets***

* At high speeds or over long distances, sending a choke packet to the source hosts does not work well because the reaction is so slow. Consider, for example, a host in San Francisco (router *A*in Fig. 5-28) that is sending traffic to a host in New York (router *D* in Fig. 5-28) at 155 Mbps. If the New York host begins to run out of buffers, it will take about 30 msec for a
* choke packet to get back to San Francisco to tell it to slow down. The choke packet propagation is shown as the second, third, and fourth steps in Fig. 5-28(a). In those 30 msec, another 4.6 megabits will have been sent. Even if the host in San Francisco completely shuts down immediately, the 4.6 megabits in the pipe will continue to pour in and have to be dealt with. Only in the seventh diagram in Fig. 5-28(a) will the New York router notice a slower flow.

***Figure 5-28. (a) A choke packet that affects only the source. (b) A choke packet that affects each hop it passes through. ***

* An alternative approach is to have the choke packet take effect at every hop it passes through, as shown in the sequence of Fig. 5-28(b). Here, as soon as the choke packet reaches *F*, *F* is required to reduce the flow to *D*. Doing so will require *F* to devote more buffers to the flow, since the source is still sending away at full blast, but it gives *D* immediate relief, like a headache remedy in a television commercial. In the next step, the choke packet reaches *E*, which tells *E* to reduce the flow to *F*. This action puts a greater demand on *E*'s buffers but gives *F* immediate relief. Finally, the choke packet reaches *A* and the flow genuinely slows down.
* The net effect of this hop-by-hop scheme is to provide quick relief at the point of congestion at the price of using up more buffers upstream. In this way, congestion can be nipped in the bud without losing any packets. The idea is discussed in detail and simulation results are given in (Mishra and Kanakia, 1992).

**5.3.5 Load Shedding**

* When none of the above methods make the congestion disappear, routers can bring out the heavy artillery: load shedding. **Load shedding** is a fancy way of saying that when routers are being inundated by packets that they cannot handle, they just throw them away. The term comes from the world of electrical power generation, where it refers to the practice of utilities intentionally blacking out certain areas to save the entire grid from collapsing on hot summer days when the demand for electricity greatly exceeds the supply.
* A router drowning in packets can just pick packets at random to drop, but usually it can do better than that. Which packet to discard may depend on the applications running. For file transfer, an old packet is worth more than a new one because dropping packet 6 and keeping packets 7 through 10 will cause a gap at the receiver that may force packets 6 through 10 to be retransmitted (if the receiver routinely discards out-of-order packets). In a 12-packet file, dropping 6 may require 7 through 12 to be retransmitted, whereas dropping 10 may require only 10 through 12 to be retransmitted. In contrast, for multimedia, a new packet is more important than an old one. The former policy (old is better than new) is often called **wine** and the latter (new is better than old) is often called **milk**.
* A step above this in intelligence requires cooperation from the senders. For many applications, some packets are more important than others. For example, certain algorithms for compressing video periodically transmit an entire frame and then send subsequent frames as differences from the last full frame. In this case, dropping a packet that is part of a difference is preferable to dropping one that is part of a full frame. As another example, consider transmitting a document containing ASCII text and pictures. Losing a line of pixels in some image is far less damaging than losing a line of readable text.
* To implement an intelligent discard policy, applications must mark their packets in priority classes to indicate how important they are. If they do this, then when packets have to be discarded, routers can first drop packets from the lowest class, then the next lowest class, and so on. Of course, unless there is some significant incentive to mark packets as anything other than VERY IMPORTANT— NEVER, EVER DISCARD, nobody will do it.
* The incentive might be in the form of money, with the low-priority packets being cheaper to send than the high-priority ones. Alternatively, senders might be allowed to send high-

priority packets under conditions of light load, but as the load increased they would be discarded, thus encouraging the users to stop sending them.

* Another option is to allow hosts to exceed the limits specified in the agreement negotiated when the virtual circuit was set up (e.g., use a higher bandwidth than allowed), but subject to the condition that all excess traffic be marked as low priority. Such a strategy is actually not a bad idea, because it makes more efficient use of idle resources, allowing hosts to use them as long as nobody else is interested, but without establishing a right to them when times get tough.

***Random Early Detection***

* It is well known that dealing with congestion after it is first detected is more effective than letting it gum up the works and then trying to deal with it. This observation leads to the idea of discarding packets before all the buffer space is really exhausted. A popular algorithm for doing this is called **RED** (**Random Early Detection**) (Floyd and Jacobson, 1993). In some transport protocols (including TCP), the response to lost packets is for the source to slow down. The reasoning behind this logic is that TCP was designed for wired networks and wired networks are very reliable, so lost packets are mostly due to buffer overruns rather than transmission errors. This fact can be exploited to help reduce congestion. By having routers drop packets before the situation has become hopeless (hence the ''early'' in the name), the idea is that there is time for action to be taken before it is too late. To determine when to start discarding, routers maintain a running average of their queue lengths. When the average queue length on some line exceeds a threshold, the line is said to be congested and action is taken.
* Since the router probably cannot tell which source is causing most of the trouble, picking a packet at random from the queue that triggered the action is probably as good as it can do.
* How should the router tell the source about the problem? One way is to send it a choke packet, as we have described. A problem with that approach is that it puts even more load on the already congested network. A different strategy is to just discard the selected packet and not report it. The source will eventually notice the lack of acknowledgement and take action. Since it knows that lost packets are generally caused by congestion and discards, it will respond by slowing down instead of trying harder. This implicit form of feedback only works when sources respond to lost packets by slowing down their transmission rate. In wireless networks, where most losses are due to noise on the air link, this approach cannot be used.

**5.3.6 Jitter Control**

* For applications such as audio and video streaming, it does not matter much if the packets take 20 msec or 30 msec to be delivered, as long as the transit time is constant. The variation (i.e., standard deviation) in the packet arrival times is called **jitter**. High jitter, for example, having some packets taking 20 msec and others taking 30 msec to arrive will give an uneven quality to the sound or movie. Jitter is illustrated in Fig. 5-29. In contrast, an
* agreement that 99 percent of the packets be delivered with a delay in the range of 24.5 msec to 25.5 msec might be acceptable.

***Figure 5-29. (a) High jitter. (b) Low jitter.***



* The range chosen must be feasible, of course. It must take into account the speed-of-light transit time and the minimum delay through the routers and perhaps leave a little slack for some inevitable delays.
* The jitter can be bounded by computing the expected transit time for each hop along the path. When a packet arrives at a router, the router checks to see how much the packet is behind or ahead of its schedule. This information is stored in the packet and updated at each hop. If the packet is ahead of schedule, it is held just long enough to get it back on schedule. If it is behind schedule, the router tries to get it out the door quickly.

In fact, the algorithm for determining which of several packets competing for an output line should go next can always choose the packet furthest behind in its schedule. In this way, packets that are ahead of schedule get slowed down and packets that are behind schedule get speeded up, in both cases reducing the amount of jitter.

* In some applications, such as video on demand, jitter can be eliminated by buffering at the receiver and then fetching data for display from the buffer instead of from the network in real time. However, for other applications, especially those that require real-time interaction between people such as Internet telephony and videoconferencing, the delay inherent in buffering is not acceptable.
* Congestion control is an active area of research. The state-of-the-art is summarized in (Gevros et al., 2001).