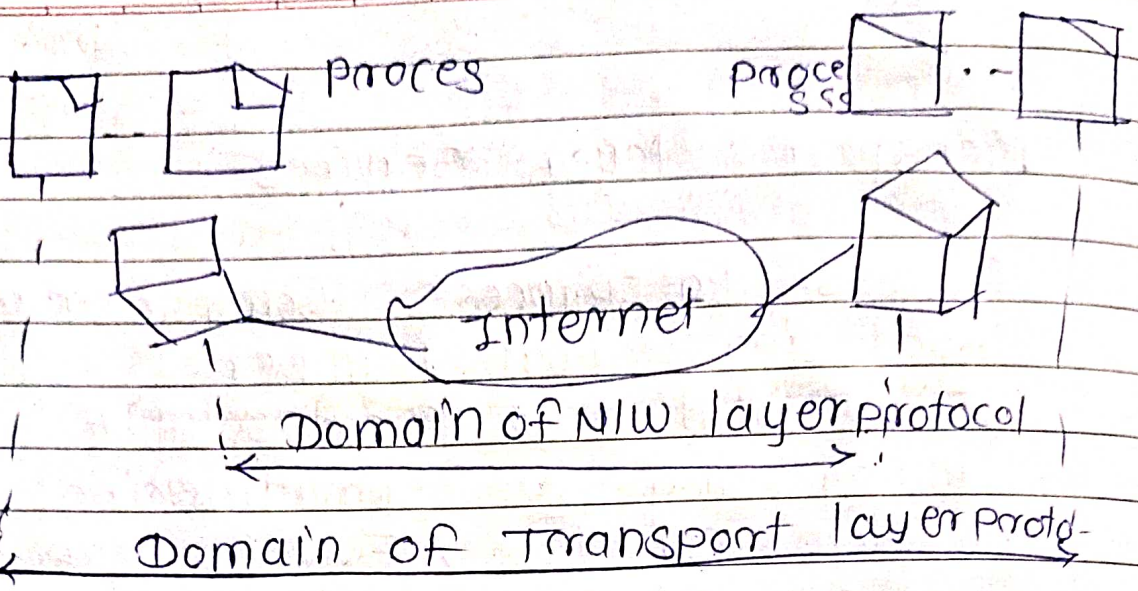
Page No.
Date

Transport layer services

Process to process Communication
Addressing : Port Number.
Encapsulation & Decapsulation
Multiplexing & Demultiplexing
flow control, Errol control
conjection control
Connectionless & connection - oriented services.

1) process to process Communication

1) The Transport layer protocol to provide process to process communication
2) A process is an application layer entity that uses the services of Transport layer.
3) A network layer is responsible for Communication at host to host
4) A network layer protocol can deliver the message only to the destination computer, Hence this process is incomplete delivery.
5) This is where transport layer protocol take over
6) Tcp is responsible for deliver message to correct process



Addressing: port Number:

1) The most common process client server paradigm.
2) A process on local host called client
3) A process on remote host called server Both process have the same name client & server.
4) A remote computer can run several server program at sametim & several local ~~number~~ computer Can run or more client program at same time.
5) for communication we must define.
   1) Local host
   2) local process
   3) Remote host
   4) Remote process

6) Local host & Remote host are defined using IP address. To define process we need Port address as a identi-

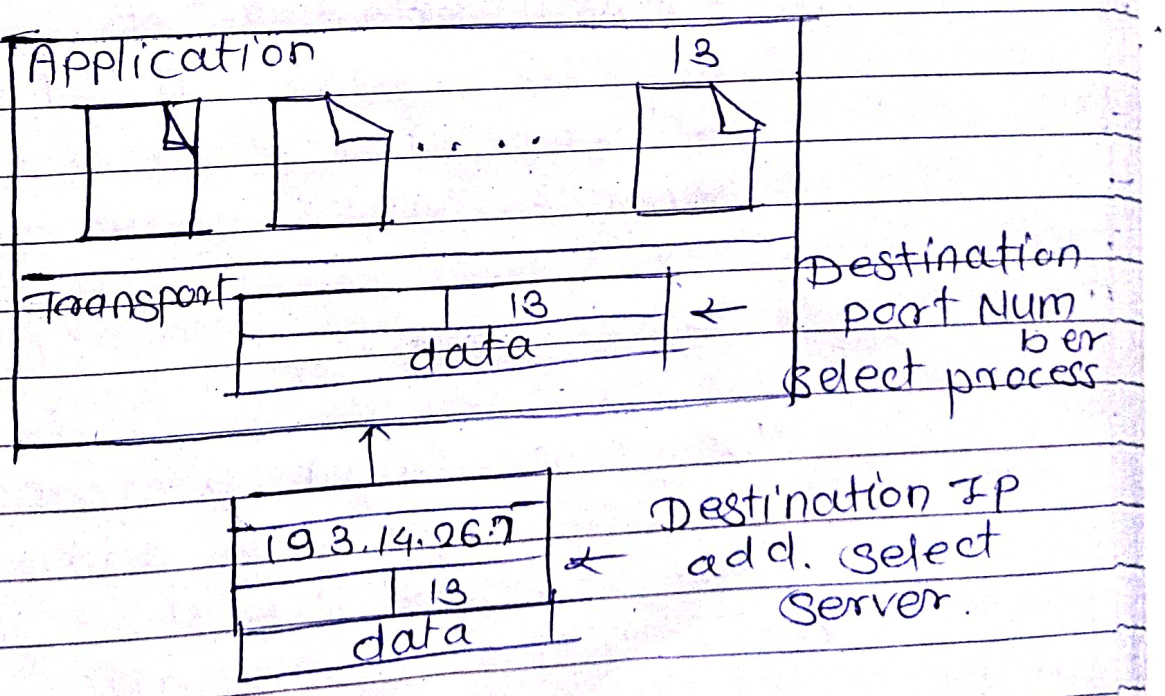
7) Tcp/IP protocol suite, the port Number are integer betn 0 & 65,535.

