Introduction, Relationship between transport layer and network layer, transport layer in the Internet, Multiplexing and De-Multiplexing, Connectionless transport, Reliable data transfer: Building a reliable data transfer protocol, Pipelined reliable data transfer protocol, GoBack-N (GBN), Selective Repeat (SR), Connection oriented transport : TCP , TCP connection, TCP segment structure, Time estimation and time out, Flow control, Principle of Congestion Control: The causes and costs of congestion, approaches to congestion control.

The primary duties of the transport layer are to transport and regulate the flow of information from a source to a destination, reliably and accurately. End-to-end control and reliability are provided by sliding windows, sequencing numbers and acknowledgements.
To understand reliability and flow control think of someone who studies a foreign language for one year and then visits the country where the language is used. In conversation, words must be repeated for the reliability. People must also speak slowly that the conversation is understood, which relates to flow control.
The transport layer establishes a logical connection between two end points of a network. Protocols in transport layer segment and reassemble data sent by upper layer applications into the same transport layer data string. This transport layer data string provides end-to-end transport services.

Functions/ Features/ Characteristics :

- Error handling
- Flow control
- Multiplexing
- Connection set-up and release
- Segmentation and reassembly
- Addressing (Port addressing)

Services

- Connectionless Service:- Unreliable unordered unicast or multicast delivery (UDP)
- Connection Oriented Service:- Reliable, in-order unicast delivery (TCP)

Relationship between the transport layer and the network layer

- In computing and telecommunications, the transport layer is the second highest layer in the four and five layer TCP/IP reference models, where it responds to service requests from the application layer and issues service requests to the Internet layer.
- It is also the name of layer four of the seven layer OSI model, where it responds to service requests from the session layer and issues service requests to the network layer. The definitions of the transport layer are slightly different in these two models.

Connectionless Transport: UDP (User Datagram Protocol)

UDP is the connectionless transport protocol in the TCP/IP protocol stack. UDP is a simple protocol that exchanges datagram without guaranteed delivery. It relies on higher layer protocols to handle error and retransmit data.
UDP doesn't use window or Asks reliability is provided by application layer protocols. UDP is designed for applications that do not need to put sequence of segments together.
The following application layer protocols use UDP: TFTP, SNMP, DHCP, and DNS
Hence,

- Used in transport layer
- Offers unreliable connectionless service
- Provides faster service than that of TCP.
- Offers minimum error checking mechanism.
- Supports multicasting because connectionless.
- Offers minimum flow control mechanism.
- Also used by SNMP (Simple Network Management Protocol)

UDP Segment Structure:

Source port number (16)	Destination port number (16)

UDP segment length (16)	UDP checksum (16)
Data	

→ Source port – number of the port that sends data.
→ Destination port – Number of the port that receives data.
→ Length – calculated of bytes in header and data.
→ Checksum – calculated checksum of the header and data field.
→ Data – upper-layer protocol data.

Connection–Oriented Transport: TCP (Transmission Control Protocol)

TCP is a connection-oriented transport layer protocol that provides reliable full-duplex data transmission. TCP is port of the TCP/IP protocol stack. In a connection-oriented environment a connection is established between both ends before the transfer of information can begin. TCP breaks messages into segments, reassembles them at the destination and resends anything that is not received. TCP supplies a virtual circuit between end user applications:
The following application layer protocols use TCP, FTP, HTTP, SMTP and Telnet. Hence

→ This is a real protocol which runs in transport layer.
→ It offers reliable connection-oriented service between source and destination.
→ It acts as if it is connecting two end points together, so that it is a point-to-point connection between two parties.
→ It doesn't support multicasting (because it is connection oriented)
→ The data in TCP is called a segment.
→ Segments are obtained after breaking big files into small pieces.
→ It assists in flow control.
→ It provides buffer to each connection.

