Computer Networking
BCA V SEM

The Transport Layer

Kailash Karki
Kailash.karki@deerwalk.edu.np

Unit 5: The Transport Layer

5.1 Functions of Transport Layer

5.2 Elements of Transport Protocols: Addressing, Establishing and Releasing Connection, Flow Control & Buffering, Error Control, Multiplexing & Demultiplexing, Crash Recovery

5.3 User Datagram Protocol(UDP): User Datagram, UDP Operations, Uses of UDP, RPC

5.4 Principles of Reliable Data Transfer: Building a Reliable Data Transfer Protocol, Pipelined Reliable Data Transfer Protocol, Go-Back-N(GBN), Selective Repeat(SR)

5.5 Transmission Control Protocol(TCP): TCP Services, TCP Features, TCP Segment Header

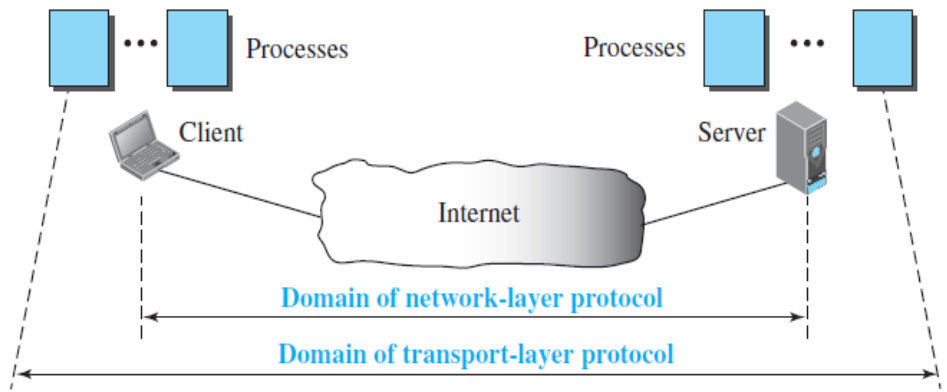
5.6 Principle of Congestion Control

Introduction

- The transport layer is the heart of the TCP/IP protocol suite; it is the end-to-end logical vehicle for transferring data from one point to another in the Internet.

- Transport layer is responsible for process-to-process delivery of the entire message. A process is a network application running on a host. Whereas the network layer oversees source-to-destination delivery of packets, it does not recognize any relationship between those packets. The transport layer ensures that the whole message arrives at the host application.

- The transport layer provides a logical communication between application processes running on different hosts. Although the application processes on different hosts are not physically connected, application processes use the logical communication provided by the transport layer to send the messages to each other.


Processes
Processes
Client
Server
Internet
Domain of network-layer protocol
Domain of transport-layer protocol

Transport Layer Functions / Services:

- **Process-to-Process Communication**

The first duty of a transport-layer protocol is to provide **process-to-process communication.** A process is an application-layer entity (running program) that uses the services of the transport layer. The network layer is responsible for communication at the computer level (host-to-host communication). A network-layer protocol can deliver the message only to the destination computer. However, this is an incomplete delivery. The message still needs to be handed to the correct process. This is where a transport-layer protocol takes over. A transport-layer protocol is responsible for delivery of the message to the appropriate process.

- **Encapsulation and Decapsulation:**

The transport layer receives the data and adds the transport-layer header. 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.

- **Segmentation and Reassembling**

A message is divided into segments; each segment contains sequence number, which enables this layer in reassembling the message. Message is reassembled correctly upon arrival at the destination and replaces packets which were lost in transmission.

Continued….

- **Multiplexing and Demultiplexing:**

Multiple packets from diverse applications are transmitted across a network needs very dedicated control mechanisms, which are found in the transport layer. The transport layer accepts packets from different processes. These packets are differentiated by their port numbers and pass them to the network layer after adding proper headers. In Demultiplexing, at the receiver's side to obtain the data coming from various processes. It receives the segments of data from the network layer and delivers it to the appropriate process running on the receiver's machine.

- **Connection Control:** It includes 2 types:
  - Connectionless Transport Layer : Each segment is considered as an independent packet and delivered to the transport layer at the destination machine.
  - Connection Oriented Transport Layer : Before delivering packets, connection is made with transport layer at the destination machine.
- **Flow Control:** In this layer, flow control is performed end to end.
- **Error Control:** Error Control is performed end to end in this layer to ensure that the complete message arrives at the receiving transport layer without any error. Error Correction is done through retransmission.

Elements of Transport Protocols

- Addressing: When an application (e.g. a user) process wishes to set up a connection to a remote application process, it must specify which one to connect to. The method normally used is to define transport address to which processes can listen for connection requests. In the Internet, these end points are called ports.
- Establishing and Releasing Connection
- Flow Control and Buffering
- Error Control
- Multiplexing and Demultiplexing
- Crash Recovery: TCP is very reliable protocol. It provides sequence number to each of byte sent in segment. It provides the feedback mechanism i.e. when a host receives a packet, it is bound to ACK that packet having the next sequence number expected (if it is not the last segment). When a TCP Server crashes mid-way communication and re-starts its process it sends TPDU broadcast to all its hosts. The hosts can then send the last data segment which was never unacknowledged and carry onwards.

Connectionless and Connection-Oriented Protocols

- Transport Layer provides two types of services:

Connection Oriented Transmission: End to End connection between the senders to the receiver before sending the data over the same or multiple networks. In this type of transmission the receiving devices sends an acknowledge back to the source after a packet or group of packet is received. It is slower transmission method.

Connectionless Transmission: Transfer the data packets between senders to the receiver without creating any connection. In this type of transmission the receiving devices does not sends an acknowledge back to the source. It is faster transmission method.

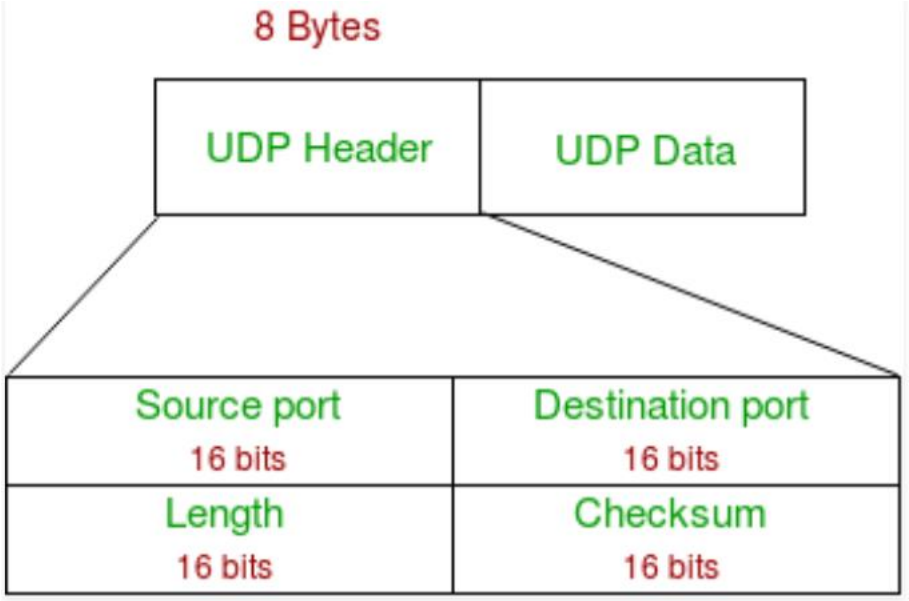
TCP	UDP
connection-oriented	connectionless
confirmed service	unconfirmed service
high overhead (header 20 bytes)	low overhead (header 8 bytes)
flow control	no flow control

User Datagram Protocol (UDP)

- The Internet protocol suites upports a connectionless transport protocol, UDP(User Datagram Protocol).
- UDP provides away for applications to send IP datagrams without having to establish a connection.
- UDP transmits segments consisting of an 8-byte header followed by the payload.
- The UDP length field includes the 8-byte header and the data.
- The **User Datagram Protocol (UDP)** is a connectionless, unreliable transport protocol.
- UDP is a very simple protocol using a minimum of overhead. If a process wants to send a small message and does not care much about reliability, it can use UDP. Sending a small message using UDP takes much less interaction between the sender and receiver than using TCP.
- UDP does not provide congestion control mechanism.
- UDP does not guarantee ordered delivery of data.

UDP Header

- UDP header is **8-bytes** fixed and simple header, while for TCP it may vary from 20 bytes to 60 bytes. First 8 Bytes contains all necessary header information and remaining part consist of data. UDP port number fields are each 16 bits long, therefore range for port numbers defined from 0 to 65535; port number 0 is reserved. Port numbers help to distinguish different user requests or process.