8) The client program define itself with Port Number called ephemeral port Number

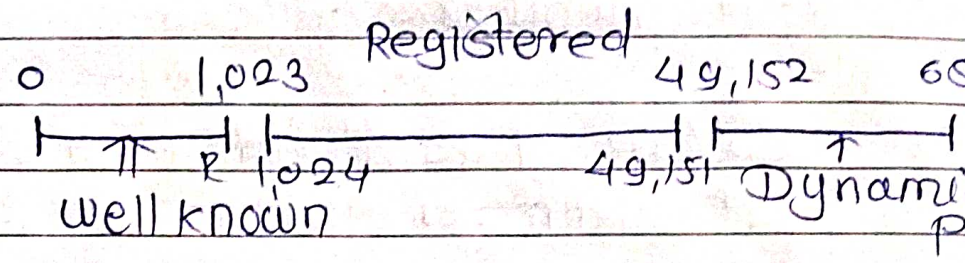
9) Tcp/IP has decided to use universal port Number for server are called well known port Number.

10) Every client process knows the well-known port Number of corresponding server process.

11) The destination IP address defin host among the diff host in world.
After the host been selected the port Number define one of Process on this particular host.



ICANN Ranges:
ICANN has divided the port Number into (3 ranges
i) well known ii) registered iii) dynami



well known port: The port ranging from 0 to 1,023 are assign & contr olled by ICANN

Registered port: The port ranging from 1,024 to 49,151 are not assigned or controlled by ICANN.

Dynamic port: The port rangi ng from ~~1,024 to 4~~ 49,152 to 65,535 are neither controlled or registered. It is Temporaryor Private port Number

Socket Address

1) Transport-layer protocol in TCP suit need both IP address & Port Numb er at each end.
2) The combination of IP add ress & port Number is called Socket address.

3) The client socket address define client process Uniquely & server socket address define server process Uniquely.

4) To use service ↑ of TCP layer in internet, we need a pair of socket address i.e client socket address & server socket add.

5) four pieces of information are part of NIW layer Packet header & TCP layer Packet header. The first header contain IP address & 2nd header contain port NO.

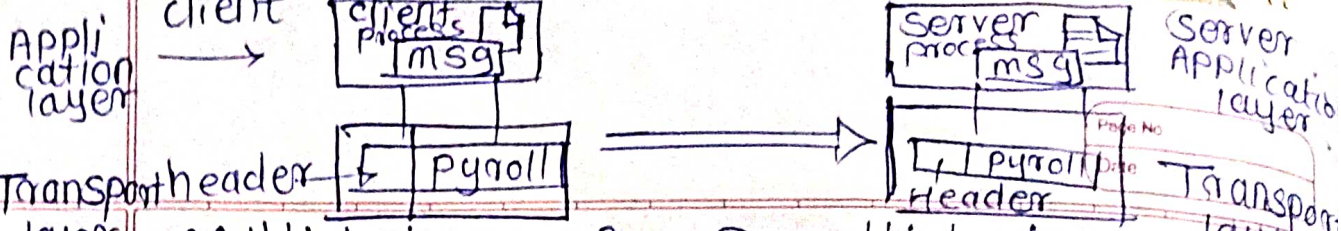
Encapsulation & Decapsulation

1) To send message from one process to another. TCP protocol encapsulate & Decapsulate message.

2) At sender site encapsulation occur. message send to Transport layer along pair of socket address. add header

3) The packet Transport layer in Internet called datagram, seg, or packet.

4) At receiver site Decapsulation. message is received. header is dropped & messge send to Application layer.



Multiplexing & Demultiplexing.

1) Whenever entity accepts items fro more than one source is multiplex ng

2) Whenever entity deliver item to more than one source it is ref er demultiplexing.

3) Three client process running at client site P1 & P2, P3, P1 & P3 need to send request to server The client process P2 send reque another server.

4) Transport layer accept messag from 3 process & createspac It act multiplexer.

5) The Packet 1 & 3 use same logical channel to reach Transport layer of 1st server.