TCP segment Structure

Source port (16)			Destination port (16)
Sequence number (32)			
Acknowledgement Number (32)			
Header length (4)	Reserved (4)	code bits (6)	window site (16)
Checksum (16)		Urgent Pointer (16)	
Options (0 or 32 if any)			
Data			

Source port – Number of the port that sends data.
Destination port – Number of the port that receives data.
Sequence number – Number used to ensure the data arrives in the correct order.
Acknowledgement number – next expected TCP octet (defines the number of next byte, a party, expects to receive)
Header length – length of TCP header.
Reserved – reserved for future use.
Code bits – control functions such as setup and termination of a session.

U	A	P	R	S	F

U -urgent valid
A –acknowledges received data
P –data push is valid
R –reset valid
S –synchronization valid (initiates a connection)
F –final valid (terminates a connection)

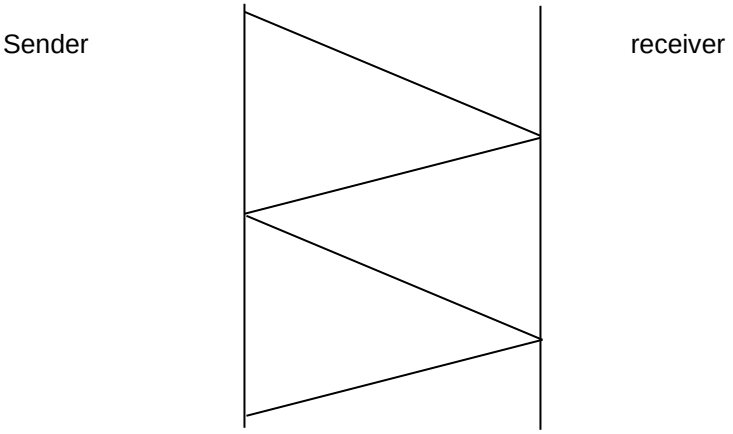
Window size - number of octets (bytes) that a receiver is willing to accept.
Checksum – indicates the end of the urgent data.
Option – used when sender and receiver negotiate the maximum segment size.

Data – upper-layer protocol data.

TCP vs. UDP

TCP	UDP
Reliable	Unreliable
Connection-oriented	Connectionless
Segment retransmission and flow control through windowing	No windowing or retransmission
Segment sequencing	No sequencing
Acknowledge sequencing	No acknowledgment

Roundtrip Time (RTT) Estimation And Timeout



→

→

→ RTT, also called round trip delay is the time required for a signal pulse or packet to travel from a specific source to a specific destination and back again.

→ The sample RRT, denoted as sample RTT, for a segment is the amount of time between when the segment is sent and when an acknowledgement for the segment is received.

→ Obviously, the sample RRT values will fluctuate from segment to segment due to congestion in the routers and to the varying loads on the end systems.

→ In order to estimate a typical RTT, it is natural to take some sort of average of the sample RTT values, called Estimated RRT as

Estimated RRT = (1-x)*estimated RRT + x*sample RRT

Where x=0.125 recommended.

→ Hence, the new value of estimated RRT is a weighted combination of the previous value of estimated RRT.

→ This weighted average puts more weight on recent samples than the old samples. This is natural, as the more recent samples better reflect the current congestion in the network. In statistics, such an average is called exponential weighted moving average (EWMA).

→ In addition to having an estimate of the RTT, it is valuable to have a measure of the variability of the RTT, called DevRTT an estimate of how much sample RRT typically deviates from estimated RRT.

DevRTT = (1-y)*DevRTT+y*|sampleRTT-Estimated RTT|

Where y=0.25 recommended.

Note that, DevRTT is an EWMA of the difference between sample RTT and estimated RTT, if the sample RTT values have little fluctuation, then Dev RTT will be small and vice versa.