- UDP Destination Port: identifies destination process
- UDP Source Port: optional – identifies source process for replies, or zero
- Message Length: length of datagram in bytes, including header and data
- Checksum: optional -- 16-bit checksum over header and data, or zero

UDP Operations

UDP's only real task is to take data from higher-layer protocols and place it in UDP messages, which are then passed down to the Internet Protocol for transmission. The basic steps for transmission using UDP are:

1. **Higher-Layer Data Transfer:** An application sends a message to the UDP .

2. **UDP Message Encapsulation:** The higher-layer message is encapsulated into the *Data* field of a UDP message. The headers of the UDP message are filled in, including the *Source Port* of the application that sent the data to UDP, and the *Destination Port* of the intended recipient. The checksum value may also be calculated.

3. **Transfer Message To IP:** The UDP message is passed to IP for transmission. And that's about it. Of course, on reception at the destination device this short procedure is reversed.

UDP Properties

- UDP provides an unreliable datagram service
  - Packets may be lost or delivered out of order
  - Message split into datagrams, user sends datagrams as packets on network layer
  - No buffer at either sending or receiving side
  - Unreliable but fast
  - Full duplex
  - Application must deal with lost packets

UDP Applications

The following shows some typical applications that can benefit more from the services of UDP than from those of TCP.

- UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control. It is not usually used for a process such as FTP that needs to send bulk data.
- UDP is suitable for a process with internal flow- and error-control mechanisms. For example, the Trivial File Transfer Protocol (TFTP) process includes flow and error control. It can easily use UDP.
- UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.
- UDP is used for management processes such as SNMP .
- UDP is used for some route updating protocols such as Routing Information Protocol (RIP).
- UDP is normally used for interactive real-time applications that cannot tolerate uneven delay between sections of a received message .

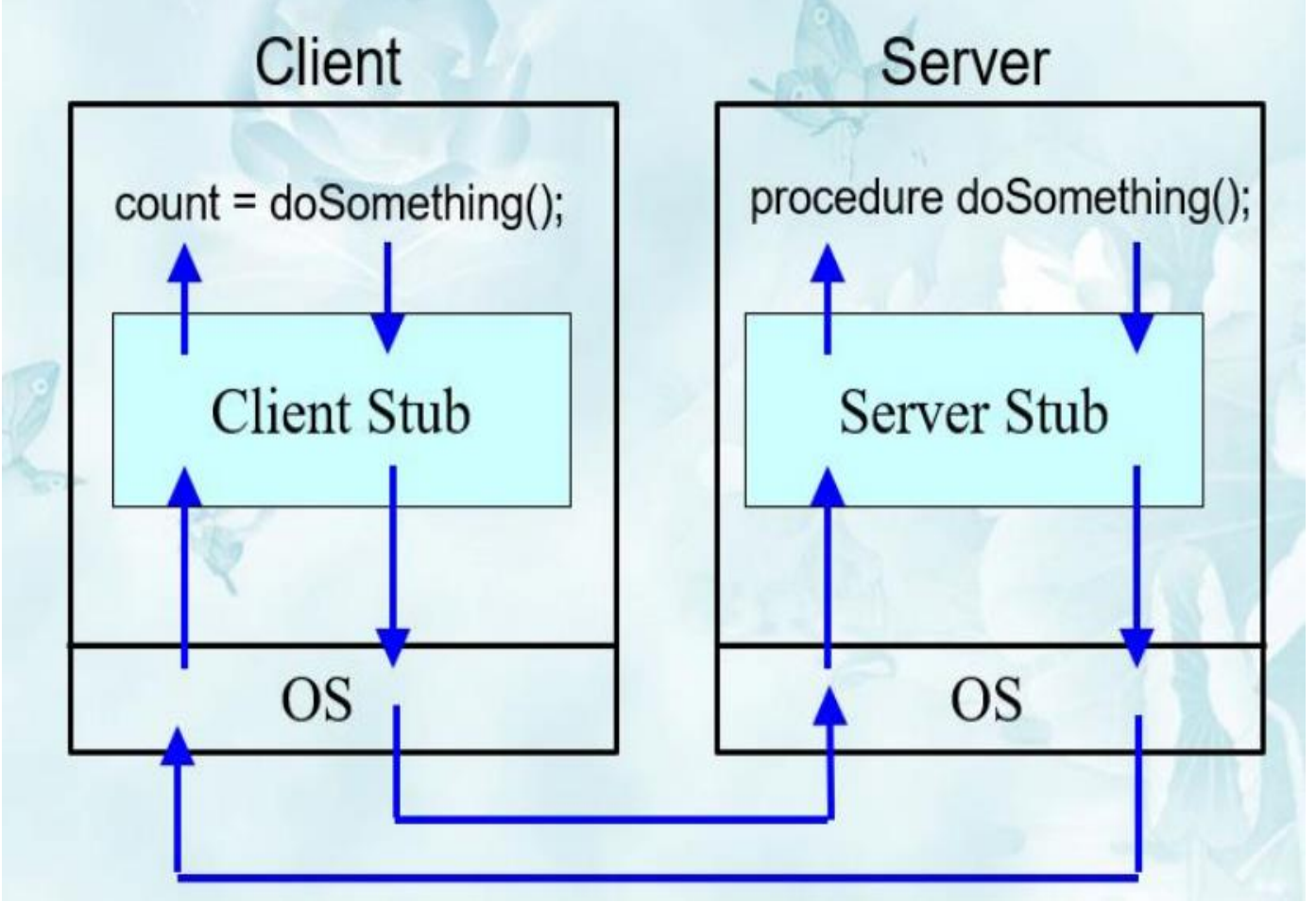
RPC (Remote Procedure Call)

- Remote Procedure Call is a software communication protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network's details. RPC is used to call other processes on the remote systems like a local system.
- We want RPC to be transparent-the calling procedure should not be aware that the called procedure is executing on a different machine or vice versa.
- RPC is a request-response protocol. An RPC is initiated by the client, which sends a request message to a known remote server to execute a specified procedure with supplied parameters. The remote server sends a response to the client, and the application continues its process. While the server is processing the call, the client is blocked (it waits until the server has finished processing before resuming execution).

- In terms of transport layer protocols, UDP is a good base on which to implement RPC. Both requests and replies may be sent as a single UDP packet in the simplest case and the operation can be fast. However, an implementation must include other machinery as well. Because the request or the reply may be lost, the client must keep a timer to retransmit the request.
- For example, just imagine a procedure named *get IP address* (*host name*) that works by sending a UDP packet to a DNS server and waiting for the reply, timing out and trying again if one is not forthcoming quickly enough.
- Generally, RPC applications will use UDP when sending data, and only fall back to TCP when the data to be transferred doesn't fit into a single UDP datagram.

Sequence of events:

1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

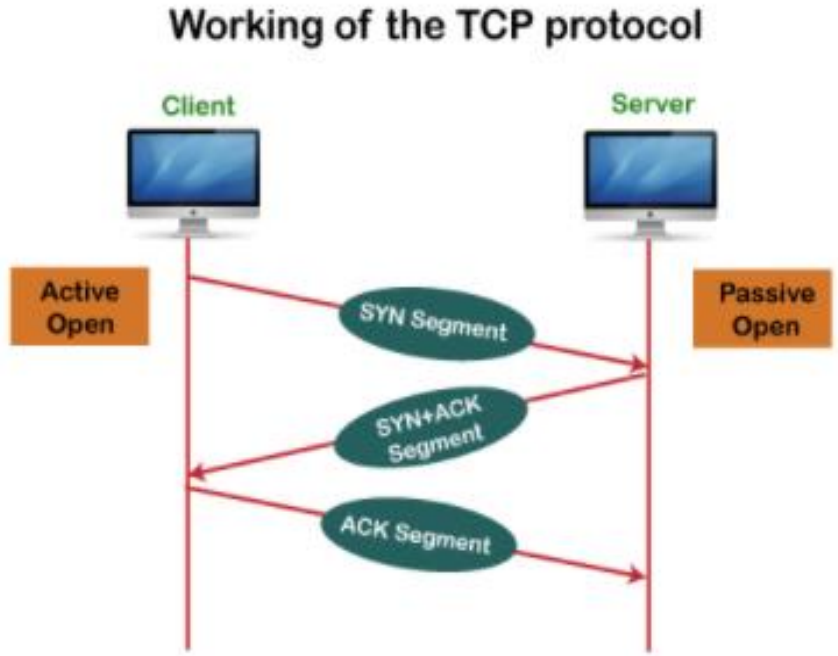


Transmission Control Protocol (TCP)

- **Transmission Control Protocol (TCP)** is a connection-oriented, reliable protocol.
- TCP explicitly defines connection establishment, data transfer, and connection teardown phases to provide a connection-oriented service.
- TCP uses a combination of Go Back-N and Selective Repeat protocols to provide reliability. To achieve this goal, TCP uses checksum (for error detection), retransmission of lost or corrupted packets, cumulative and selective acknowledgments, and timers.
- TCP organizes data so that it can be transmitted between a server and a client. It guarantees the integrity of the data being communicated over a network. Before it transmits data, TCP establishes a connection between a source and its destination, which it ensures remains live until communication begins. It then breaks large amounts of data into smaller packets(segments), while ensuring data integrity is in place throughout the process.