6) They arrive at server Transpor layer does job of multiplexer.

7) The Transport layer at send Second server receives packet 2

Pushing or Pulling

1) delivery of item from producer to consumer. can occur in one of Two ways.

2) Pushing or pulling
If Sender deliver item when They are produced without any request from Consumer The delivery is refer Pushing.

3) If producer deliver item after consumer has request. The delivery is pulling.

4) When producer pushes the item Consumer will be overhelmed. & it need to flow contool.

5) When consumer pull the item it request them when it is ready.

In case. No ↓ flow control Need.

Page No.
Date

flow Control at Transport layer

① In communication at Transport layer four entities :- sender process, sender Transport layer, receiver process, receiver Transport layer.

② The sending process at application layer is only producer. It produce message chunk and pushes them to Transport ~~Transpoo~~ layer.

③ The sending transport layer has double role it is both a consumer & producer. It consumed message. Pushed by producer. It encapsulates

④ message in packet & pushes Them to Receiving Transport layer.

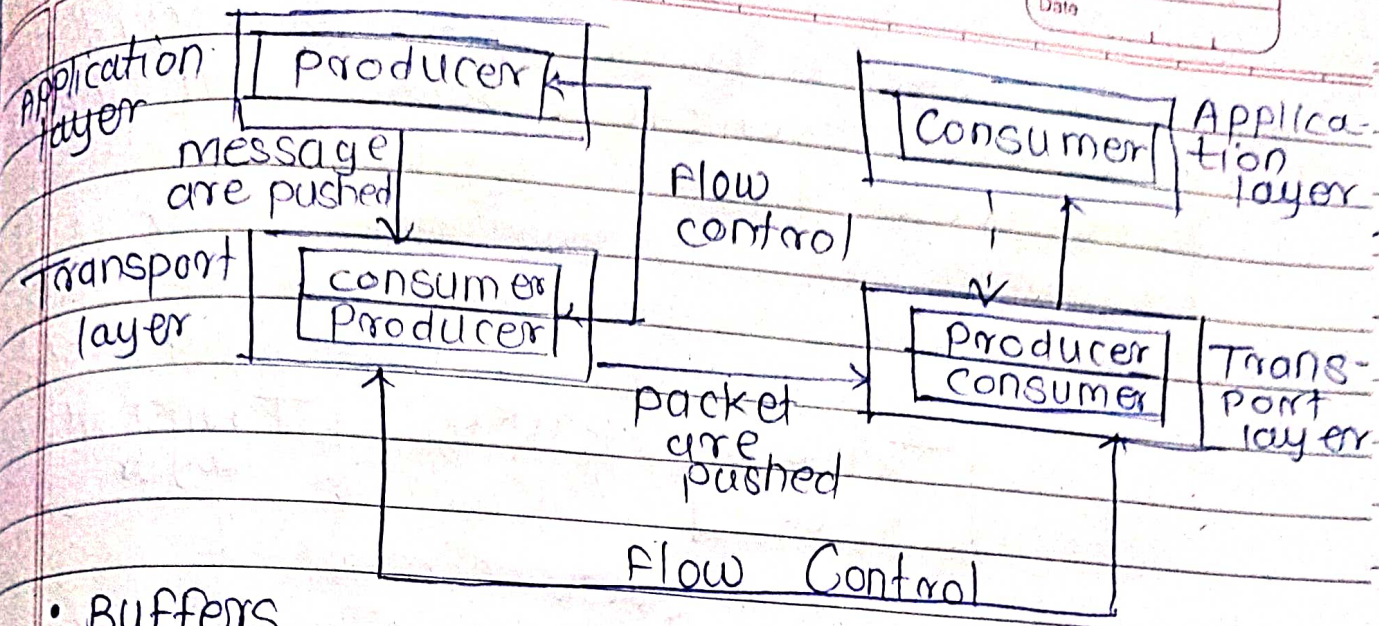
⑤ Receiving Transport 2 layer. it is consumer for packet received from the sender. It is also produce it need to decapsulate message & deliver to application layer.

⑥ The last delivery, normally a pulling delivery The Transport layer wait until application layer process ask for msg.

⑦ Two cases of flow Control:

① from sending Transport layer to sending application layer.

② from receiving transport layer to receiving Transport layer.