Setting and Managing The Retransmission Timeout Interval

Given the values of EstimatedRTT and DevRTT, what value should be used for TCP's timeout interval? Clearly, the interval should be greater than or equal to EstimatedRTT, or unnecessary retransmissions would be sent. But the timeout interval should not be much larger than Estimated RTT; otherwise when a segment is lost, TCP would not quickly retransmit the segment, leading to large data transfer delays. It is therefore desirable to set the timeout equal to the EstimatedRTT plus some margin. The margin should be large when there is a lot of fluctuation in the sample RTT values, it should be small when there is little fluctuation. The value of DevRTT should thus come into play here:

Timeout Interval = estimated RTT + 4 * Dev RTT

Multiplexing and De-multiplexing

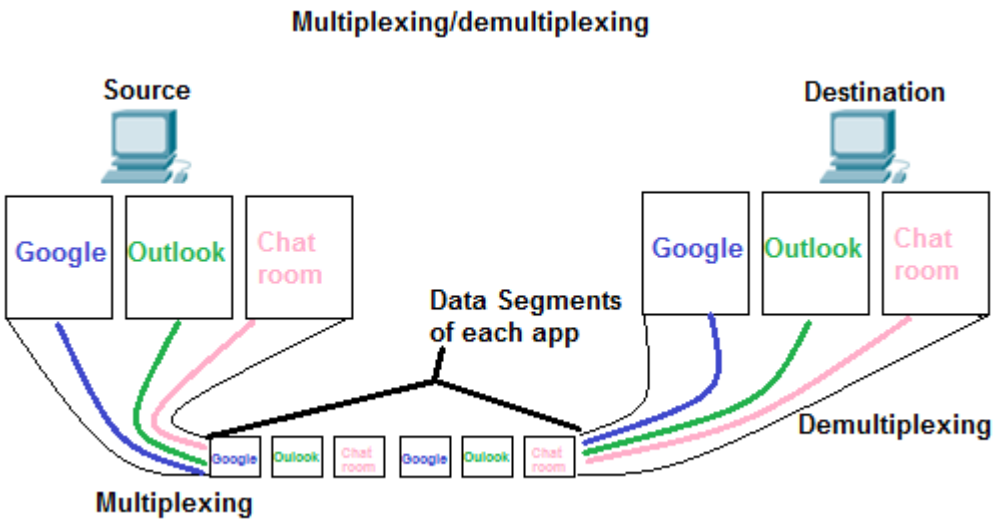


Fig.: Multiplexing and De-multiplexing

→ Extending the host-to-host delivery service provided by the network layer to a process-to-process delivery service application running on the hosts.

→ Consider how a receiving host directs on incoming transport layer segment to the appropriate socket. Each transport layer segments has a set of fields for this purpose. At the receiving end, the transport layer examines these fields to identify the receiving socket and then it directs the segment to that socket. The job of delivering the data in the transport layer segment to the correct socket is called de-multiplexing. The job of gathering data chunks at the source host from the different sockets, encapsulating each data chunk with the header information to create segments and passing the segments to the network layer is called multiplexing.

Flow Control

A TCP connection sets aside a receiver buffer for the connection. When the TCP connection receives bytes that are correct and in sequence, it places the data in the receiver buffer. The associated application process will read data from this buffer, but not necessarily at this instant the data arrives. Indeed, the receiving application may be busy with some other task and may not attempt to read the data until longer after it has arrived. If the application is relatively slow at reading data the sender can very easily overflow the receive buffer by sending too much data quickly.

TCP provides a flow control service to its application to estimate the possibility of the sender overflowing the receive buffer. Flow control is thus a speed-matching service matching the rate at which the sender is sending against the rate at which receiving application is receiving.

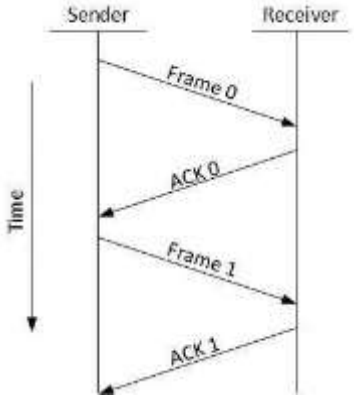
Before knowing how the layers are actually controlling the flow via various algorithms, you must know the reason why is it actually necessary.

Flow control in transport layer ensures the delivery of the message globally, as the two points of connection over this protocol are logically connected.

Whereas in data-link layer, the concern is to deliver message locally, as the two points of connection over this protocol are physically connected.