How does TCP work?

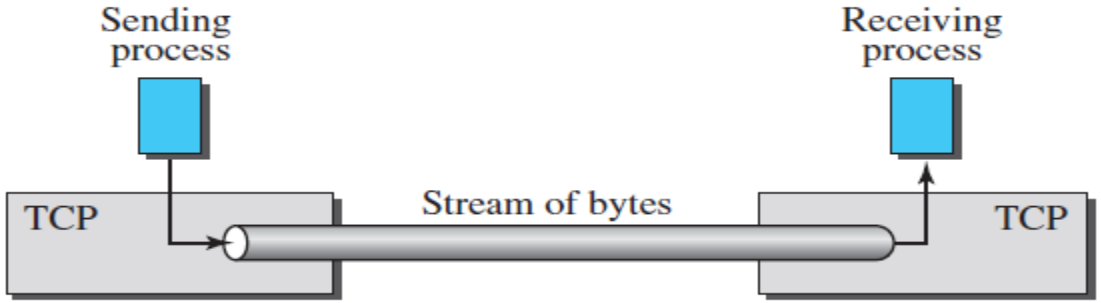
- In TCP, the connection is established by using three-way handshaking. The client sends the segment with its sequence number. The server, in return, sends its segment with its own sequence number as well as the acknowledgement sequence, which is one more than the client sequence number. When the client receives the acknowledgment of its segment, then it sends the acknowledgment to the server. In this way, the connection is established between the client and the server.



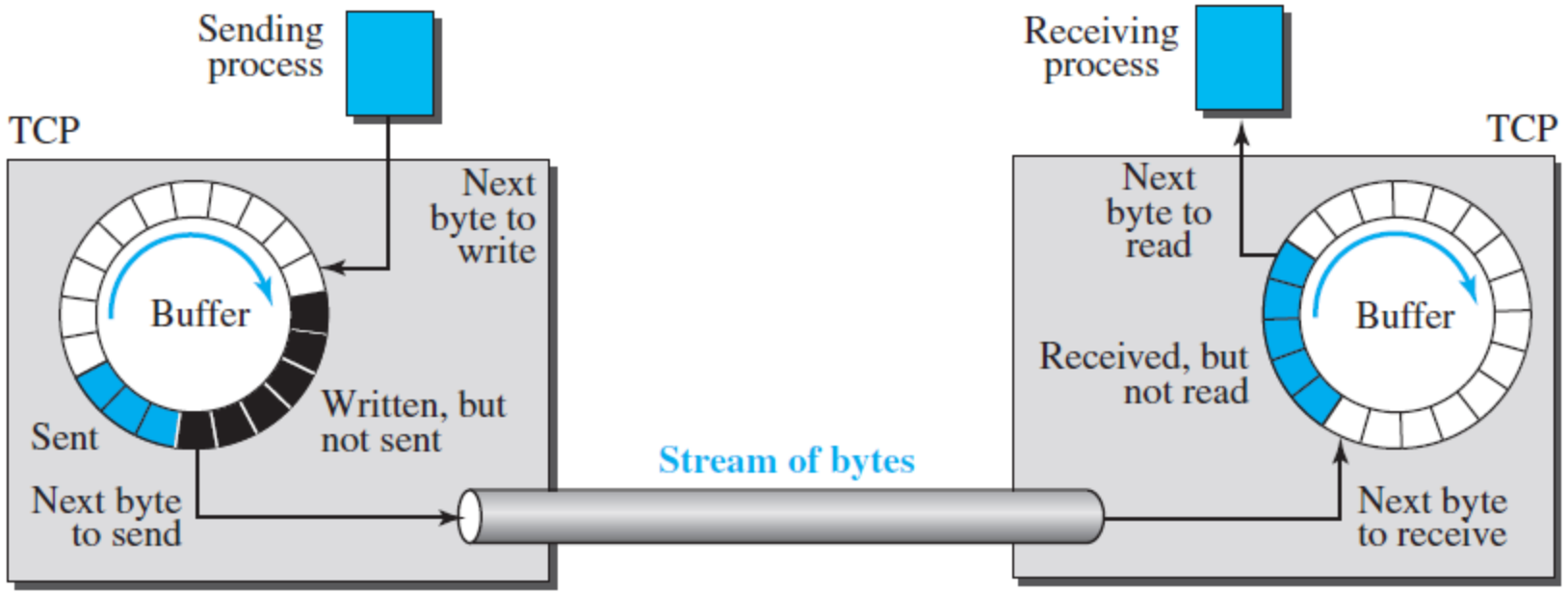
TCP Services

- ***Process-to-Process Communication:*** As with UDP, TCP provides process-to-process communication using port numbers.

- ***Stream Delivery Service:*** TCP, unlike UDP, is a stream-oriented protocol. In UDP, a process sends messages with predefined boundaries to UDP for delivery. UDP adds its own header to each of these messages and delivers it to IP for transmission. Each message from the process is called a *user datagram*, and becomes, eventually, one IP datagram. Neither IP nor UDP recognizes any relationship between the datagrams. TCP, on the other hand, allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes. TCP creates an environment in which the two processes seem to be connected by an imaginary "tube" that carries their bytes across the Internet.



- ***Sending and Receiving Buffers:*** Because the sending and the receiving processes may not necessarily write or read data at the same rate, TCP needs buffers for storage. There are two buffers, the sending buffer and the receiving buffer, one for each direction.



- ***Segments:*** At the transport layer, TCP groups a number of bytes together into a packet called a *segment*. TCP adds a header to each segment (for control purposes) and delivers the segment to the network layer for transmission. The segments are encapsulated in an IP datagram and transmitted.

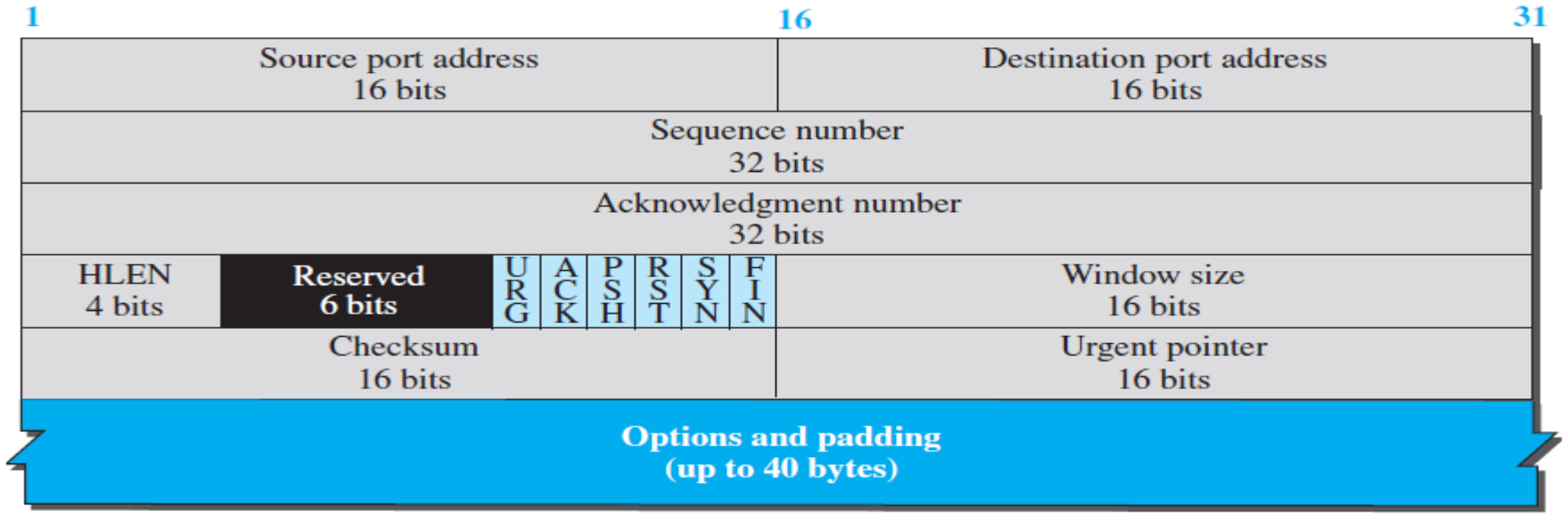
- ***Full-Duplex Communication:*** TCP offers *full-duplex service*, where data can flow in both directions at the same time. Each TCP endpoint then has its own sending and receiving buffer, and segments move in both directions.
- ***Multiplexing and Demultiplexing:*** Like UDP, TCP performs multiplexing at the sender and demultiplexing at the receiver. However, since TCP is a connection-oriented protocol, a connection needs to be established for each pair of processes.
- ***Connection-Oriented Service:*** TCP, unlike UDP, is a connection-oriented protocol. When a process at site A wants to send to and receive data from another process at site B, the following three phases occur:

1. The two TCP's establish a logical connection between them.
2. Data are exchanged in both directions.
3. The connection is terminated.