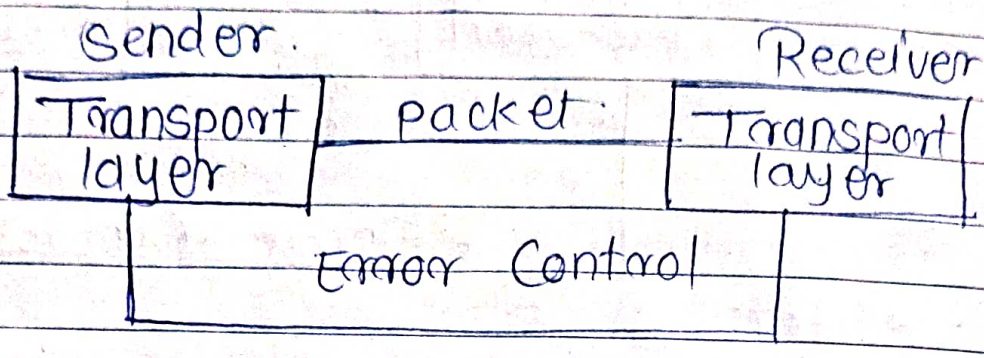
- Buffers

1) Buffer is set of memory location that can hold packet at sender & receiver.
2) When buffer of sending transport layer is full, it inform the application layer to stop passing chunk of message. When there ~~so~~ are some vancancy it inform application layer that it can pass message chunks again.
3) When buffer of Receiveing Transport layer is full it inform Sending Tcp layer to stop sending Packet. When There are some vacancy it inform ~~Application layer~~ Transport layer.

Errol Control:

1) Reliability can be acheived to add error Control service to Transport layer.
2) Error Control at Transport layer is responsible to
   1. Detect & discard Corrupted Packet.
   2. keep track of lost & discarded packet & re-send them.

3. Recognize duplicate packet & discard Them
4. Buffer out of order packet until The missing packet arrive.



5) For error control. The sequence Number are modulo 2^m. where m is size of Seq^n number field in bit.

Sequence Number:-

Error control require that the sending Transport layer knows which packet is to resent & receiving Transport layer knows which packet is duplicate.
if packet are numbered we can add field to transport layer packet to hold Seq^n number. of packet.

Sliding window:
1) Sequence Number used modulo 2^m. a circle can represent number from 0 to $2^m - 1$
2) The buffer is represented as set of slice called sliding window.
3) At sender site when packet is sent,

The corresponding slice is marked.

4) When all slice are marked it means buffer full & no message can accept from application layer.

5) When ack arrives corresponding slice unmarked.

6) If some consecutive slice from beginning of window are unmarked. The window slides over range of corresponding seq? number to allow more free slices at end of window.

Conjestion Control:

1) An important issue in internet Conjestion
2) Conjestion in a network may occur if the load on the Network the no. of packet sent to Network is greater than capacity of Network. - the no. of packet a netwo can handle.
3) Conjection in a network occur because router & switches have Queues buffer tha hold packet before & after processing.
4) Conjestion control refer to Technique & mechanism that can either prevent. congestion before it happen. or remove Congestion after it happen.

Open-Loop Conjection Control.
In this policy are apply to Prevent conjestion before it happen In these mechanism, Congestion control is handle by either the source or the destinatio

Retransmission policy: If sender feels that a sent packet is lost or corrupted. Hence packet need to sent retransmit. However good Retransmission policy can Prevent Conjestion.

window policy: The Type of window at sender may also effect Congestion. we will see later in Chapter Selective Repeat window is better than Go-Back-N for conjestion control

Acknowledgment policy. The Acknowledgment policy imposed by receiver may also affect congestion. If the receiver does not acknowledge every packet it receives it may slow down the sender & help prevent Congestion.

Connectionless Service

1) In a connectionless service, The source process needs to divide its message into chunk of data of size acceptable by Transport layer & deliver them to Transport layer one by one.
2) Transport layer each treat each chunk as a single unit without any relation betn chunk.
3) When chunks arrives from application layer The Transport layer encapsulate it in packet & send it. To show the independency of packet, assume that client process has 3 chunks of message to send, server process. The chunks are handed over to the connectionless transport protocol in order.
4) There is no dependency betn the packet at Transport layer. The packet may arrive out of order at destination & will be delivered out of order to server process.

ex. Fig shows 3 chunks of message are delivered to client transport layer in order (1, 2 & 3) because of extra layer in Transport of second packet. The delivery of messages at the server is not in order (1, 3 & 2)

(5) The situation would be worse if one or packets were lost. There is no numbering on packet. The receiving transport layer has no idea that one of messages has been lost.

(6) The above two problem arise from fact that the two Transport layer do not co-ordinate with each other. (If we can say that no flow control, error control, or congestion control can be effectively implemented in connectionless service.

Connectionless - oriented Service.

1) In Connectionless - oriented service, the client & the server first need to establish connection betⁿ themselves.
2) The data exchange can only happen after the connection establish.
3) The connection oriented service at the Transport layer is different from same service at the Network layer. In the Network layer, connection-oriented service.

mean a coordinating between the two end hosts and all the routers in betn.

(4) At Transport layer, connection-oriented service involves only the two hosts service is end to end.

(5) This means that we should be able to make a connection-oriented protocol over either a connectionless or connection oriented protocol.

(6) We can implement flow control, & congestion control in a connection oriented protocol.