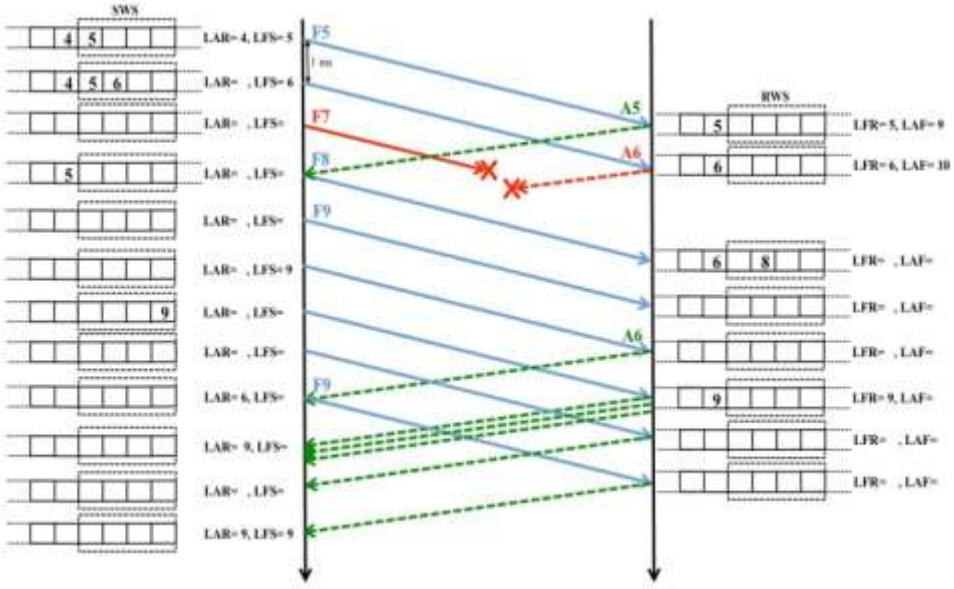
Now, coming upon the algorithms that control flow of a network:

1. **Stop and Wait -** This flow control mechanism forces the sender after transmitting a data frame to stop and wait



until the acknowledgement of the data-frame sent is received.

2. **Sliding Window -** In this flow control mechanism, both sender and receiver agree on the number of data-frames after which the acknowledgement should be sent. As we learnt, stop and wait flow control mechanism wastes resources, this protocol tries to make use of underlying resources as much as



possible.

These are the 2 basic algorithms for flow control, whereas others are used for error control mechanism.

TCP uses the sliding window protocol for flow control, the size of which is dependent upon the bandwidth, RTT and errors in packets.

Congestion

When too many packets are present in a subnet or a part of subnet, performance degrades. This situation is called congestion. When number of packets dumped into the subnet by the hosts is within its carrying capacity, they are all delivered (except for a few that contain 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. At very high traffics, performance collapses completely and almost no packets are delivered.

Causes of Congestion

→ When there are more input lines and less or single output lines.

→ When there is slow router i.e., if routers CPU's, are slow

→ If the router has no free buffers i.e., insufficient memory to hold queue of packets.

→ If the components used in subnet (link, router, switches, etc) have different traffics carrying and switching capacities, then congestion occurs.

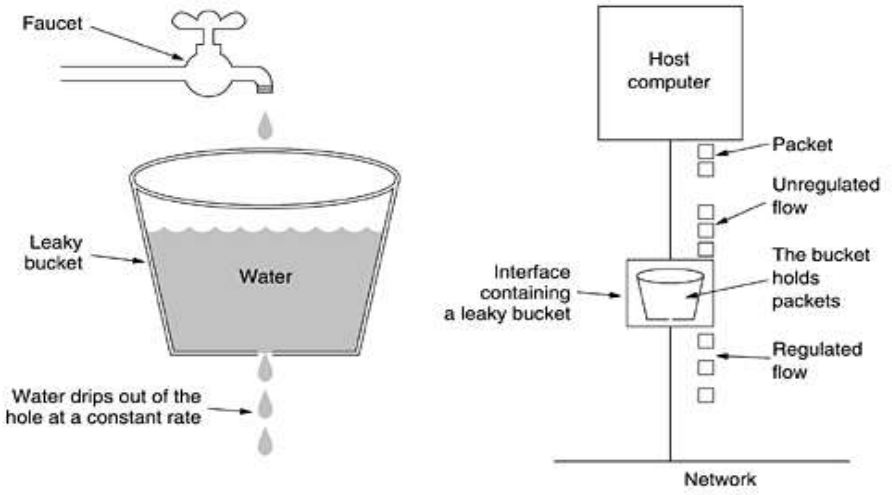
→ If the bandwidths of the lines are low, it can't carry large volume of packets and caused congestion. Hence, congestion cannot be eradicated but can be controlled.