- ***Reliable Service:*** TCP is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe and sound arrival of data. We will discuss this feature further in the section on error control.

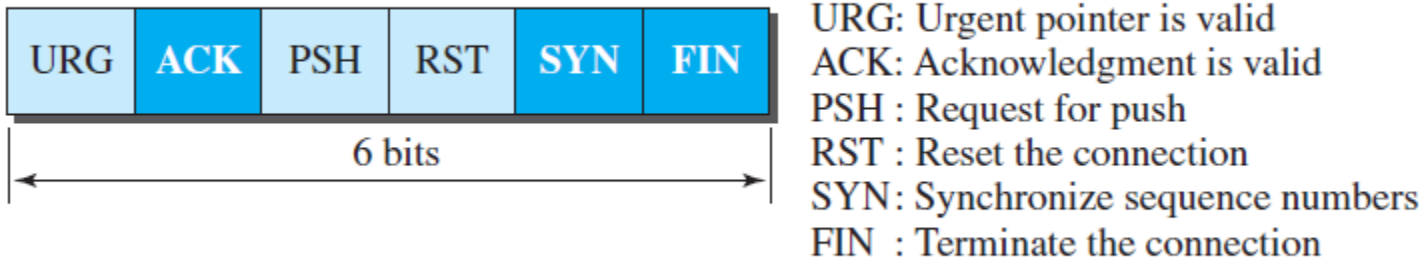
TCP Header

a. Segment



b. Header

- ✓ ***Source port address.*** This is a 16-bit field that defines the port number of the application program in the host that is sending the segment.
- ✓ ***Destination port address.*** This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment.
- ✓ ***Sequence number.*** This 32-bit field defines the number assigned to the first byte of data contained in this segment.
- ✓ ***Acknowledgment number.*** This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party.
- ✓ ***Header length.*** This 4-bit field indicates the number of 4-byte words in the TCP header.
- ✓ ***Control.*** This field defines 6 different control bits or flags. One or more of these bits can be set at a time. These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.

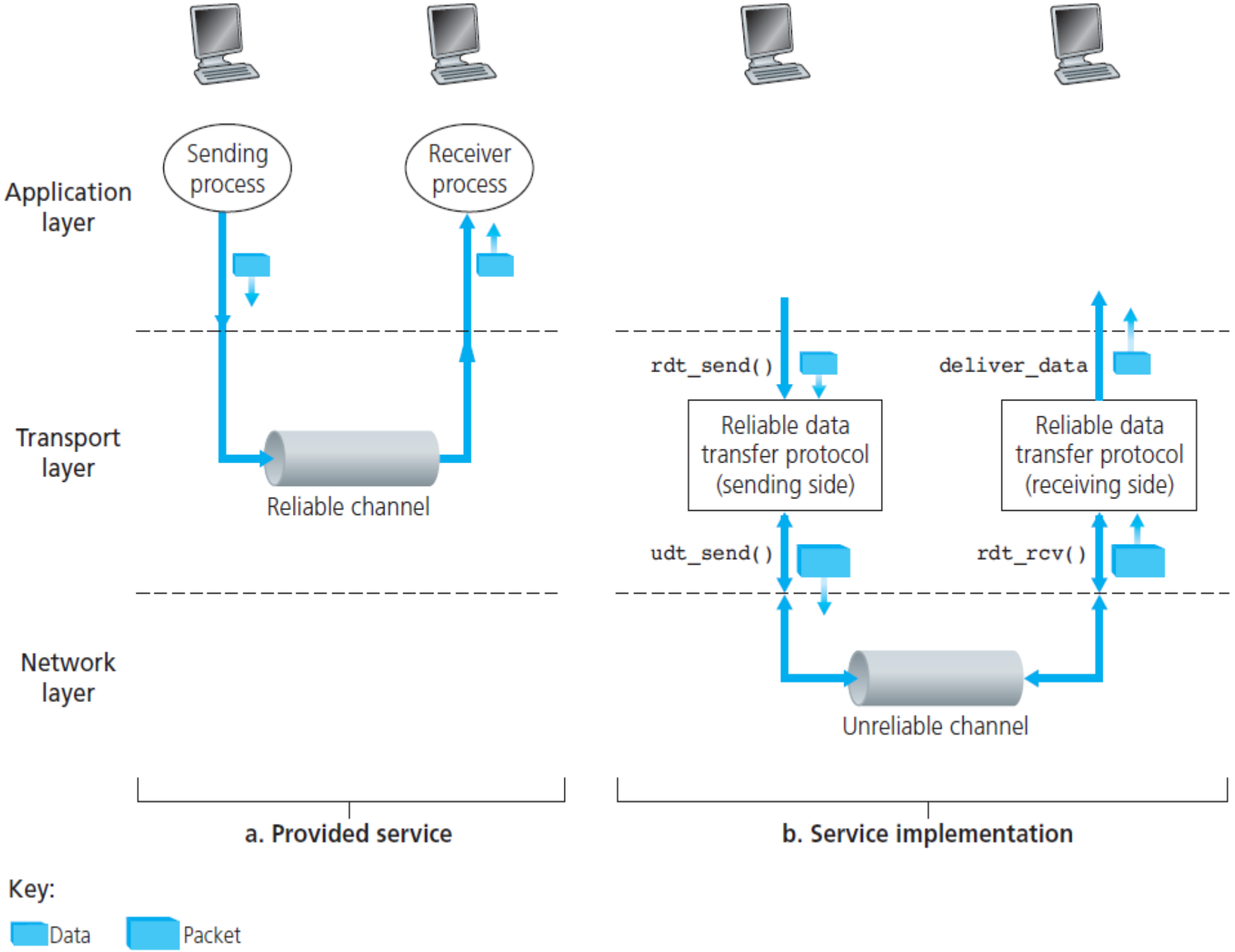


- ✓ ***Window size.*** This field defines the window size of the sending TCP in bytes.
- ✓ ***Checksum.*** This 16-bit field contains the checksum. The calculation of the checksum for TCP follows the same procedure as the one described for UDP. However, the use of the checksum in the UDP datagram is optional, whereas the use of the checksum for TCP is mandatory.

- ***Encapsulation:*** A TCP segment encapsulates the data received from the application layer. The TCP segment is encapsulated in an IP datagram, which in turn is encapsulated in a frame at the data-link layer.

Principles of Reliable Data Transfer

- Transport Layer Protocols are central piece of layered architectures, these provides the logical communication between application processes. These processes uses the logical communication to transfer data from transport layer to network layer and this transfer of data should be reliable and secure.
- The problem of transferring the data occurs not only at the transport layer, but also at the application layer as well as in the link layer. This problem occur when a reliable service runs on an unreliable service, For example, TCP (Transmission Control Protocol) is a reliable data transfer protocol that is implemented on top of an unreliable layer, i.e., Internet Protocol (IP) is an end to end network layer protocol

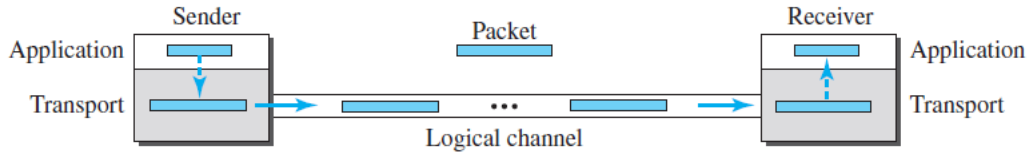


- Characteristics of unreliable channel will determine the complexity of reliable data transfer protocol (rdt).

1.Simple Protocol

- simple connectionless protocol with neither flow nor error control
- the receiver can immediately handle any packet it receives
- 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.
- No bit errors and, No loss of packets

Figure 23.17 *Simple protocol*



2. Stop-and-Wait Protocol

- 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 the packet was either lost or corrupted. This means that the sender needs to keep a copy of the packet until its acknowledgment arrives.
- To prevent duplicate packets, the protocol uses sequence numbers and acknowledgment numbers.