Page No
Date

Transport - Layer protocols

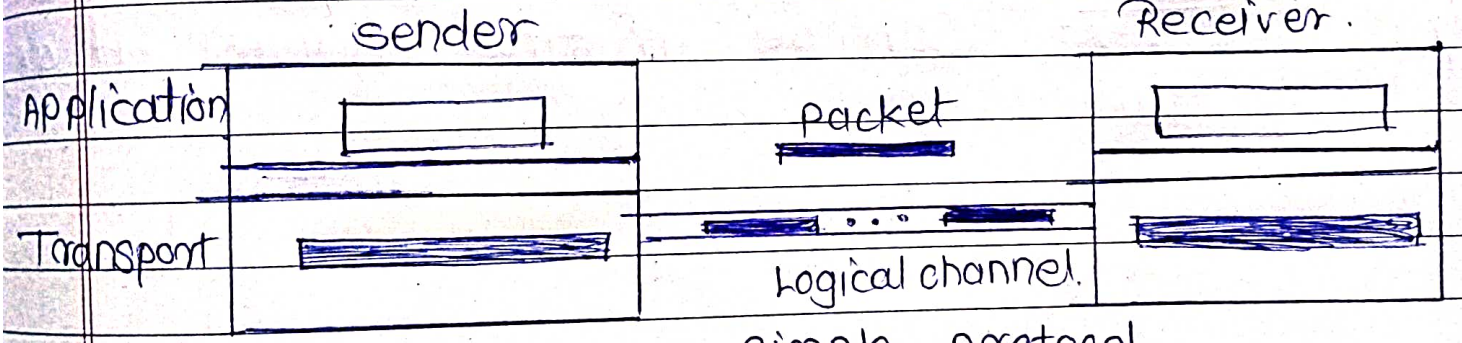
① simple protocols:

(1) Our first protocol is simple connectionless protocol with either flow nor error control.

(2) The Transport layer at sender gets a message from its application layer. make packet out of it & send the packet.

(3) The Transport layer at receiver receives a packet from it NIW layer, extract message from packet & deliver message to application layer.

(4) Transport layer of sender & receiver provide transmission services for their Application layer



simple protocol

② Stop & wait protocol

① It is second protocol & it is connection-oriented protocols called stop&wait protocol

② It used both Error control & Flow control.

③ Sender & receiver use a sliding window size 1.

④ The sender sends one packet at time. &

waits for an acknowledgment before sending the next one.

(5) When packet arrives. To detect corrupted packet we need to add checksum to each data packet.

(6) When packet arrive at receiver site, it is checked. If it's checksum is incorrect, The packet is corrupt.

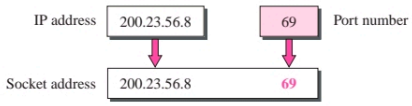
(7) Every time the sender send a packet it starts a timer. If an acknowledgement arrive before the timer expires, The timer is stopped & sender sends the next packet.

(8) If the Timer expires the Timer sender resends the previous packet, This means Sender keep copy of packet until its acknowledgment arrive.

Note only one packet & one acknowledgement can be in the channel at any time.

A transport-layer protocol in the TCP suite needs both the IP address and the port number, at each end, to make a connection. The combination of an IP address and a port number is called a **socket address.** The client socket address defines the client process uniquely just as the server socket address defines the server process uniquely (see Figure 13.5).

Figure 13.5 *Socket address*



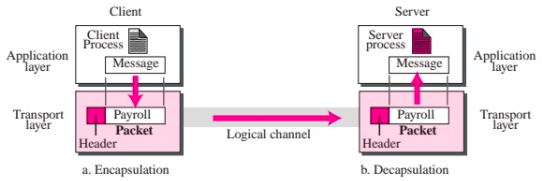
To use the services of transport layer in the Internet, we need a pair of socket addresses: the client socket address and the server socket address. These four pieces of information are part of the network-layer packet header and the transport-layer packet header. The first header contains the IP addresses; the second header contains the port numbers.

Encapsulation and Decapsulation

To send a message from one process to another, the transport layer protocol encapsulates and decapsulates messages (Figure 13.6).

Encapsulation happens at the sender site. When a process has a message to send, it passes the message to the transport layer along with a pair of socket addresses and

Figure 13.6 *Encapsulation and decapsulation*



some other pieces of information that depends on the transport layer protocol. The transport layer receives the data and adds the transport-layer header. The packets at the transport layers in the Internet are called *user datagrams*, *segments*, or *packets*. We call them packets in this chapter.

Decapsulation happens at the receiver site. When the message arrives at the destination transport layer, the header is dropped and the transport layer delivers the message to the process running at the application layer. The sender socket address is passed to the process in case it needs to respond to the message received.

Multiplexing and Demultiplexing

Whenever an entity accepts items from more than one source, it is referred to as **multiplexing** (many to one); whenever an entity delivers items to more than one source, it is referred to as **demultiplexing** (one to many). The transport layer at the source performs multiplexing; the transport layer at the destination performs demultiplexing (Figure 13.7).