Congestion Control Algorithms

i) **Leaky Bucket Algorithm**
ii) **Token Bucket Algorithm**
iii) **Choke Bucket Algorithm**

Leaky Bucket Algorithm

Figure (a) A leaky bucket with water. (b) A leaky bucket with packets.



Imagine a bucket with a small hole in the bottom. No matter at what rate water enters the bucket, the outflow is at constant rate, when there is any water in the bucket and zero when the bucket is empty. Also once the bucket is full any additional water entering it spills over the sides and is lost.

The same idea can be applied to the packets conceptually; each host is connected to the network by an interface, containing a leaky bucket, i.e. finite interval queue. If a packet arrives at the queue when it is full, the packet is discarded if one or more processes within the host try to send a packet when a maximum number are already is queue, the new packet is discarded.

Main working steps

- When the host has to send a packet , packet is thrown in bucket.
- Bucket leaks at constant rate.
- Bursty traffic is converted into uniform traffic by leaky bucket.
- In practice bucket is a finite queue outputs at finite rate.

Token Bucket Algorithm

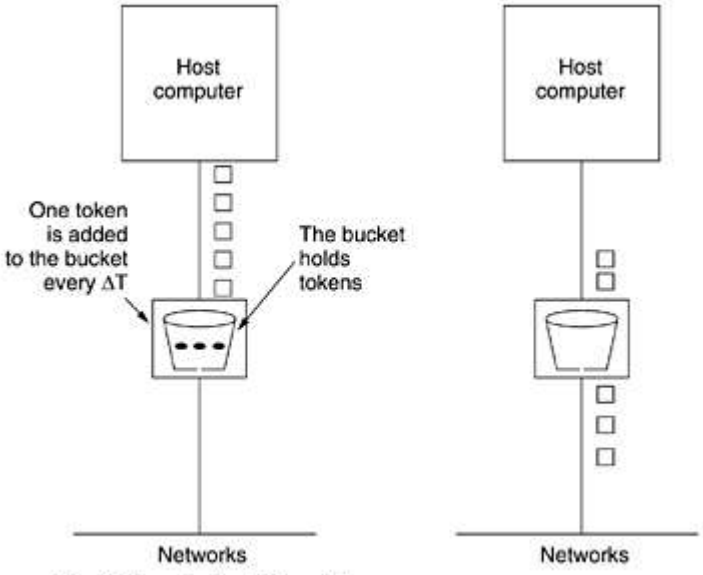


Fig. Token Bucket Algorithm

A leaky bucket algorithm is based on the rigid output pattern at the average rate no matter how bursty the traffic is. In leaky bucket, there are chances of loss of packet as packet is filled in bucket and overflow if bucket is full. To minimize such limitation of bucket, token bucket algorithm was introduced.

The bucket holds token, not packet. Tokens are generated by clock at the rate of one token per ΔT sec. for a packet to be transmitted; it must capture and destroy one token. Token bucket algorithm allows saving up permission to bucket as leaky bucket doesn't allow. This property means that bursts of packets can be sent at once allowing some burstness at output stream and giving faster response to sudden bursts of input.

Another difference between token bucket and leaky bucket is that the token bucket throws away token when the bucket fills up but never discards packets. In token bucket also allows sending bytes basis, for variable size packets. A packet can only be transmitted if enough token are available to cover its length in bytes fractional tokens are kept for further use.

The implementation of basic token bucket algorithm is just a variable counts tokens. The counter is incremented by one at every ΔT sec and decremented by one whenever one packet is sent. When counter hits zero, no packet can be sent.