3. Pipelined Reliable Data Transfer Protocols

- Rather than operate in a stop-and-wait manner, if the protocol would allow a sender to send multiple packet without waiting for the acknowledgement, the performance could drastically improve. This technique is called **pipelining** and require the protocol to increase the sequence number, buffer more than just one packet.
- Thus, a pipelined protocol can have better performance than the Stop-and-Wait protocol.
- If the sender is allowed to transmit three packets before having to wait for acknowledgments, the utilization of the sender is essentially tripled. Since the many in-transit sender-to-receiver packets can be visualized as filling a pipeline, this technique is known as **pipelining**
- Pipelining has the following consequences for reliable data transfer protocols:

a) The range of sequence numbers must be increased, since each in-transit packet (not counting retransmissions) must have a unique sequence number and there may be multiple, in-transit, unacknowledged packets.

b) The sender and receiver sides of the protocols may have to buffer more than one packet.

Two basic pipelined protocols, "Go back *n*" and "Selective Repeat" acknowledge individual packets, but differ in the number of timers that they use.

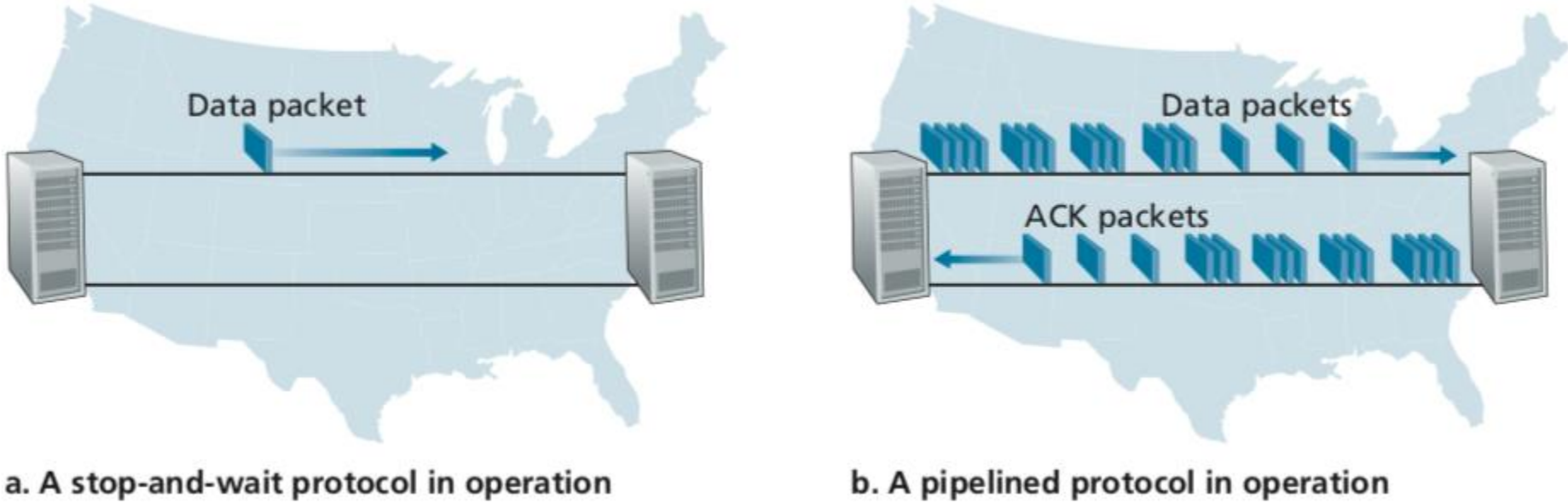
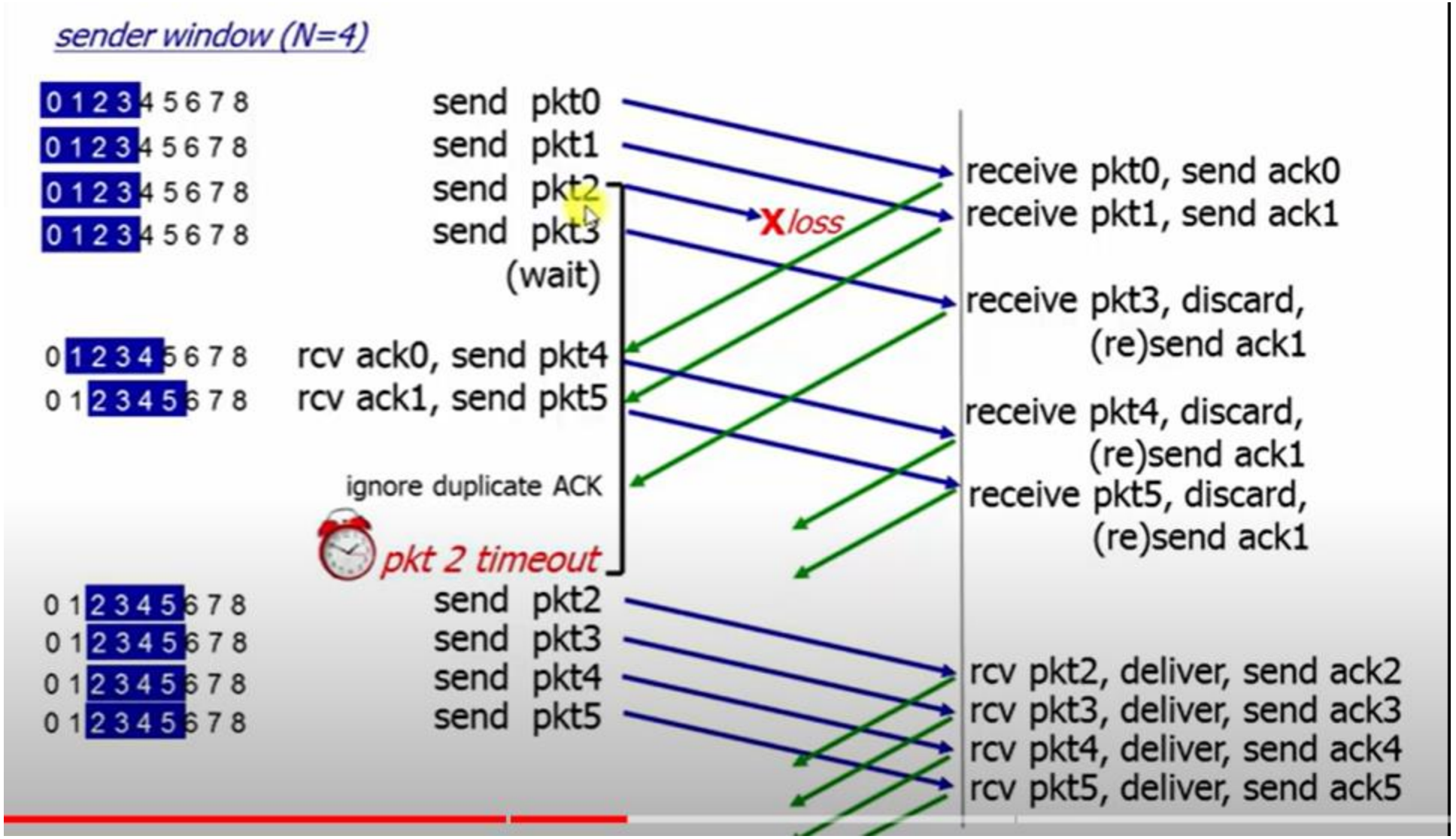