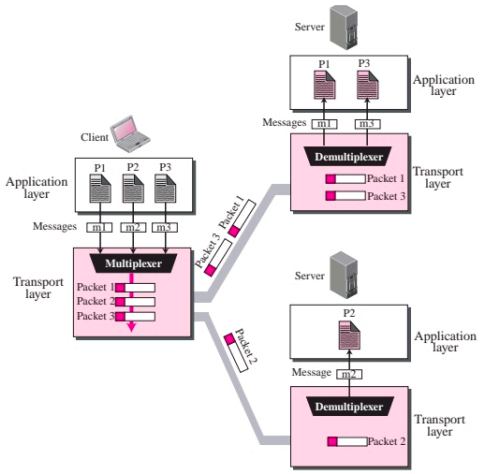
Figure 13.7 shows communication between a client and two servers. Three client processes are running at the client site, P1, P2, and P3. The processes P1 and P3 need to send requests to the corresponding server process running in a server. The client process P2 needs to send a request to the corresponding server process running at another server. The transport layer at the client site accepts three messages from the three processes and creates three packets. It acts as a *multiplexer*. The packets 1 and 3 use the same logical channel to reach the transport layer of the first server. When they arrive at the server, the transport layer does the job of a *multiplexer* and distributes the messages to two different processes. The transport layer at the second server receives packet 2 and delivers it to the corresponding process.

processes are running at the client site, P1, P2, and P3. The processes P1 and P3 need to send requests to the corresponding server process running in a server. The client process P2 needs to send a request to the corresponding server process running at another server. The transport layer at the client site accepts three messages from the three processes and creates three packets. It acts as a *multiplexer*. The packets 1 and 3 use the same logical channel to reach the transport layer of the first server. When they arrive at the server, the transport layer does the job of a *multiplexer* and distributes the messages to two different processes. The transport layer at the second server receives packet 2 and delivers it to the corresponding process.

Flow Control

Whenever an entity produces items and another entity consumes them, there should be a balance between production and consumption rates. If the items are produced faster than they can be consumed, the consumer can be overwhelmed and needs to discard some items. If the items are produced slower than they can be consumed, the consumer should wait; the system becomes less efficient. Flow control is related to the first issue. We need to prevent losing the data items at the consumer site.

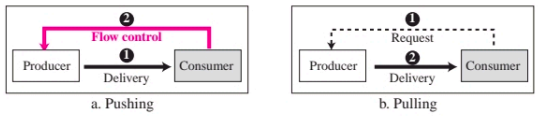
Figure 13.7 *Multiplexing and demultiplexing*



Pushing or Pulling

Delivery of items from a producer to a consumer can occur in one of the two ways: *pushing* or *pulling*. If the sender delivers items whenever they are produced—without the prior request from the consumer—the delivery is referred to as pushing. If the producer delivers the items after the consumer has requested them, the delivery is referred to as pulling. Figure 13.8 shows these two types of delivery.

Figure 13.8 *Pushing or pulling*



When the producer *pushes* the items, the consumer may be overwhelmed and there is a need for flow control, in the opposite direction, to prevent the discarding of the items. In other words, the consumer needs to warn the producer to stop the delivery and to inform it when it is ready again to receive the items. When the consumer pulls the items, it requests them when it is ready. In this case, there is no need for flow control.

order packets can be either discarded (to be treated as lost packets by the sender), or stored until the missing ones arrives.

Combination of Flow and Error Control

We have discussed that flow control requires the use of two buffers, one at the sender site and the other at the receiver site. We have also discussed that the error control requires the use of sequence and acknowledgment numbers by both sides. These two requirements can be combined if we use two numbered buffers, one at the sender, one at the receiver.

At the sender, when a packet is prepared to be sent, we use the number of the next free location, *x*, in the buffer as the sequence number of the packet. When the packet is sent, a copy is stored at memory location *x*, awaiting the acknowledgment from the other end. When an acknowledgment related to a sent packet arrives, the packet is purged and the memory location becomes free.

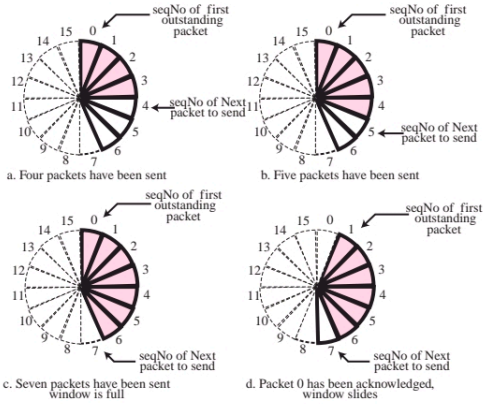
At the receiver, when a packet with sequence number y arrives, it is stored at the memory location *y* until the application layer is ready to receive it. An acknowledgment can be sent to announce the arrival of packet *y*.

Sliding Window

Since the sequence numbers used modulo 2^m, a circle can represent the sequence number from 0 to $2^m - 1$ (Figure 13.11).