Main working steps

- In this leaky bucket holds tokens generated at regular intervals of time.
- Bucket has maximum capacity.
- If there is a ready packet , a token is removed from Bucket and packet is send.
- If there is a no token in bucket, packet can not be send.

Leaky bucket vs. Token bucket

1) The leaky bucket is an algorithm that may be used to determine whether some sequence of discrete events conforms to defined limits on their average and peak rates or frequencies.
   The token bucket is an algorithm used in packet switched computer networks and telecommunications networks. It can be used to check that data transmissions, in the form of packets, conform to defined limits on bandwidth and burstiness.
2) The Leaky Bucket Algorithm is used to control rate in a network. In this algorithm the input rate can vary but the output rate remains constant.
   The Token Bucket Algorithm allows the output rate vary depending on the size of burst.
3) In Leaky bucket, Token independent.
   In Token bucket, Token dependent.
4) In Leaky bucket, if bucket is full, packet or data is discarded.
   In Token bucket, if bucket is full, token are discarded but not the packet.
5) In Leaky bucket, Packets are transmitted continuously.
   In Token bucket, Packets can only be transmitted when there are enough token.
6) In Leaky bucket, It sends the packet at constant rate.
   In Token bucket, It allows large bursts to be sent faster rate after that constant rate.

7) In Leaky bucket, It does not save token.
In Token bucket, It saves token to send large bursts.

Reliable Data Transfer (RDT)

RDT is the mechanism where no transferred data bits are corrupted (flipped from o to 1, or vice versa) or lost, and all are delivered in the order in which they were sent.
TCP creates a RDT service on top of IPs unreliable best effort service.
TCP's RDT service ensures that the data stream that a process reads out of its TCP receive buffer is uncorrupted, without gaps, without duplication and in sequence, that is, the byte stream is exactly the same byte stream that was sent by the end system on toe other side of the connection.

Building a RDT protocol

1) Reliable Data transfer over a Perfectly Reliable Channel

→ We first consider the simplest case, in which the underlying channel is completely reliable.
→ It is called finite-state machine (FSM).

2) Reliable Data Transfer over a channel with Bit Errors

→ A more realistic model of underlying channel is one in which bits in a packet may be corrupted.
→ If receiver receives the packet, the receiver must acknowledge it to the sender whether the packet has received with error-free or not through these:
Positive Acknowledgement (ACK)
Negative Acknowledgement (NAK)
→ If NAK provided, the sender should retransmit the packet.
→ Such protocol is called ARQ (Automatic Repeat Request) protocols.
→ Fundamentally, three additional protocol capabilities are required in ARQ protocols to handle the presence of bit errors :

- **Error Detection**

→ Internet checksum field
→ Error-detection and correction techniques
→ Require extra bits (beyond the bits of original data to be transferred) to be sent from the sender to the receiver, these bits will be gathered into the packet checksum field.

- **Receiver Feed Back**

→ Receiver provides feed back
→ Positive (ACK) -1 value
→ Negative (NAK) – 0 values

- **Retransmission**

→ A packet that is received in error at the receiver will be retransmitted by the sender.

These phenomena are called **stop-and-wait protocols**.

An amazing case will occur if ACK or NAK is corrupted i.e. the sender could not get the feedback from sender.
Consider three probabilities for handling corrupted ACKs or NAKs.

- A second alternative is to add enough checksum bits to allow the sender not only to detect, but also to receiver from bit errors. This solves immediate problem for a channel that can corrupt packets but not lose them.
- A third approach is for the sender simply to resend the current data packet when it receives a garbled ACK or NAK packet. This introduces duplicate packets into the sender-to-receiver channel. The receiver doesn't know whether the ACK or NAK it last sent was received correctly at the ender. Thus, it cannot know whether an arriving packet contains new data or is a retransmission.

→ A solution to this problem is to a new field called "sequence number" to the data packet.
→ For this stop-and-wait protocol, a 1-bit sequence number will be ok.

3) Reliable Data Transfer Over A lossy Channel with Bit Errors

→ Suppose now that in addition to corrupting bits, the underlying channel can lose packets as well.
→ The sender must get information of packet loss on the way from the receiver so that the sender can retransmit.