Fig: Stop and Wait vs Pipelined Protocol

3.1 Go-Back-N (GBN) Pipeline Protocol:

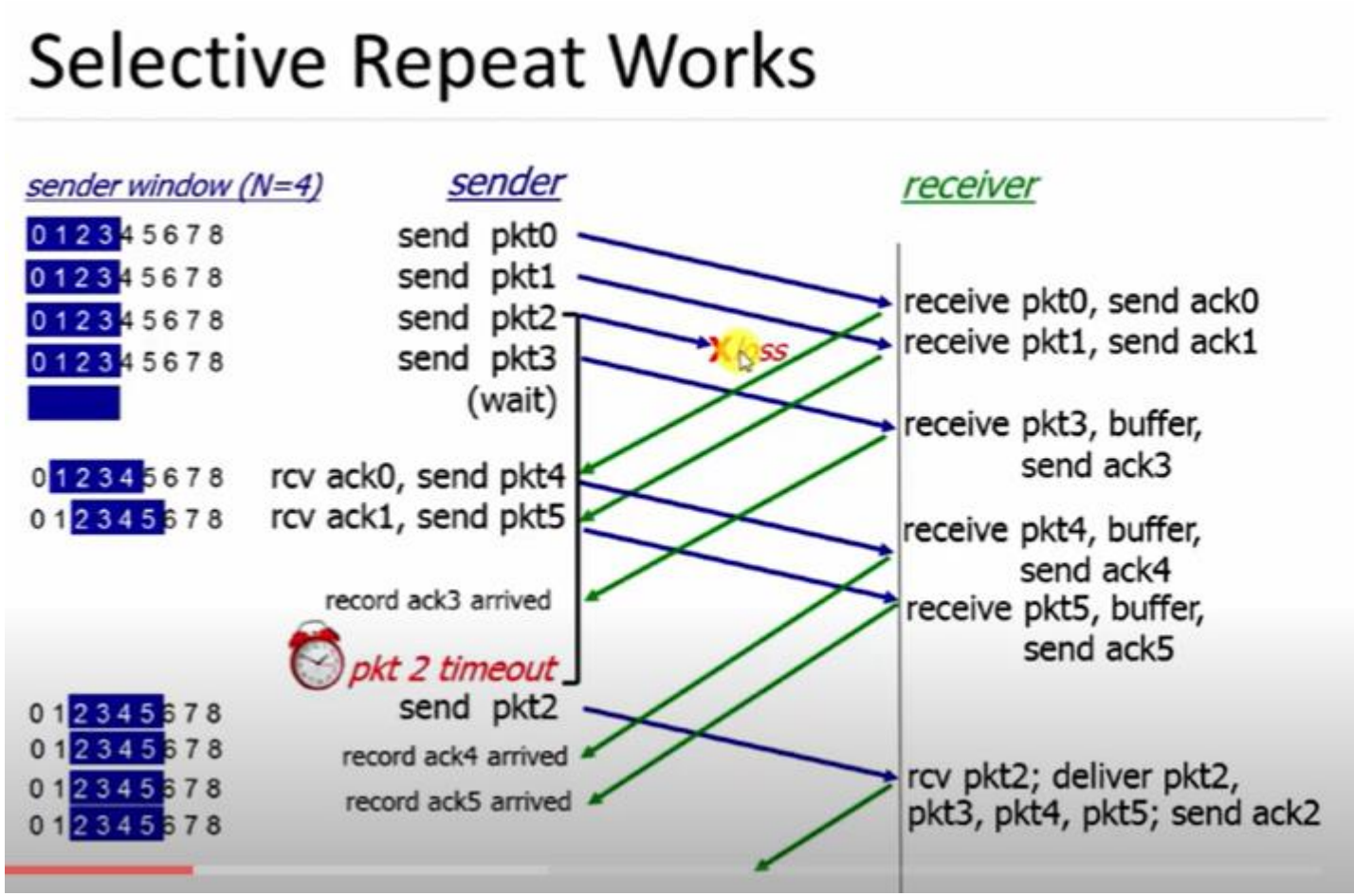
- In a **Go-Back-N (GBN) protocol**, the sender is allowed to transmit multiple packets without waiting for an acknowledgment, but is constrained to have no more than some maximum allowable number, *N*, of unacknowledged packets in the pipeline.
- Receiver sends only cumulative acknowledgement. It does not ACK packet if there is a gap.
- Sender has timer for oldest un-acked packet. When time expires, retransmit all un-acked packets.
- In Go-Back-N , **N** is the sender's window size. Suppose we say that Go-Back-3, which means that the three frames can be sent at a time before expecting the acknowledgment from the receiver.
- Receiver keep track of the sequence number of the next packet it expects to receive, and sends that number with every ACK it sends.



Go-Back-N Operations

3.2 Selective Repeat Pipeline Protocol

- A sender can have N unacknowledged packet in pipeline.
- Receiver sends individual ACK for each packet.
- Sender maintains timer for each UNACKED packet. When time expires , retransmit only that UNACKED packet.
- Selective Repeat attempts to retransmit only those packets that are actually lost due to errors.
- Receiver must be able to receive packets out of order.



Principle of Congestion Control:

- Too many packets present in (a part of) the network causes packet delay and loss that degrades performance. This situation is called congestion.
- In other words, congestion in a network may occur if the load on the network, the number of packets sent to the network (the number of packets a network can handle) is greater than the capacity of the network
- The network and transport layers share the responsibility for handling congestion. Since congestion occurs within the network, it is the network layer that directly experiences it and must ultimately determine what to do with the excess packets. However, the most effective way to control congestion is to reduce the load that the transport layer is placing on the network.
- Congestion control refers to the mechanisms and techniques to control the congestion and keep the load below the capacity.

Effects of Congestion:

- As delay increases, performance decreases.
- If delay increases, retransmission occurs, making situation even worse.

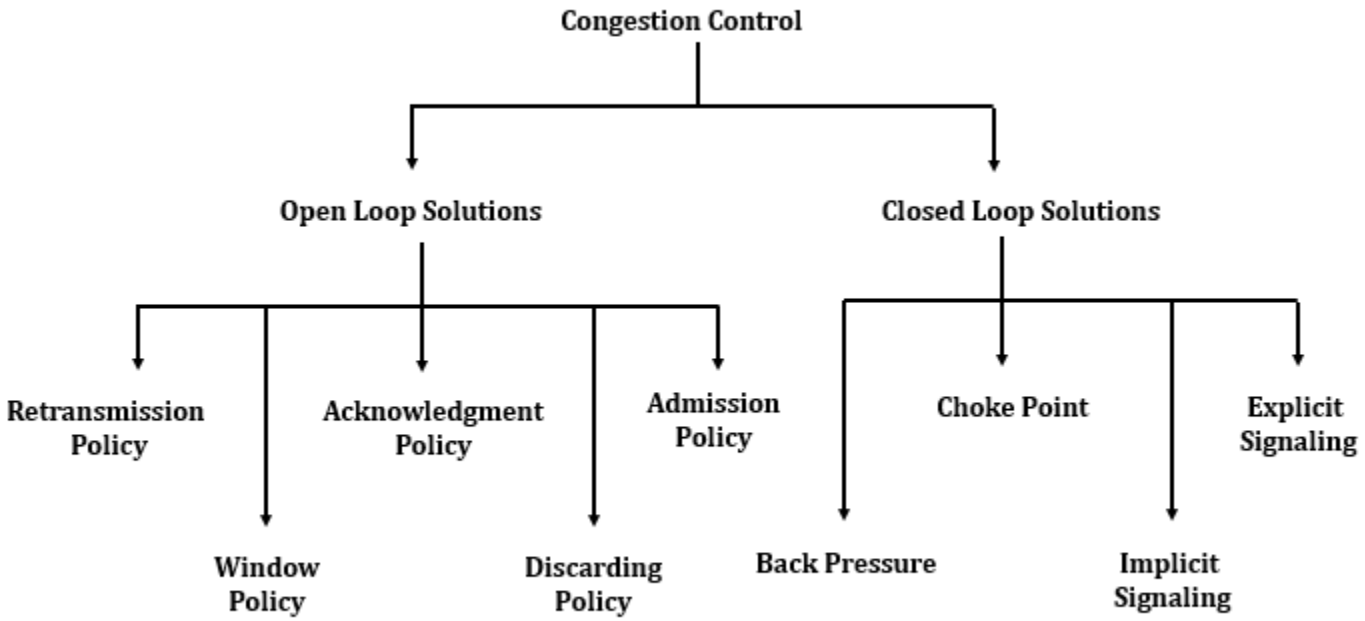
Congestion control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened.

The General Principles of Congestion Control are as follows:

- Open Loop Principle :
  - Attempt to prevent congestion from happening
  - After system running, no corrections made

- Closed Loop Principle:
  - monitor system to detect congestion
  - pass information to where action should be taken
  - adjust system operation to correct problem

Congestion Control Strategies



Open Loop Congestion Control

In open loop congestion control, policies are applied to prevent congestion before it happens. In these mechanisms, congestion control is handled by either the source or the destination. The list of policies that can prevent congestion are:

- Retransmission Policy: It is the policy in which retransmission of the packets are taken care. If the sender feels that a sent packet is lost or corrupted, the packet needs to be retransmitted. This transmission may increase the congestion in the network. To prevent congestion, retransmission timers must be designed to prevent congestion and also able to optimize efficiency.

- Window Policy: The type of window at the sender's side may also affect the congestion. Several packets in the Go-back-n window are re-sent, although some packets may be received successfully at the receiver side. This duplication may increase the congestion in the network and make it worse. Therefore, Selective repeat window should be adopted as it sends the specific packet that may have been lost.

- Acknowledgement Policy: Since acknowledgements are also the part of the load in the network, the acknowledgment policy imposed by the receiver may also affect congestion.. The receiver should send acknowledgement for N packets rather than sending acknowledgement for a single packet. The receiver should send an acknowledgment only if it has to send a packet or a timer expires.