The buffer is represented as a set of slices, called the **sliding window,** that occupy part of the circle at any time. At the sender site, when a packet is sent, the corresponding slice is marked. When all the slices are marked, it means that the buffer is full and no further messages can be accepted from the application layer. When an acknowledgment

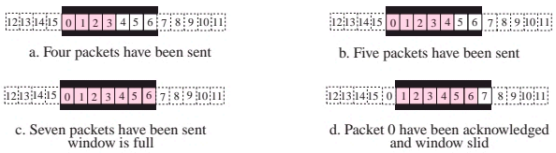
Figure 13.11 *Sliding window in circular format*



arrives, the corresponding slice is unmarked. If some consecutive slices from the beginning of the window are unmarked, the window slides over the range of the corresponding sequence number to allow more free slices at the end of the window. Figure 13.11 shows the sliding window at the sender. The sequence number are modulo 16 ($m = 4$) and the size of the window is 7. Note that the sliding window is just an abstraction: the actual situation uses computer variables to hold the sequence number of the next packet to be sent and the last packet sent.

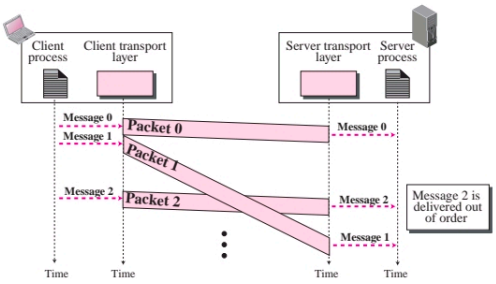
Most protocols show the sliding window using linear representation. The idea is the same, but it normally takes less space on paper. Figure 13.12 shows this representation. Both representations tell us the same thing. If we take both sides of each part in Figure 13.11 and bend them up, we can make the same part in Figure 13.12.

Figure 13.12 *Sliding window in linear format*



layer, the packets may arrive out of order at the destination and will be delivered out of order to the server process. In Figure 13.13, we have shown the movement of packets using a time line, but we have assumed that the delivery of the process to the transport layer and vice versa are instantaneous.

Figure 13.13 *Connectionless service*



The figure shows that at the client site, the three chunks of messages are delivered to the client transport layer in order (1, 2, and 3). Because of the extra delay in transportation of the second packet, the delivery of messages at the server is not in order (1, 3, 2). If these three chunks of data belong to the same message, the server process may have received a strange message.

The situation would be worse if one of the packets were lost. Since there is no numbering on the packets, the receiving transport layer has no idea that one of the messages has been lost. It just delivers two chunks of data to the server process.

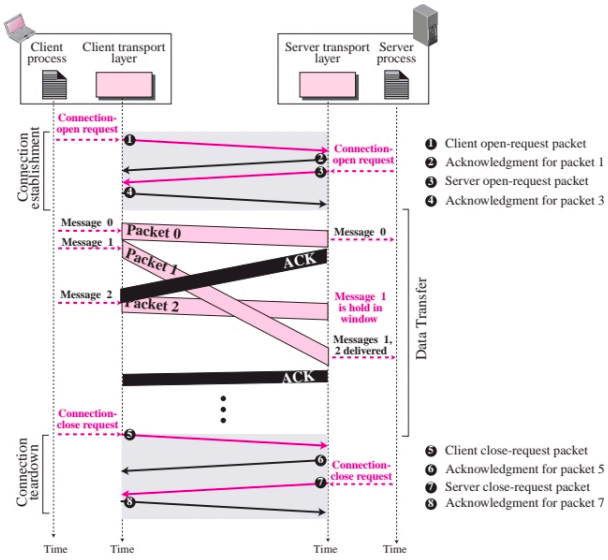
The above two problems arise from the fact that the two transport layers do not coordinate with each other. The receiving transport layer does not know when the first packet will come nor when all of the packets have arrived.

We can say that no flow control, error control, or congestion control can be effectively implemented in a connectionless service.

Connection-Oriented Service

In a connection-oriented service, the client and the server first need to establish a connection between themselves.The data exchange can only happen after the connection establishment. After data exchange, the connection needs to be teared down (Figure 13.14). As we mentioned before, the connection-oriented service at the transport layer is different from the same service at the network layer. In the network layer, connection-oriented

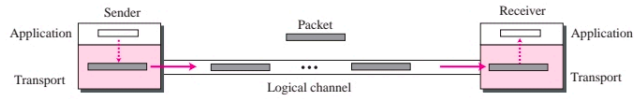
Figure 13.14 *Connection-oriented service*



Simple Protocol

Our first protocol is a simple connectionless protocol with neither flow nor error control. We assume that the receiver can immediately handle any packet it receives. In other words, the receiver can never be overwhelmed with incoming packets. Figure 13.16 shows the layout for this protocol.

Figure 13.16 *Simple protocol*

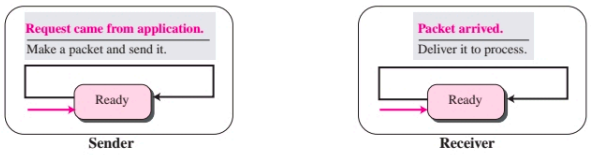


The transport layer at the sender gets a message from its application layer, makes a packet out of it, and sends the packet. The transport layer at the receiver receives a packet from its network layer, extracts the message from the packet, and delivers the message to its application layer. The transport layers of the sender and receiver provide transmission services for their application layers.