→ The sender must clearly wait at least as long as a round-trip delay between the sender and the receiver.
→ If ACK is not received within this time, the packet is retransmitted.
→ If a packet experiences a particularly large delay, the sender may retransmit the packet even though neither the data packet nor its ACK have been lost.
→ This introduces the possibility of duplicate data packets in the sender-to-receiver channel.
→ For all this, we can do is retransmit.
→ But implementing a time-based retransmission mechanism requires a "countdown timer" that an interrupt the sender after a given amount of time has expired.
→ The sender will thus need to be able to
- Start the timer each time a packet (either a first time packet or a retransmission) is sent.
- Respond to a timer interrupt (taking appropriate actions)
- Stop the timer.

Checksums: sequence numbers, timers, positive and negative acknowledgement 1 (page 254)

Reliable Data Transfer Protocol

1. **Pipelined**
2. **Go-Back-N(GBN)**
3. **Selective Repeat (SR)**

Pipelined Reliable Data Transfer Protocol:

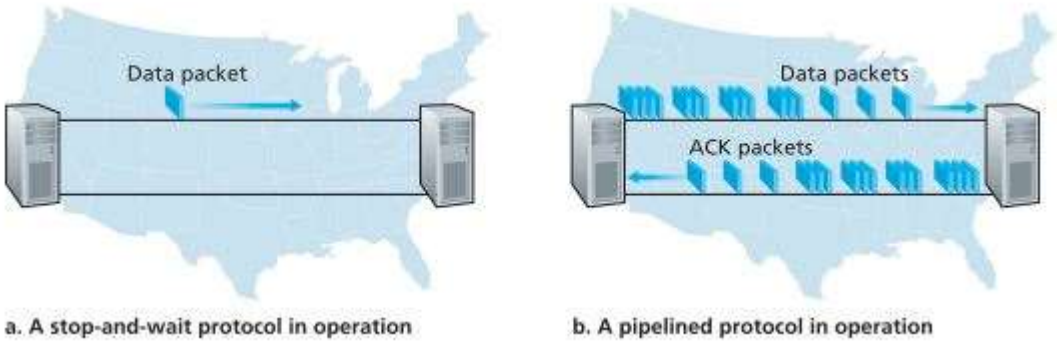


Fig: stop-and-wait vs. pipelined protocol

Instead of sending a single packet in stop and wait manner, the sender is allowed to send multiple packet without waiting for acknowledgements, as illustrate in fib (b). fig (b) shows that if the sender is allowed to transmit three packets before having to wait for acknowledgement the utilization of the sender is essentially tripled. Since the many in-transmit sender-to-receiver packets can be visualized as a filling a pipeline, this technique is known as pipelining.

Consequences of pipelined protocol

- Increment in the range of sequence numbers.
- Sender and receiver have to buffer more than one packet.
- Range of sequence numbers and the buffering requirements will depend on the manner in which a data transfer protocol responds to lost, corrupted and overly delayed packets.

Go-Back-N ARQ

Stop and wait ARQ mechanism does not utilize the resources at their best. When the acknowledgement is received, the sender sits idle and does nothing. In Go-Back-N ARQ method, both sender and receiver maintain a window.

The **sending**-window size *enables the sender to send multiple frames without receiving the acknowledgement of the previous ones.* The **receiving**-window enables the receiver to receive multiple frames and acknowledge them. The receiver keeps track of incoming frame's sequence number.