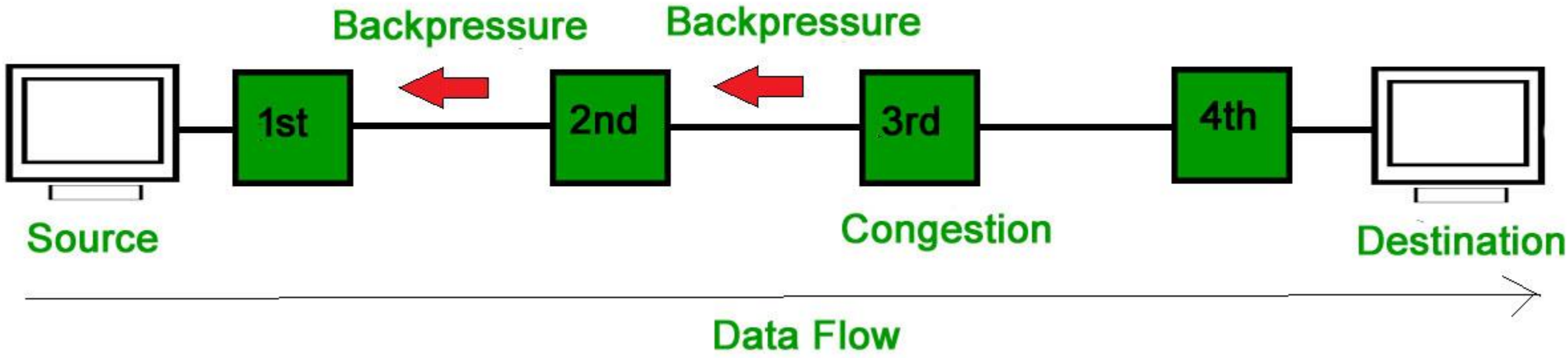
- Discarding Policy: A good discarding policy adopted by the routers is that the routers may prevent congestion and at the same time partially discard the corrupted or less sensitive packages and also be able to maintain the quality of a message. In case of audio file transmission, routers can discard less sensitive packets to prevent congestion and also maintain the quality of the audio file.

- Admission Policy: In admission policy a mechanism should be used to prevent congestion. Switches in a flow should first check the resource requirement of a network flow before transmitting it further. If there is a chance of a congestion or there is a congestion in the network, router should deny establishing a virtual network connection to prevent further congestion.

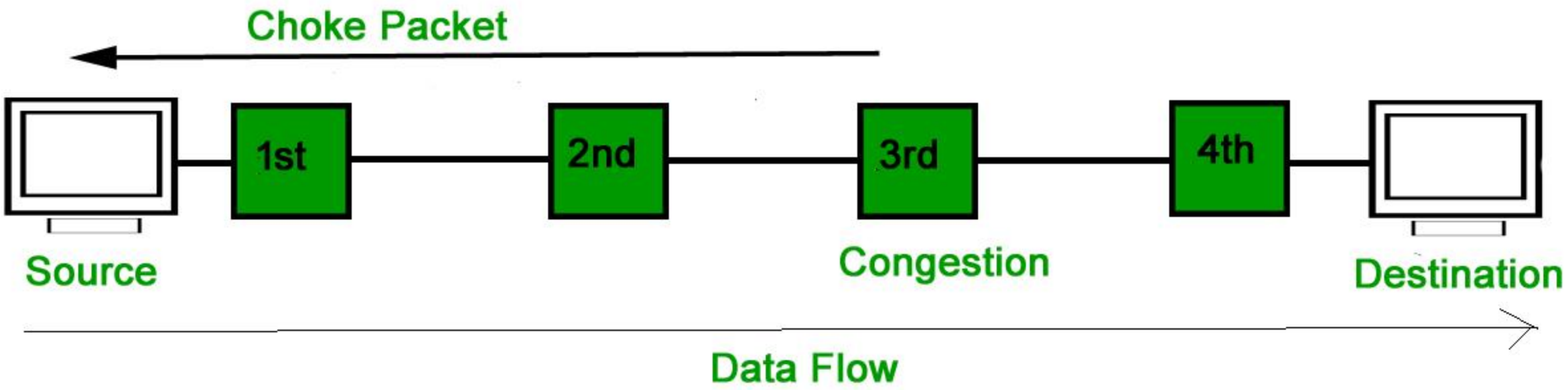
Closed Loop Congestion Control

- Closed loop congestion control techniques are used to treat or alleviate congestion after it happens. Several techniques are used by different protocols; some of them are:

1. BackPressure: Backpressure is a node-to-node congestion control technique that propagate in the opposite direction of data flow.



2. Choke Packet: A choke packet is a packet sent by a node to the source to inform it of congestion. Each router monitors its resources and the utilization at each of its output lines. Whenever the resource utilization exceeds the threshold value which is set by the administrator, the router directly sends a choke packet to the source giving it a feedback to reduce the traffic.

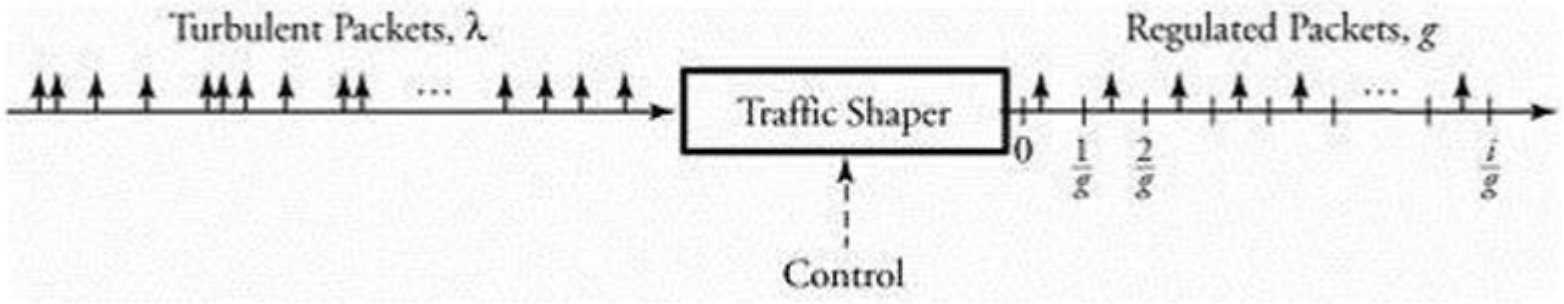


3. Implicit Signaling: In implicit signaling, there is no communication between the congested nodes and the source. The source guesses that there is congestion in a network. For example when sender sends several packets and there is no acknowledgment for a while, one assumption is that there is a congestion.

4. Explicit signaling: In explicit signaling, if a node experiences congestion it can explicitly sends a packet to the source or destination to inform about congestion. The difference between choke packet and explicit signaling is that the signal is included in the packets that carry data rather than creating a different packet as in case of choke packet technique.

Traffic Shaping

- **Traffic Shaping** is a mechanism to control the amount and the rate of traffic sent to the network. Approach of congestion management is called Traffic shaping. Traffic shaping helps to regulate the rate of data transmission and reduces congestion.



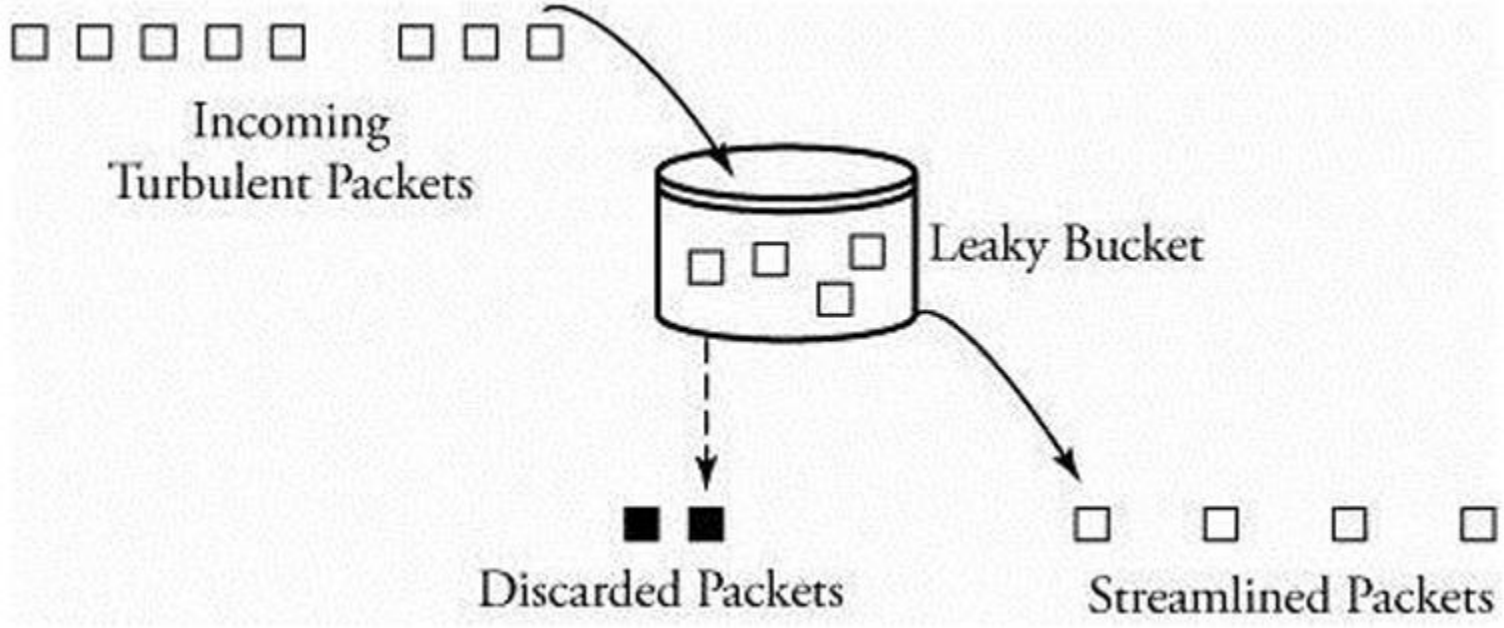
- There are 2 types of traffic shaping algorithms:

1. Leaky Bucket
2. Token Bucket

Leaky-Bucket Traffic Shaping

- This algorithm converts any turbulent incoming traffic into a smooth, regular stream of packets

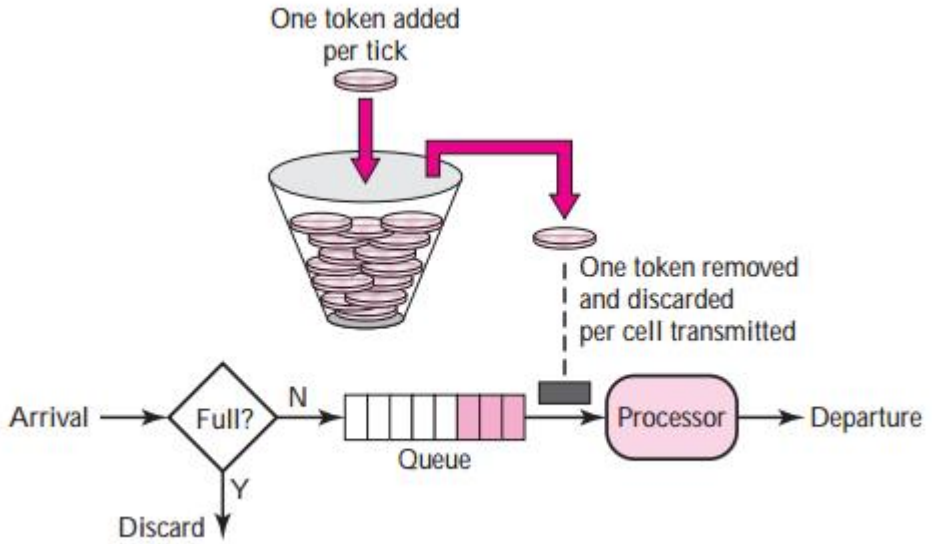
- A leaky-bucket interface is connected between a packet transmitter and the network. No matter at what rate packets enter the traffic shaper, the outflow is regulated at a constant rate, much like the flow of water from a leaky bucket.



- When a packet arrives, the interface decides whether that packet should be queued or discarded, depending on the capacity of the buffer.
- Packets are transmitted as either fixed-size packets or variable-size packets. In the fixed-size packet environment, a packet is transmitted at each clock tick. In the variable-size packet environment, a fixed-sized block of a packet is transmitted.

Token Bucket Traffic Shaping

- It allows bursty traffic at a regulated maximum rate.
- For a packet to be transmitted, it must capture and destroy one token. The token bucket algorithm allows idle hosts to save up permission to the maximum size of bucket n for burst traffic latter.

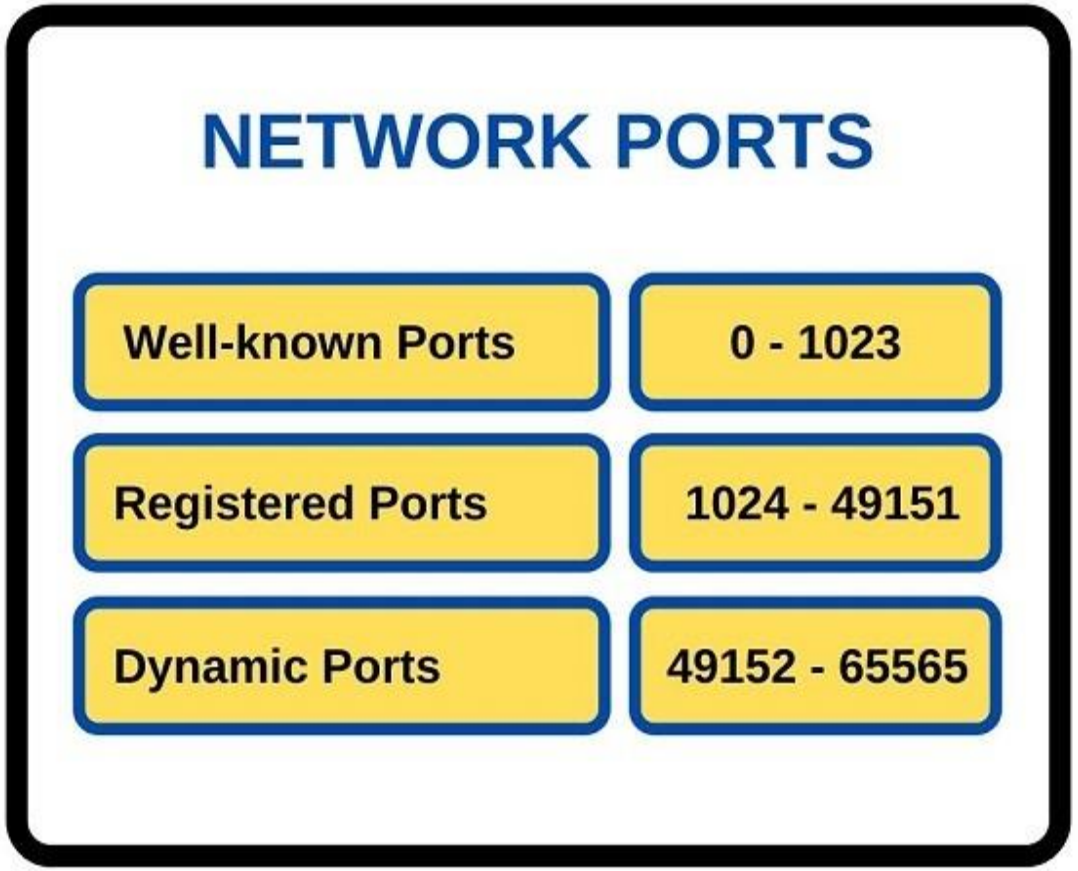


Leaky Bucket vs Token Bucket

Parameter	Leaky Bucket	Token Bucket
Token Dependency	Token independent.	Dependent on Token.
Filled bucket for token	When bucket is full, data or packets are discarded.	If bucket is full, token are discard not packets.
Packet transmission	Leaky bucket sends packets at constant rate.	Token bucket can send large burst of packets at faster rate.
Condition for packet transmission	In Leaky bucket algorithm, Packets are transmitted continuously.	In Token bucket algorithm, Packets can only transmit when there is enough token.
Token saving	It does not save any token.	It saves token for the burst of packet transmission.
Restrictive Algorithm	Leaky bucket algorithm is more restrictive as compared to Token bucket algorithm.	Token bucket algorithm is less restrictive as compared to Leaky bucket algorithm.

Some Well Known Port Address:

Port Address: 16 bits



Port #	Application Layer Protocol	Type	Description
20	FTP	TCP	File Transfer Protocol - data
21	FTP	TCP	File Transfer Protocol - control
22	SSH	TCP/UDP	Secure Shell for secure login
23	Telnet	TCP	Unencrypted login
25	SMTP	TCP	Simple Mail Transfer Protocol
53	DNS	TCP/UDP	Domain Name Server
67/68	DHCP	UDP	Dynamic Host
80	HTTP	TCP	HyperText Transfer Protocol
123	NTP	UDP	Network Time Protocol
161,162	SNMP	TCP/UDP	Simple Network Management Protocol
389	LDAP	TCP/UDP	Lightweight Directory Authentication Protocol
443	HTTPS	TCP/UDP	HTTP with Secure Socket Layer