FSMs

The sender site should not send a packet until its application layer has a message to send. The receiver site cannot deliver a message to its application layer until a packet arrives. We can show these requirements using two FSMs. Each FSM has only one state, the *ready state*. The sending machine remains in the ready state until a request comes from the process in the application layer. When this event occurs, the sending machine encapsulates the message in a packet and sends it to the receiving machine. The receiving machine remains in the ready state until a packet arrives from the sending machine. When this event occurs, the receiving machine decapsulates the message out of the packet and delivers it to the process at the application layer. Figure 13.17 shows the FSMs for the simple protocol. We see in Chapter 14 that UDP protocol is a slight modification of this protocol.

Figure 13.17 *FSMs for the simple protocol*

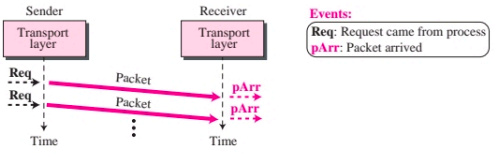


The simple protocol is a connectionless protocol that provides neither flow nor error control.

Example 13.3

Figure 13.18 shows an example of communication using this protocol. It is very simple. The sender sends packets one after another without even thinking about the receiver.

Figure 13.18 *Flow diagram for Example 13.3*



Stop-and-Wait Protocol

Our second protocol is a connection-oriented protocol called the **Stop-and-Wait protocol,** which uses both flow and error control. Both the sender and the receiver use a sliding window of size 1. The sender sends one packet at a time and waits for an acknowledgment before sending the next one.To detect corrupted packets, we need to add a checksum to each data packet. When a packet arrives at the receiver site, it is checked. If its checksum is incorrect, the packet is corrupted and silently discarded. The silence of the receiver is a signal for the sender that a packet was either corrupted or lost. Every time the sender sends a packet, it starts a timer. If an acknowledgment arrives before the timer expires, the timer is stopped and the sender sends the next

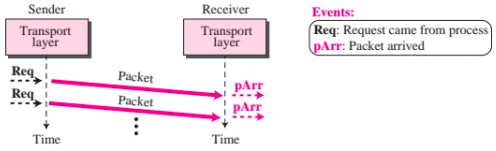
Sender Receiver

The simple protocol is a connectionless protocol that provides neither flow nor error control.

Example 13.3

Figure 13.18 shows an example of communication using this protocol. It is very simple. The sender sends packets one after another without even thinking about the receiver.

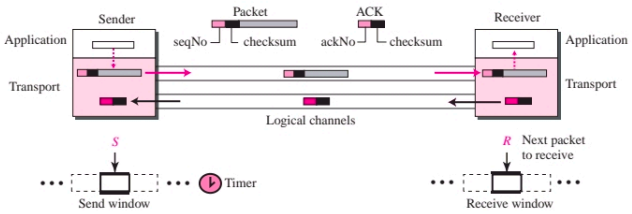
Figure 13.18 *Flow diagram for Example 13.3*



Stop-and-Wait Protocol

Our second protocol is a connection-oriented protocol called the **Stop-and-Wait protocol,** which uses both flow and error control. Both the sender and the receiver use a sliding window of size 1. The sender sends one packet at a time and waits for an acknowledgment before sending the next one.To detect corrupted packets, we need to add a checksum to each data packet. When a packet arrives at the receiver site, it is checked. If its checksum is incorrect, the packet is corrupted and silently discarded. The silence of the receiver is a signal for the sender that a packet was either corrupted or lost. Every time the sender sends a packet, it starts a timer. If an acknowledgment arrives before the timer expires, the timer is stopped and the sender sends the next packet (if it has one to send). If the timer expires, the sender resends the previous packet assuming that either the packet was lost or corrupted. This means that the sender needs to keep a copy of the packet until its acknowledgment arrives. Figure 13.19 shows the outline for the Stop-and-Wait protocol. Note that only one packet and one acknowledgment can be in the channels at any time.

Figure 13.19 *Stop-and-Wait protocol*



The Stop-and-Wait protocol is a connection-oriented protocol that provides flow and error control.

In Stop-and-Wait protocol, flow control is achieved by forcing the sender to wait for an acknowledgment, and error control is achieved by discarding corrupted packets and letting the sender resend unacknowledged packets when the timer expires.

Sequence Numbers

To prevent duplicate packets, the protocol uses sequence numbers and acknowledgment numbers. A field is added to the packet header to hold the sequence number of that packet. One important consideration is the range of the sequence numbers. Since we want to minimize the packet size, we look for the smallest range that provides unambiguous communication. Let us reason out the range of sequence numbers we need. Assume we have used x as a sequence number; we only need to use $x + 1$ after that. There is no need for $x + 2$. To show this, assume that the sender has sent the packet with sequence number x. Three things can happen.

1. The packet arrives safe and sound at the receiver site; the receiver sends an acknowledgment. The acknowledgment arrives at the sender site, causing the sender to send the next packet numbered $x + 1$.