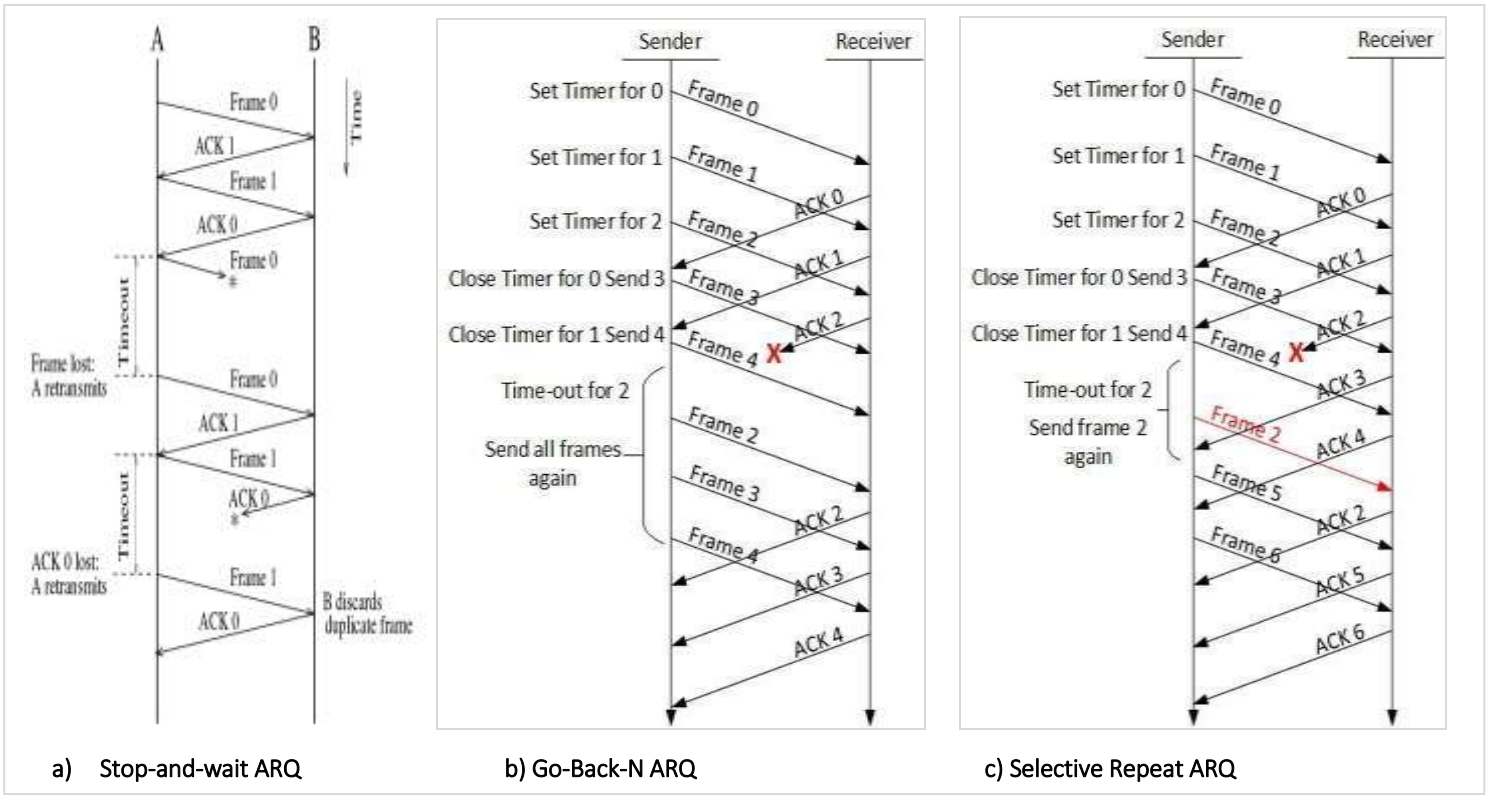
- When the sender sends all the frames in window, it checks up to what sequence number it has received positive acknowledgement. If all frames are positively acknowledged, the sender sends next set of frames.

- If sender finds that it has received NACK or has *not receive any ACK for a particular frame, it retransmits all the frames* **after which it does not receive any positive ACK.**

Selective Repeat ARQ

In Go-back-N ARQ, it is assumed that the receiver does not have any buffer space for its window size and has to process each frame as it comes. This enforces the sender to retransmit all the frames which are not acknowledged.

In Selective-Repeat ARQ, the receiver while keeping track of sequence numbers, buffers the frames in memory and sends NACK for only frame which is missing or damaged. The sender in this case, sends only packet for which NACK is received.



a) Stop-and-wait ARQ b) Go-Back-N ARQ c) Selective Repeat ARQ